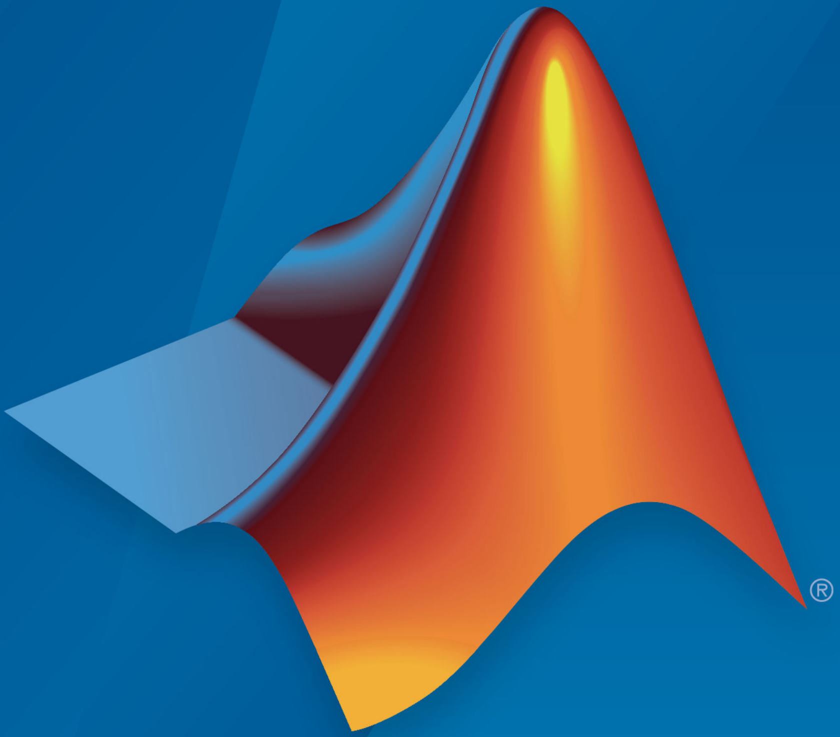
Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors

Reference



MATLAB®&SIMULINK®

R2019b

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Embedded Coder® Support Package for Texas Instruments™ C2000™ Processors Reference

Revision History

October 2014	Online only	Revised for Version 14.2.0 (R2014b)
March 2015	Online only	Revised for Version 15.1.0 (R2015a)
September 2015	Online only	Revised for Version 15.2.0 (R2015b)
November 2015	Online only	Rereleased for Version 15.2.2 (R2015b)
March 2016	Online only	Revised for Version 16.1.0 (R2016a)
September 2016	Online only	Revised for Version 16.2.0 (R2016b)
March 2017	Online only	Revised for Version 17.1.0 (R2017a)
September 2017	Online only	Revised for Version 17.2.0 (R2017b)
March 2018	Online only	Revised for Version 18.1.0 (R2018a)
September 2018	Online only	Revised for Version 18.2.0 (R2018b)
October 2018	Online only	Revised for Version 18.2.1 (R2018b)
March 2019	Online only	Revised for Version 19.1.0 (R2019a)
September 2019	Online only	Revised for Version 19.2.0 (R2019b)
October 2019	Online only	Revised for Version 19.2.1 (R2019b)
January 2020	Online only	Revised for Version 19.2.2 (R2019b)

Contents

1 Configuration Parameters

Configuration Parameters

Hardware Implementation Pane: Texas Instruments C2000 Processors

To configure hardware parameters for Texas Instruments C2000 processors:

1. In the Simulink® Editor, select **Simulation** > **Model Configuration Parameters**.
2. In the Configuration Parameter dialog box, click **Hardware Implementation**.
3. Set the **Hardware board** parameter to your C2000 processor.
4. The parameter values under **Hardware board settings** are automatically populated to their default values.

   You can optionally adjust these parameters for your particular use case.
5. Click **Apply**.

Note In the **Hardware board** drop-down list, some processors have multiple options. Select the generic option for controlCARDs and custom boards, and select the LaunchPad option for LaunchPads. For example, select **TI Delfino F2837xS** as the generic option, and select **TI Delfino F28377S Launchpad** as the LaunchPad option. Based on your selection, the default values for clock settings, pin selection, and memory mapping change.

Hardware Board Settings

For each hardware board you select, you can configure the board parameters according to your requirements.

Scheduler Options

Parameter	Description	Default Value
Base rate trigger on page 1-37	Set the static priority of the base rate task in the operating system.	`Timer 0`

Build Options

Parameter	Description	Default Value
Build action on page 1-38	Define how Embedded Coder responds when you build your model.	`Build, load, and run`
Device name on page 1-38	Select your device from the selected processor family.	
Enable TMU on page 1-38	Enables support for Trigonometric Math Unit (TMU)	`enabled`
Select CPU on page 1-38	Select a CPU core to run the generated code on a dual-core processor, such as F2837xD.	
Boot From Flash (stand alone execution) on page 1-38	Specify if the application loads to the flash memory.	`enabled`
Use custom linker command file on page 1-38	Indicates that the custom linker command file must be used during the build action.	`enabled`
Linker command file on page 1-38	The path to the memory description file required during linking.	
CCS hardware configuration file on page 1-38	The Code Composer Studio™ file required for downloading the application on the hardware.	
Enable DMA to access ePWM Registers instead of CLA on page 1-38	Select to access ePWM Registers	
Enable DMA to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC,CMPSS, and SDFM) instead of CLA on page 1-38	Select to enable the DMA to access peripheral frame 1	

Parameter	Description	Default Value
Enable DMA to peripheral frame 2 (SPI and McBSP) instead of CLA on page 1-38	Select to enable the DMA to access peripheral frame 2	
Enable FastRTS on page 1-38	Enables use of optimized floating point math functions from C28x FPU fastRTS library	enabled
Remap ePWMs for DMA access (Requires silicon revision A and above) on page 1-38	Select to remap ePWMs registers for DMA access	

Clocking

Parameter	Description	Default Value
Desired CPU Clock in MHz on page 1-42	Specify the desired CPU clock frequency (CLKIN).	
Use internal oscillator on page 1-42	Use the internal zero pin oscillator on the CPU.	`enabled`
Oscillator clock (OSCCLK) frequency in MHz on page 1-42	Oscillator frequency used in the processor.	
Auto set PLL based on OSCCLK and CPU clock on page 1-42	PLL values in PLLCR, DIVSEL, and **Achievable SYSCLKOUT in MHz** are automatically calculated based on the CPU clock entered on the board.	
PLL control register (PLLCR) on page 1-42	If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto-calculated control register value matches the specified CPU clock value, based on the oscillator clock frequency.	
PLL output divider (ODIV) on page 1-42	Calculates SYSCLKOUT = ((OSCCLK×SYSPLLMULT)/ODIV)/SYSDIVSEL.	
Clock divider (DIVSEL) on page 1-42	If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto-calculated control register value matches the specified CPU clock value, based on the oscillator clock frequency.	

Parameter	Description	Default Value
Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/ DIVSEL on page 1-42	The auto-calculated feedback value that matches the **Desired C28x CPU clock in MHz** value, based on the values of OSCCLK, PLLCR, and DIVSEL.	
Set the 'Achievable SYSCLKOUT in MHz = (OSCCLK*SYSPLLMULT)/ SYSDIVSEL' value calculated in CPU1 on page 1-42	Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Achievable SYSCLKOUT in MHz = (OSCCLK*PLLCR)/DIVSEL** (auto calculated).	
Select the 'Low-Speed Peripheral Clock Prescaler (LSPCLK)' option used in CPU1 on page 1-42	Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Low-Speed Peripheral Clock Prescaler (LSPCLK)** specified in CPU1.	
Low-Speed Peripheral Clock Prescaler (LSPCLK) on page 1-42	Prescaler value used to calculate LSPCLK based on SYSCLKOUT.	
Low-Speed Peripheral Clock (LSPCLK) in MHz on page 1-42	The LSPCLK value calculated using the SYSCLKOUT and LSPCLK Prescaler values.	
High-Speed Peripheral Clock Prescaler (HSPCLK) on page 1-42	Prescaler value used to calculate HSPCLK based on SYSCLKOUT.	

Parameter	Description	Default Value
High-Speed Peripheral Clock (HSPCLK) in MHz on page 1-42	The HSPCLK value calculated using the SYSCLKOUT and HSPCLK Prescaler values.	
Analog Subsystem Clock Prescaler (ASYSCLK) on page 1-42	Prescaler value used to calculate ASYSCLK based on SYSCLKOUT.	
Analog Subsystem Clock (ASYSCLK) in MHz on page 1-42	The ASYSCLK value calculated using the SYSCLKOUT and ASYSCLK Prescaler values.	

ADC_x

Parameter	Description	Default Value
Select the CPU core which controls ADC_x module on page 1-46	The CPU core that controls the ADC module.	
ADC clock prescaler (ADCCLK) on page 1-46	The ADCCLK divider for the c2802x, c2803x, c2806x, F28M3x, F2807x, or F2837x processor.	
ADC clock frequency in MHz on page 1-46	The clock frequency for ADC, which is auto generated based on the value you select in **ADC clock prescaler (ADCCLK)**.	
ADC overlap of sample and conversion (ADC#NONOVERLAP) on page 1-46	Enable or disable overlap of sample and conversion.	
ADC clock prescaler (ADCLKPS) on page 1-46	The HSPCLK is divided by ADCLKPS (a 4-bit value) as the first step in deriving the core clock speed of the ADC.	3
ADC Core clock prescaler (CPS) on page 1-46	After dividing the HSPCLK speed by the **ADC clock prescaler (ADCLKPS)** value, divides the result by 2.	1
ADC Module clock (ADCCLK = HSPCLK/ADCLKPS×2)/(CPS+1)) in MHz on page 1-46	The ADC module clock, which indicates the ADC operating clock speed.	

Parameter	Description	Default Value
Acquisition window prescaler (ACQ_PS) on page 1-46	Determine the width of the sampling or acquisition period. A higher value indicates a wider sampling period.	4
Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel on page 1-46	Determine the duration for which the sampling switch is closed.	
Offset on page 1-46	Specify the offset value.	
Use external reference 2.048V on page 1-46	Allows using a 2.048 V external voltage reference.	
Use external reference on page 1-46	Allows using an external voltage reference.	
Continuous mode on page 1-46	When the ADC generates an end of conversion (EOC) signal, an ADCINT# interrupt is generated. The interrupt indicates whether the previous interrupt flag has been acknowledged.	
ADC offset correction (OFFSET_TRIM: –256 to 255) on page 1-46	The 280x ADC supports offset correction using a 9-bit value that it adds or subtracts before the results are available in the ADC result registers.	0

Parameter	Description	Default Value
VREFHI, VREFLO on page 1-46	When you disable the **Use external reference 2.048V** or **External reference** option, the ADC logic uses a fixed 0–3.3 V input range, and VREFHI and VREFLO are disabled. To interpret the ADC input as a ratiometric signal, select the **External reference** option. Then, set values for the high-voltage reference (VREFHI) and the low voltage reference (VREFLO).	
INT pulse control on page 1-46	Set the time when the ADC sets ADCINTFLG ADCINTx relative to the SOC and EOC pulses.	
SOC high priority on page 1-46	Enable SOC high priority mode.	`All in round robin mode`
XINT2SOC external pin on page 1-46	The pin to which the ADC sends the XINT2SOC pulse.	
ADCEXTSOC external pin on page 1-46	The pin to which the ADC sends the ADCEXTSOC pulse.	

COMP

Parameter	Description	Default Value
Comparator x (COMPx) pin assignment on page 1-49	Assign COMP pin to a GPIO pin.	

DAC

Parameter	**Description**	**Default Value**
DACx reference voltage on page 1-50	Select the reference voltage for the DAC channel A, B, or C.	`ADC reference voltage (VREFHIA/VREFHIB)`
DACx synchronization signal on page 1-50	Select the synchronization signal to load the value from the writable shadow register into the active register.	`SYSCLK`

eCAN_x

Parameter	Description	Default Value
CAN module clock frequency (= SYSCLKOUT) in MHz on page 1-51	The clock for the enhanced CAN module.	
CAN module clock frequency (=SYSCLKOUT/2) in MHz on page 1-51	The clock for the enhanced CAN module.	
Baud rate prescaler (BRP: 2 to 256)/Baud rate prescaler (BRP: 1 to 1024) on page 1-51	Scale the bit rate using this value.	
Time segment 1 (TSEG1) on page 1-51	Set the value of time segment 1. This value, with **TSEG2** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus.	
Time segment 2 (TSEG2) on page 1-51	Set the value of time segment 2. This value, with **TSEG1** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus.	
Baud rate (CAN Module Clock/BRP/(TSEG1 + TSEG2 +1)) in bits/sec on page 1-51	CAN module communication speed represented in bits/second.	
SBG on page 1-51	Set the message resynchronization triggering.	

Parameter	Description	Default Value
SJW on page 1-51	Set the synchronization jump width, which determines how many units of TQ a bit can be shortened or lengthened by when resynchronizing.	
SAM on page 1-51	Number of samples used by the CAN module to determine the CAN bus level.	
Enhanced CAN Mode on page 1-51	Enable time stamping and usage of Mailbox Numbers 16 through 31 in the C2000 eCAN blocks.	
Self test mode on page 1-51	If you set this parameter to `True`, the eCAN module goes to loopback mode. The loopback mode sends a dummy acknowledge message back.	`False`
Pin assignment (Tx) on page 1-51	Assign the CAN transmit pin to use with the eCAN_B module.	
Pin assignment (Rx) on page 1-51	Assign the CAN receive pin to use with the eCAN_B module.	

eCAP

Parameter	Description	Default Value
ECAPx pin assignment on page 1-53	Assign eCAP pin to a GPIO pin.	

ePWM

Parameter	Description	Default Value
EPWM clock divider (EPWMCLKDIV) on page 1-54	Select the ePWM clock divider.	
Select the 'EPWM clock divider (EPWMCLKDIV)' option used for CPU1 on page 1-54	Available only for CPU2 of dual C28x core processors. Its value must be the same as the value of the parameter EPWM clock divider (EPWMCLKDIV) selected in CPU1.	
TZx pin assignment on page 1-54	Assign the trip-zone input x (TZx) to a GPIO pin.	
SYNCI pin assignment on page 1-54	Assign the ePWM external sync pulse input (SYNCI) to a GPIO pin.	
SYNCO pin assignment on page 1-54	Assign the ePWM external sync pulse output (SYNCO) to a GPIO pin.	
PWM#x pin assignment on page 1-54	Assign the GPIO pin to the PWM#x module.	
GPTRIP#SEL pin assignment(GPIO0~63) on page 1-54	Assign the ePWM trip-zone input to a GPIO pin.	
PWM1SYNCI/ GPTRIP6SEL pin assignment on page 1-54	Assign the ePWM sync pulse input (SYNCI) to a GPIO pin.	
DCxHTRIPSEL (Enter Hex value between 0 and 0x6FFF) on page 1-54	Assign the **Digital Compare A** high trip input to a GPIO pin.	
DCxLTRIPSEL (Enter Hex value between 0 and 0x6FFF) on page 1-54	Assign the **Digital Compare A** low trip input to a GPIO pin.	

I2C

Parameter	Description	Default Value
Mode on page 1-57	Configure the I2C module as `Master` or `Slave`.	
Addressing format on page 1-57	In `Slave` mode, determines the addressing format of the I2C master and sets the I2C module to the same mode.	
Own address register on page 1-57	In `Slave` mode, enter the 7-bit (0–127) or 10-bit (0–1023) address that the I2C module uses.	
Bit count on page 1-57	In `Slave` mode, sets the number of bits in each data byte the I2C module transmits and receives.	
Module clock prescaler (IPSC: 0 to 255) on page 1-57	In `Master` mode, enter a value in the range 0–255, inclusive, to configure the model clock frequency.	
I2C Module clock frequency (SYSCLKOUT / (IPSC+1)) in Hz on page 1-57	Display the frequency the I2C module uses internally. To set this value, change the **Module clock prescaler**.	
I2C Master clock frequency (Module Clock Freq/(ICCL +ICCH+10)) in Hz on page 1-57	Display the master clock frequency.	
Master clock Low-time divider (ICCL: 1 to 65535) on page 1-57	In `Master` mode, determines the duration of the low state of the SCL on the I2C bus.	

Parameter	Description	Default Value
Master clock High-time divider (ICCH: 1 to 65535) on page 1-57	In `Master` mode, determines the duration of the high state of the SCL on the I2C bus.	
Enable loopback on page 1-57	In `Master` mode, enables or disables digital loopback mode.	
SDA pin assignment on page 1-57	Select a GPIO pin as an I2C data bidirectional port.	
SCL pin assignment on page 1-57	Select a GPIO pin as an I2C clock bidirectional port.	
Enable Tx interrupt on page 1-57	This parameter corresponds to bit 5 (TXFFIENA) of the I2C Transmit FIFO Register (I2CFFTX).	
Tx FIFO interrupt level on page 1-57	This parameter corresponds to bits 4–0 (TXFFIL4-0) of the I2C transmit FIFO register (I2CFFTX).	
Enable Rx interrupt on page 1-57	This parameter corresponds to bit 5 (RXFFIENA) of the I2C receive FIFO register (I2CFFRX).	
Rx FIFO interrupt level on page 1-57	This parameter corresponds to bit 4–0 (RXFFIL4-0) of the I2C receive FIFO register (I2CFFRX).	
Enable system interrupt on page 1-57	Select this parameter to configure the five basic I2C interrupt request parameters in the interrupt enable register (I2CIER).	
Enable AAS interrupt on page 1-57	Enable the addressed-as-slave interrupt bit.	

Parameter	Description	Default Value
Enable SCD interrupt on page 1-57	Enable the stop condition detected interrupt bit.	
Enable ARDY interrupt on page 1-57	Enable the register-access-ready interrupt bit.	
Enable NACK interrupt on page 1-57	Enable the no acknowledgment interrupt bit.	
Enable AL interrupt on page 1-57	Enable the arbitration-lost interrupt bit.	

SCI_x

Parameter	Description	Default Value
Enable loopback on page 1-63	Enable the loopback function for self-test and diagnostics.	
Suspension mode on page 1-63	The type of suspension to use while debugging your program with Code Composer Studio.	
Number of stop bits on page 1-63	Specify the number of stop bits transmitted.	
Parity mode on page 1-63	The type of parity to use.	
Character length bits on page 1-63	Length in bits of each transmitted or received character.	8
Desired baud rate in bits/sec on page 1-63	Specify the desired baud rate.	
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)) on page 1-63	Scale the SCI baud rate using this value.	
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec on page 1-63	The closest achievable baud rate, calculated based on LSPCLK and BRR.	
Communication mode on page 1-63	Select the mode for transmitting and receiving data.	
Blocking mode on page 1-63	If this option is enabled, the system waits until data is available to read (when data length is reached).	
Data byte order on page 1-63	Select an option to match the endianness of the data being moved.	

Parameter	Description	Default Value
Pin assignment (Tx) on page 1-63	Assign the SCI transmit pin to use with the SCI module.	
Pin assignment (Rx) on page 1-63	Assign the SCI receive pin to use with the SCI module.	

SPI_x

Parameter	Description	Default Value
Mode on page 1-66	Set to `Master` or `Slave`.	
Desired baud rate in bits/sec on page 1-66	Specify the desired baud rate.	
Baud rate factor (SPIBRR: between 3 and 127) on page 1-66	The value used to calculate the baud rate.	
Closest achievable baud rate (LSPCLK/(SPIBRR+1)) in bits/sec on page 1-66	The closest achievable baud rate, calculated based on LSPCLK and SPIBRR.	
Suspension mode on page 1-66	The type of suspension to use while debugging your program with Code Composer Studio.	
Enable loopback on page 1-66	Enable the loopback function for self-test and diagnostics.	
Enable 3-wire mode on page 1-66	Enables SPI communication over three pins instead of the normal four pins.	
Enable Tx interrupt on page 1-66	Enable SPI transmit interrupt operation.	
FIFO interrupt level (Tx) on page 1-66	Set level for transmit FIFO interrupt.	
Enable Rx interrupt on page 1-66	Enable SPI receive interrupt operation.	
FIFO interrupt level (Rx) on page 1-66	Set level for receive FIFO interrupt.	
FIFO transmit delay on page 1-66	FIFO transmit delay (in processor clock cycles) to pause between data transmissions.	

Parameter	Description	Default Value
SIMO pin assignment on page 1-66	Assign the SPI (SIMO) to a GPIO pin.	
SOMI pin assignment on page 1-66	Assign the SPI value (SOMI) to a GPIO pin.	
CLK pin assignment on page 1-66	Assign the CLK pin to a GPIO pin.	
STE pin assignment on page 1-66	Assign the SPI value (STE) to a GPIO pin.	

eQEP

Parameter	Description	Default Value
EQEP#x pin assignment on page 1-68	Assign eQEP pin to a GPIO pin.	

Watchdog

Parameter	Description	Default Value
Enable watchdog on page 1-69	Enable the watchdog timer module.	
Counter clock on page 1-69	Set the watchdog timer period relative to OSCCLK/512.	
Timer period ((1/Counter clock)×256) in seconds on page 1-69	Display the timer period in seconds. This value automatically updates when you change the **Counter clock** parameter.	
Time out event on page 1-69	Configure the watchdog to reset the processor or generate an interrupt when the software fails to reset the watchdog counter.	

GPIO

Parameter	Description	Default Value
GPIO# on page 1-71	Use the GPIO pins for digital input or output by connecting to one of the three peripheral I/O ports.	

DMA_ch#

Parameter	Description	Default Value
Enable DMA channel on page 1-76	Enable to edit the configuration of a specific DMA channel.	
Data size on page 1-76	Select the size of the data bit transfer.	
Interrupt source on page 1-76	Select the peripheral interrupt that triggers a DMA burst for the specified channel.	
SRC wrap on page 1-76	Specify the number of bursts before returning the current source address pointer to the **Source Begin Address** value.	
DST wrap on page 1-76	Specify the number of bursts before returning the current destination address pointer to the **Destination Begin Address** value.	
SRC Begin address on page 1-76	Set the starting address for the current source address pointer.	
DST Begin address on page 1-76	Set the starting address for the current destination address pointer.	
Burst on page 1-76	Specify the number of 16-bit words in a burst, from 1 to 32.	
Transfer on page 1-76	Specify the number of bursts in a transfer, from 1 to 65536.	

Parameter	Description	Default Value
SRC Burst step on page 1-76	Increment or decrement the current address pointer by this number of 16-bit words before the next burst.	
DST Burst step on page 1-76	Increment or decrement the current address pointer by this number of 16-bit words before the next burst.	
SRC Transfer step on page 1-76	Increment or decrement the current address pointer by this number of 16-bit words before the next transfer.	
DST Transfer step on page 1-76	Increment or decrement the current address pointer by this number of 16-bit words before the next transfer.	
SRC Wrap step on page 1-76	Increment or decrement the SRC_BEG_ADDR address pointer by this number of 16-bit words when a wrap event occurs.	
DST Wrap step on page 1-76	Increment or decrement the DST_BEG_ADDR address pointer by this number of 16-bit words when a wrap event occurs.	
Generate interrupt on page 1-76	Enable this parameter to have the DMA channel send an interrupt to the CPU through the Peripheral Interrupt Expansion (PIE) at the beginning or end of a data transfer.	

Parameter	Description	Default Value
Enable one shot mode on page 1-76	Enable this parameter to have the DMA channel complete an entire *transfer* in response to an interrupt event trigger.	
Sync enable on page 1-76	Enable this parameter to reset the DMA wrap counter when the **Interrupt source** is set to SEQ1INT and sends the ADCSYNC signal to the DMA wrap counter.	
Enable continuous mode on page 1-76	Select this parameter to leave the DMA channel enabled upon completing a transfer. The channel waits for the next interrupt event trigger.	
Enable DST sync mode on page 1-76	Enabling this parameter resets the destination wrap counter (DST_WRAP_COUNT) when **Sync enable** is enabled and the DMA module receives the SEQ1INT interrupt/ ADCSYNC signal.	
Set channel 1 to highest priority on page 1-76	Enable this option when DMA channel 1 is configured to handle high-bandwidth data, such as ADC data, and the other DMA channels are configured to handle lower-priority data.	

Parameter	Description	Default Value
Enable overflow interrupt on page 1-76	Enable this parameter to have the DMA channel send an interrupt to the CPU through the PIE if the DMA module receives a peripheral interrupt while a previous interrupt from the same peripheral is waiting to be serviced.	

EMIF#

Parameter	Description	Default Value
EMIF clock divider (EMIF1CLKDIV) on page 1-84	Clock divider for clock frequency generation.	`SYSCLKOUT/2`
Enable CS0 for Synchronous memory on page 1-84	Chip select (CS0) to interface with the SDRAM.	`off`
Enable CS# for Asynchronous memory on page 1-84	Chip select (CS2/CS3/CS4) to interface with the asynchronous RAM.	`off`
SDRAM Column address bits on page 1-84	Value of the column address bits or the required page size of the connected SDRAM.	`8`
Number of internal SDRAM banks on page 1-84	Number of memory banks inside the connected SDRAM.	`3`
SDRAM data bus width in bits on page 1-84	Data bus width of the connected SDRAM.	`16`
Refresh to active command delay cycles (T_RFC) on page 1-84	Minimum number of EM#CLK cycles from the refresh or load mode command to the refresh or activate command in the connected SDRAM.	`3`
Row precharge to Active command delay cycles (T_RP) on page 1-84	Minimum number of EM#CLK cycles required from the row precharge command to the activate or refresh command in the connected SDRAM.	`1`

Parameter	Description	Default Value
Active to read or write command delay cycles (T_RCD) on page 1-84	Minimum number of EM#CLK cycles from the activate command to the read or write command in the connected SDRAM.	2
Last write to row precharge command delay cycles (T_WR) on page 1-84	Minimum number of EM#CLK cycles from the last write transfer or last data in command to the row precharge command in the connected SDRAM.	1
Active to precharge command delay cycles (T_RAS) on page 1-84	Minimum number of EM#CLK cycles from the activate command to the row precharge command in the connected SDRAM.	4
Active to active command delay cycles (T_RC) on page 1-84	Minimum number of EM#CLK cycles from an activate command to the next activate command in the same bank in the connected SDRAM.	6
Active one bank to active another bank command delay cycles (T_RRD) on page 1-84	Minimum number of EM#CLK cycles from an activate command in one bank to an activate command in a different bank in the connected SDRAM.	1
Self-refresh exit to other command delay cycles (T_XSR) on page 1-84	Minimum number of EM#CLK cycles from the self refresh exit command to any other command in the connected SDRAM.	7
SDRAM refresh period (tRefreshPeriod) in ms on page 1-84	Defines the rate at which the connected SDRAM refreshes.	64

Parameter	Description	Default Value
SDRAM CAS Latency on page 1-84	CAS latency required to access the connected SDRAM.	3
Asynchronous mode on page 1-84	Asynchronous mode for the connected asynchronous memory.	Normal
Asynchronous data bus width in bits on page 1-84	Data bus width of the connected asynchronous memory.	16
Read strobe setup cycles (R_SETUP) on page 1-84	Number of EM#CLK cycles from the EMIF chip select to the pin enable for asynchronous memory assert.	15
Read strobe duration cycles (R_STROBE) on page 1-84	Number of EM#CLK cycles during which the pin enable for the asynchronous memory is held active.	64
Read strobe hold cycles (R_HOLD) on page 1-84	Number of EM#CLK cycles during which the EMIF chip select is held active after pin enable for the asynchronous memory is deasserted.	7
Write strobe setup cycles (W_SETUP) on page 1-84	Number of EM#CLK cycles from the EMIF chip select to the write enable for the asynchronous memory assert.	15
Write strobe duration cycles (W_STROBE) on page 1-84	Number of EM#CLK cycles during which the write enable for the asynchronous memory is held active.	63

Parameter	Description	Default Value
Write strobe hold cycles (W_HOLD) on page 1-84	Number of EM#CLK cycles during which the EMIF chip select is held active after write enable for the asynchronous memory is deasserted.	7
Turn around cycles (TA) on page 1-84	Number of EM#CLK cycles between the end of one asynchronous memory access and the start of another asynchronous memory access.	3
Enable extended wait mode on page 1-84	Enable the extended wait option for the asynchronous memory.	off
Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255] on page 1-84	EMIF waits for *(MAX_EXT_WAIT+1) * 16* clock cycles before the asynchronous cycle is terminated.	128
Pin polarity of extended wait on page 1-84	Make EMIF wait if the pin is low or high.	High
Enable wait rise interrupt on page 1-84	Get an interrupt based on the detection of a rising edge on the EM#WAIT pin.	off
Enable timeout interrupt on page 1-84	Get an interrupt when the EM#WAIT pin does not become inactive within the number of cycles defined in **Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0-255]**.	off

Parameter	Description	Default Value
Enable line trap interrupt on page 1-84	Get an interrupt when there is an invalid cache line size or illegal memory access.	`off`

LIN

Parameter	Description	Default Value
LIN Module clock frequency (LM_CLK = SYSCLKOUT/2) in MHz on page 1-91	Display the frequency of the LIN module clock in MHz.	
Enable loopback on page 1-91	Enable LIN loopback testing.	
Suspension mode on page 1-91	Use this option to configure how the LIN state machine behaves while you debug the program using an emulator.	`Free_run`
Parity mode on page 1-91	Use this option to configure parity checking.	`None`
Frame length bytes on page 1-91	Set the number of data bytes in the response field, from 1–8 bytes.	`8`
Baud rate prescaler (P: 0-16777215) on page 1-91	To set the LIN baud manually, enter a prescaler value from 0–16777215.	`15`
Baud rate fractional divider (M: 0–15) on page 1-91	To set the LIN baud manually, enter a fractional divider value from 0–15.	`4`
Baud rate (FLINCLK = LM_CLK/(16×(P+1+M/16)) in bits/sec on page 1-91	Display the baud rate.	
Communication mode on page 1-91	Enable or disable the LIN module from using the ID-field bits ID4 and ID5 for length control.	`ID4 and ID5 not used for length control`
Data byte order on page 1-91	Set the endianness of the LIN message data bytes.	`Little_Endian`
Data swap width on page 1-91	Set the width for data swap.	

Parameter	Description	Default Value
Pin assignment (Tx) on page 1-91	Map the LINTX output to a specific GPIO pin.	`GPIO9`
Pin assignment (Rx) on page 1-91	Map the LINRX input to a specific GPIO pin.	`GPIO11`
LIN mode on page 1-91	Set the LIN module as a master or a slave.	`Slave`
ID filtering on page 1-91	Select the type of mask filtering comparison the LIN module performs.	`ID slave task byte`
ID byte on page 1-91	If you set **ID filtering** as `ID byte`, use this option to set the ID BYTE, also known as the “LIN mode message ID”.	`0x3A`
ID slave task byte on page 1-91	If you set **ID filtering** to `ID slave task byte`, use this option to set the ID-SlaveTask BYTE.	`0x30`
Checksum type on page 1-91	Select the checksum type.	`Classic`
Enable multibuffer mode on page 1-91	When you select this check box, the LIN node uses transmit and receive buffers instead of just one register.	Selected
Enable baud rate adapt mode on page 1-91	This option is displayed when you set **LIN mode** to `Slave`.	Not selected
Inconsistent synch field error interrupt on page 1-91	If you enable this option, the slave node generates interrupts when it detects irregularities in the synch field.	`Disabled`

Parameter	Description	Default Value
No response error interrupt on page 1-91	If you enable this option, the LIN module generates an interrupt if it does not receive a complete response from the master node within a timeout period.	`Disabled`
Timeout after 3 wakeup signals interrupt on page 1-91	When enabled, the slave node generates an interrupt when it sends three wakeup signals to the master node and does not receive a header in response.	`Disabled`
Timeout after wakeup signal interrupt on page 1-91	When enabled, the slave node generates an interrupt when it sends a wakeup signal to the master node and does not receive a header in response.	`Disabled`
Timeout interrupt on page 1-91	When enabled, the slave node generates an interrupt after 4 seconds of inactivity on the LIN bus.	`Disabled`
Wakeup interrupt on page 1-91	The LIN slave mode generates a wakeup interrupt based on a request or condition.	`Disabled`

External Interrupt

Parameter	Description	Default Value
XINT# GPIO on page 1-97	Set the GPIO pin for external interrupt.	
XINT# Polarity on page 1-97	Set the polarity for external interrupt.	

External Mode

Parameter	Description	Default Value
Communication interface on page 1-98	Use the `serial` option to run your model in external mode with serial communication.	
Serial port on page 1-98	Enter the COM port used by the target hardware.	
Verbose on page 1-98	Select to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.	

Execution Profiling

Parameter	Description	Default Value
Number of profiling samples to collect on page 1-99	Enter the number of profiling samples to collect.	

SD Card Logging

Parameter	Description	Default Value
Enable MAT-file logging on SD card on page 1-100	Enables the MAT-file logging for SD card.	`off`
SPI module on page 1-100	Select the desired interface on which the SD card is connected to hardware board.	
SPI baud rate on page 1-100	Select the desired option for the SPI interface used by the SD card.	`Maximum achievable supported by the inserted SD Card`

For more information on selecting a hardware support package and general configuration settings, see “Hardware Implementation Pane” (Simulink).

See Also

More About

- “Modeling”
- “Run on Target Hardware”

C28x-Scheduler Options

Use scheduler options to specify the base rate of your model. An interrupt can be used as a base rate trigger source for the scheduler. The scheduler options available are:

- `Timer 0`—The default option to schedule all synchronous rates present in your model with CPU Timer 0. When you select this option, the CPU Timer 0 is set according to the base rate of the model.
- `ADCINT1`—The option to schedule all synchronous rates present in your model with ADC interrupt 1 (ADCINT1). When you select this option, make sure that ADCINT1 triggers periodically at base rate used in the model.
- `ADCINT2`—The option to schedule all synchronous rates present in your model with ADC interrupt 2 (ADCINT2). When you select this option, make sure that ADCINT2 triggers periodically at base rate used in the model.

Warning Replacing the default scheduler interrupt source Timer 0 with ADCINT1 or ADCINT2 is an advanced setting. Ensure that you configure ADCINT1 or ADCINT2 to trigger periodically at the specified base rate. If the ADCINT1 or ADCINT2 does not trigger periodically or triggers at a different rate from the base rate, the model execution on the target is unpredictable.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-Build Options

Use the build options to specify how the build process takes place.

You can set the following parameters for build options:

Build action

Define how Embedded Coder responds when you build your model.

The `Build, load and run` option is supported for Texas Instruments Code Composer Studio CCS v4 and the later versions.

If you select the `Build, load and run` option, you must provide the required CCS hardware configuration file.

The TI Concerto F28M35x/F28M36x processors support only CCS v5 and the later versions. The TI Delfino F2807x/F2837x processors support only CCS v6 and the later versions.

Device Name

Select a particular device from the selected processor family.

Enable TMU

This option enables support for Trigonometric Math Unit (TMU). Relaxed floating-point mode also gets enabled as TMU hardware instructions are replaced only in relaxed floating point mode.

RTS library calls are replaced with the corresponding TMU hardware instructions for the following floating-point operations: floating point division, sqrt, sin, cos, atan, and atan2.

Note There are algorithmic differences between the TMU hardware instructions and the library routines, so the results of operations may differ slightly.

This option is available only for TI F2804x, F2807x, F2837xD and F2837xS processors.

Select CPU

Select a CPU core to run the generated code on a dual-core processor, such as F2837xD. Ensure you create two different models to run in different CPU cores. Use the appropriate options of this parameter to generate the code for your models.

The clock settings and the CPU assignment for F2837xD processor peripherals are available only when you select `CPU1` option. If you select `CPU2`, ensure you specify the same clock frequency as you have specified for the CPU1 model.

The GPIO pin configuration registers are available only for CPU1. For the GPIOs used in CPU2 model, configure the GPIO pins from CPU1.

For a Simulink generated code that you run in CPU1, the GPIO pin configuration for CPU2 takes place automatically. However, ensure you do not use the same GPIO pins in both CPU1 and CPU2.

For a handwritten code that you run in CPU1, ensure you configure the GPIO pins for CPU2 from CPU1.

Boot From Flash (stand alone execution)

The option to specify if the application has to load to the flash. If you do not select this option, the application loads to the RAM.

Use custom linker command file

Select this option, if you have your own custom linker file, which you can specify in the Linker command file parameter. If you do not select this option, based on the device you have selected, a default custom linker command file is used.

Linker command file

For each family of TI processor selected under **Target hardware resources**, one linker command file is selected automatically.

For a different variant of the processor, you can select the variant from the 'src' folder in the Support Package installation path. You can also create custom linker command file and select the file path using the **Browse** button.

The linker command file path provided can be absolute or relative. If the path provided is relative, the path must be selected with respect to the folder where the model is present or the code generation folder.

CCS hardware configuration file

In the Support Package installation folder, open CCS_Config and select one of the `ccxml` files.

Alternately, can use Code Composer Studio to create the `ccxml` file. In Code Composer Studio, go to **File > New > Target Configuration File**. Select the file you created using the **Browse** button. You can also edit the `ccxml` file using the **Edit** button.

The `ccxml` files provided with Embedded Coder Support Package for Texas Instruments C2000 Processors are as follows:

- f28027.ccxml—TI F28027 with Texas Instruments XDS100v1 USB Emulator
- f28035.ccxml—TI F28035 with Texas Instruments XDS100v1 USB Emulator
- f28069.ccxml—TI F28069 with Texas Instruments XDS100v1 USB Emulator
- f2808.ccxml—TI F2808 with Texas Instruments XDS100v1 USB Emulator
- f2808_eZdsp.ccxml—F2808 Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
- f28044.ccxml—TI F28044 with Texas Instruments XDS100v1 USB Emulator
- f28335.ccxml—TI F28335 with Texas Instruments XDS100v1 USB Emulator
- f28335_eZdsp.ccxml—F28335 Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
- f2812_BH2000.ccxml—Blackhawk USB2000 Controller for F2812 eZDSP
- f28x_generic.ccxml—Generic Texas Instruments XDS100v1 USB Emulator
- f28x_ezdsp_generic.ccxml—Generic Spectrum Digital eZdsp onboard USB Emulator
- f28x_ezdsp_generic.ccxml—Generic Spectrum Digital eZdsp onboard USB Emulator
- f28377S.ccxml—TI F2837xS with Texas Instruments XDS100v2 USB Emulator
- f28075.ccxml—TI F2807x with Texas Instruments XDS100v2 USB Emulator
- f28377D.ccxml—TI F2837xD with Texas Instruments XDS100v2 USB Emulator
- f28379D.ccxml—TI F2839xD with Texas Instruments XDS100v2 USB Emulator
- f28004x.ccxml—TI F28004x with Texas Instruments XDS100v2 USB Emulator

The `ccxml` files provided with Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto™ Processors are as follows:

- f28M35x.ccxml – Texas Instruments XDS100v2 USB Emulator_0
- f28M36x.ccxml – Texas Instruments XDS100v2 USB Emulator_0

Enable DMA to access ePWM Registers instead of CLA

The option that you can select to enable the DMA to access ePWM registers instead of CLA. This option is available only for F2806x processors.

Enable DMA to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS, and SDFM) instead of CLA

The option that you can select to enable the DMA to access peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS and SDFM) registers instead of CLA. This option is available only for F2837xD, F2837xS, F2807x processors.

Enable DMA to peripheral frame 2 (SPI and McBSP) instead of CLA

The option that you can select to enable the DMA to access peripheral frame 2 (SPI and McBSP) registers instead of CLA. This option is available only for F2837xD, F2837xS, F2807x processors.

Enable FastRTS

This option enables the use of optimized floating point math functions from C28x FPU fastRTS library instead of standard RTS library functions.

By using FastRTS library routines, you can achieve execution speeds considerable faster without rewriting existing code. This option is available only for F2806x, F2833x, F28M35x (C28x) and F28M35x (C28x) processors.

Remap ePWMs for DMA access (Requires silicon revision A and above)

The option that you can select to remap ePWMs registers for DMA access. This option is available only for F2833x processors.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-Clocking

Use the clocking options to achieve the CPU clock rate specified on the board. The default clocking values run the CPU clock (CLKIN) at its maximum frequency. The parameters use the external oscillator frequency on the board (OSCCLK) that is recommended by the processor vendor.

For F2837xD and F2838xD dual-core processor, the clock settings are available only when you select the CPU1 option in the **Build options > Select CPU** parameter. When you select CPU2 option in the **Build options > Select CPU** parameter, set the CPU clock with the value available in the **Achievable SYSCLKOUT in MHz** parameter for the CPU1 model.

You can get feedback on the closest achievable SYSCLKOUT value with the specified oscillator clock frequency by selecting the **Auto set PLL based on OSCCLK and CPU clock** check box. Alternatively, you can manually specify the PLL value for the SYSCLKOUT value calculation.

Change the clocking values if:

- You want to change the CPU frequency.
- The external oscillator frequency differs from the value recommended by the manufacturer.

To determine the CPU frequency (CLKIN), use the following equation:

CLKIN = (OSCCLK × PLLCR) / (DIVSEL or CLKINDIV)

Where,

- CLKIN is the frequency at which the CPU operates, also known as the CPU clock.
- OSCCLK is the frequency of the oscillator.
- **PLLCR** is the PLL control register value.
- **CLKINDIV** is the clock in the divider.
- **DIVSEL** is the divider select.

The availability of the DIVSEL or CLKINDIV parameters changes depending on the processor that you select. If neither parameter is available, use the following equation:

CLKIN = (OSCCLK × PLLCR) / 2

You can set the following parameters for clocking:

Desired C28x CPU clock in MHz

Specify the desired CPU clock frequency (CLKIN). This value is taken automatically for **Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/DIVSEL**.

CPU Clock in MHz (C28SYSCLK/SYSCLKOUT)

Enter the value that you specified for **Desired C28x CPU clock in MHz**. This parameter is available only for TI Concerto F28M35x/ F28M36x processors. For more information, see the PLL-Based Clock Module section in the Texas Instruments *Reference Guide* for your processor.

Use internal oscillator

Use the internal zero pin oscillator on the CPU. This parameter is enabled by default.

Oscillator clock (OSCCLK) frequency in MHz

Oscillator frequency used in the processor. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Auto set PLL based on OSCCLK and CPU clock

PLL values in PLLCR, DIVSEL, and **Achievable SYSCLKOUT in MHz** are automatically calculated based on the CPU clock entered on the board. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

PLL control register (PLLCR)

If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto calculated control register value achieves the specified CPU clock value, based on the oscillator clock frequency. Alternatively, you can select a value for **PLL control register (PLLCR)**. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

PLL output divider (ODIV)

Calculates SYSCLKOUT = ((OSCCLK×SYSPLLMULT)/ODIV)/SYSDIVSEL.

Clock divider (DIVSEL)

If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto calculated control register value achieves the specified CPU clock value, based on the oscillator clock frequency. Alternatively, you can select a value for **Clock divider (DIVSEL)**. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/DIVSEL

The auto calculated feedback value that matches the **Desired C28x CPU clock in MHz** value, based on the values of OSCCLK, PLLCR, and DIVSEL. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Set the 'Achievable SYSCLKOUT in MHz = (OSCCLK*SYSPLLMULT)/SYSDIVSEL' value calculated in CPU1

Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Achievable SYSCLKOUT in MHz = (OSCCLK*PLLCR)/DIVSEL** (auto calculated).

Select the 'Low-Speed Peripheral Clock Prescaler (LSPCLK)' option used in CPU1

Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Low-Speed Peripheral Clock Prescaler (LSPCLK)** specified in CPU1.

Low-Speed Peripheral Clock Prescaler (LSPCLK)

The value using which LSPCLK is scaled. This value is based on SYSCLKOUT.

Low-Speed Peripheral Clock (LSPCLK) in MHz

The value is calculated based on LSPCLK Prescaler. Example: SPI uses a LSPCLK.

High-Speed Peripheral Clock Prescaler (HSPCLK)

The value using which HSPCLK is scaled. This value is based on SYSCLKOUT.

High-Speed Peripheral Clock (HSPCLK) in MHZ

The value is calculated based on HSPCLK Prescaler. Example: ADC uses a HSPCLK.

Analog Subsystem Clock Prescaler (ASYSCLK)

The value using which ASYSCLK is scaled. This value is based on SYSCLKOUT. This option is available only for TI Concerto F28M35x/ F28M36x processors.

Analog Subsystem Clock (ASYSCLK)

The value calculated using the SYSCLKOUT and ASYSCLK Prescaler values. This option is available only for TI Concerto F28M35x/ F28M36x processors.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-ADC/C28x-ADC_A/C28x-ADC#

The high-speed peripheral clock (HSPCLK) or the system clock (SYSCLKOUT) controls the internal timing of ADC modules. The ADC derives the operating clock speed from the HSPCLK/SYSCLKOUT speed in several prescaler stages. For more information about configuring these scalers, see “Configuring Acquisition Window Width for ADC Blocks”.

You can set the following parameters for the ADC clock prescaler:

Select the CPU core which controls ADC_x module

This parameter is available only for the dual-core processor F2837xD with the **Build options > Select CPU** parameter set to `CPU1`.

The CPU core that controls the ADC module. When you select the `Auto` option for the ADC_x module in a model, the ADC_x module is assigned to the CPU1 core during code generation if the ADC_x block is present in the model, else it is assigned to the CPU2 core. If an ADC_x module is assigned to a CPU core, you cannot use that module in a model that runs in the other CPU core.

ADC clock prescaler (ADCCLK)

Select the ADCCLK divider. This is specific to a processor.

ADC clock frequency in MHz

The clock frequency for ADC, which is auto generated based on the value you select in **ADC clock prescaler (ADCCLK)**.

ADC overlap of sample and conversion (ADC#NONOVERLAP)

Enable or disable overlap of sample and conversion.

ADC clock prescaler (ADCLKPS)

The HSPCLK is divided by ADCLKPS (a 4-bit value) as the first step in deriving the core clock speed of the ADC. The default value is `3`.

ADC Core clock prescaler (CPS)

After dividing the HSPCLK speed by the **ADC clock prescaler (ADCLKPS)** value, divides the result by 2. The default value is `1`.

ADC Module clock (ADCCLK = HSPCLK/ADCLKPS*2)/(CPS+1)) in MHz

The ADC module clock, which indicates the ADC operating clock speed.

Acquisition window prescaler (ACQ_PS)

This value determines the width of the sampling or acquisition period. The higher the value, the wider is the sampling period. This value does not directly alter the core clock speed of the ADC. The default value is `4`.

Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel

Acquisition window size determines the duration for which the sampling switch is closed. The width of SOC pulse is ADCTRL1[11:8] + 1 times the ADCLK period.

Offset

Specifies the offset value.

Use external reference 2.048V External reference

By default, an internally generated band gap voltage reference supplies the ADC logic. However, depending on application requirements, you can enable the external reference so that the ADC logic uses an external voltage reference instead.

Continuous mode

When the ADC generates an end of conversion (EOC) signal, an ADCINT# interrupt that indicates whether the previous interrupt flag has been acknowledged or not is generated.

ADC offset correction (OFFSET_TRIM: –256 to 255)

The 280x ADC supports offset correction via a 9-bit value that it adds or subtracts before the results are available in the ADC result registers. Timing for results is not affected. The default value is `0`.

VREFHIVREFLO

When you disable the **Use external reference 2.048V** or **External reference** option, the ADC logic uses a fixed 0–3.3 volt input range, and **VREFHI** and **VREFLO** are disabled. To interpret the ADC input as a ratiometric signal, select the **External reference** option. Then, set values for the high voltage reference (**VREFHI**) and the low voltage reference (**VREFLO**). **VREFHI** uses the external ADCINA0 pin, and **VREFLO** uses the internal GND.

INT pulse control

Set the time when the ADC sets ADCINTFLG ADCINTx relative to the SOC and EOC pulses.

SOC high priority

Enables **SOC high priority mode**. In `all in round robin mode`, the default selection, the ADC services each SOC interrupt in a numerical sequence.

Choose one of the `high priority` selections to assign high priority to one or more of the SOCs. In this mode, the ADC operates in round robin mode until it receives a high priority SOC interrupt. The ADC finishes servicing the current SOC, services the high priority SOCs, and then returns to the next SOC in the round robin sequence.

For example, the ADC is servicing SOC8 when it receives a high priority interrupt on SOC1. The ADC completes servicing SOC8, services SOC1, and then services SOC9.

XINT2SOC external pin

The pin to which the ADC sends the XINT2SOC pulse.

ADCEXTSOC external pin

The pin to which the ADC sends the ADCEXTSOC pulse. This parameter is available only for F2807x, F2837x processors.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-COMP

Assign COMP pins to GPIO pins.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-DAC

DACx reference voltage

Select the reference voltage for the DAC channel A, B, or C. In this case, x represents the DAC channel A, B, or C.

- ADC reference voltage (VREFHIA/VREFHIB) — The reference voltage used for the ADC. You can use this as reference voltage VREFHIA for DAC A, DAC B and VREFHIB for DAC C.
- External reference voltage through ADCINB0 (VDAC) — A separate external reference voltage for DAC. Ensure that you connect the ADCINB0 pin to the supply voltage.

DACx synchronization signal

Select the synchronization signal to load the value from the writable shadow register into the active register. In this case, x represents the DAC channel A, B, or C.

- SYSCLK — Loads the value from the writable shadow register DACVALS into the active register DACVALA on the next clock cycle.
- PWMSYNC1–12 — Loads the value from the writable shadow register DACVALS into the active register DACVALA on the next PWM synchronization event.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-eCAN_A, C28x-eCAN_B

You can set the following parameters for the eCAN module:

CAN module clock frequency (= SYSCLKOUT) in MHz

The clock to the enhanced CAN module. The CAN module clock frequency is equal to SYSCLKOUT for processors such as c280x, c281x, c28044.

CAN module clock frequency (=SYSCLKOUT/2) in MHz

The clock to the enhanced CAN module. The CAN module clock frequency is equal to SYSCLKOUT/2 for processors such as piccolo, c2834x, c28x3x.

Baud rate prescaler (BRP: 2 to 256)/Baud rate prescaler (BRP: 1 to 1024)

The value using which bit rate is scaled.

Time segment 1 (TSEG1):

Set the value of time segment 1. This value, with **TSEG2** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from `1` through `16`.

Time segment 2 (TSEG2):

Set the value of time segment 2. This value, with **TSEG1** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from `1` through `8`.

Baud rate (CAN Module Clock/BRP/(TSEG1 + TSEG2 +1)) in bits/sec:

CAN module communication speed represented in bits/sec.

SBG

Set the message resynchronization triggering.

SJW

Set the synchronization jump width, which determines how many units of TQ a bit can be shortened or lengthened when resynchronizing. Where, TQ=Baud Rate Prescaler/CAN_CLK.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of TQ/2. The CAN module makes a majority decision from the three points.

Enhanced CAN Mode

Enable time-stamping and usage of **Mailbox Numbers** 16 through 31 in the C2000 eCAN blocks. Texas Instruments documentation refers to this as “HECC mode”.

Self test mode

If you set this parameter to `True`, the eCAN module goes to loopback mode. The loopback mode sends a “dummy” acknowledge message back. This mode does not need an acknowledge bit. The default is `False`.

Pin assignment (Tx)

Assign the CAN transmit pin to use with the `eCAN_B` module.

Pin assignment (Rx)

Assign the CAN receive pin to use with the `eCAN_B` module.

For more information about setting the timing parameters for the eCAN modules, see “Configuring Timing Parameters for CAN Blocks”.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-eCAP

Assign eCAP pins to GPIO pins.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-ePWM

Assign ePWM signals to GPIO pins. You can set the following parameters for ePWM:

EPWM clock divider (EPWMCLKDIV)

Select the ePWM clock divider. This parameter is available only for F2807x, F2837x, F2838x processors.

Select the 'EPWM clock divider (EPWMCLKDIV)' option used for CPU1

Available only for CPU2 of dual C28x core processors. Its value must be the same as the value of the parameter EPWM clock divider (EPWMCLKDIV) selected in CPU1.

TZx pin assignment

Assign the trip-zone input x (TZx) to a GPIO pin.

For F2807x, F2837x and F2838x processors, select `None` or `GPIO#`.

Note To enter the GPIO numbers in the**TZ# pin assignment** parameters for F2807x, F2837x and F2838x processors, select `CPU1` in **Build options >Select CPU**.

Note The TZ# pin assignments are available only for TI F280x processors.

SYNCI pin assignment

Assign the ePWM external sync pulse input (SYNCI) to a GPIO pin.

For F2807x, F2837x and F2838x processors, select `None` or `GPIO#`.

Note To enter the GPIO numbers in **SYNCI pin assignment** for F2807x, F2837x and F2838x processors, you must select `CPU1` in **Build options >Select CPU**.

SYNCO pin assignment

Assign the ePWM external sync pulse output (SYNCO) to a GPIO pin.

Note SYNCI and SYNCO pin assignments are available for TI F28044, TI F280x, TI Delfino F2833x, TI Delfino F2834x, TI Piccolo F2802x, TI Piccolo F2803x, TI Piccolo F2806 processors.

PWM#x pin assignment

Assign the GPIO pin to the PWM#x module.

GPTRIP#SEL pin assignment(GPIO0~63)

Assign the ePWM trip-zone input to a GPIO pin.

Note The GPTRIP#SEL pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

PWM1SYNCI/ GPTRIP6SEL pin assignment

Assign the ePWM sync pulse input (SYNCI) to a GPIO pin.

Note The PWM1SYNCI/GPTRIP#SEL pin assignments are available only for TI Concerto F28M35x/F28M36x processors.

DCxHTRIPSEL (Enter Hex value between 0 and 0x6FFF) / DCBHTRIPSEL (Enter Hex value between 0 and 0x6FFF)

Assign the **Digital Compare A** high trip input to a GPIO pin.

Note DCxHTRIPSel pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

DCxLTRIPSEL (Enter Hex value between 0 and 0x6FFF) / DCBLTRIPSEL (Enter Hex value between 0 and 0x6FFF)

Assign the **Digital Compare A** low trip input to a GPIO pin.

Note The DCxLTRIPSEL pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-I2C

You can set the following parameters for I2C:

Mode

Configure the I2C module as `Master` or `Slave`.

If a module is an I2C master, it:

- Initiates communication with slave nodes by sending the slave address and requesting data transfer to or from the slave.
- Outputs the **Master clock frequency** on the serial clock line (SCL) line.

If a module is an I2C slave, it:

- Synchronizes itself with the serial clock line (SCL) line.
- Responds to communication requests from the master.

In `Slave` mode, you can configure the **Addressing format**, **Address register**, and **Bit count** parameters.

The **Mode** parameter corresponds to bit 10 (MST) of the I2C mode register (I2CMDR).

Addressing format

In `Slave` mode, determines the addressing format of the I2C master and sets the I2C module to the same mode:

- `7-Bit Addressing`—the normal address mode.
- `10-Bit Addressing`—the expanded address mode.
- `Free Data Format`—a mode that does not use addresses. (If you **Enable loopback**, the `Free data format` is not supported.)

The **Addressing format** parameter corresponds to bit 3 (FDF) and bit 8 (XA) of the I2C mode register (I2CMDR).

Own address register

In `Slave` mode, enter the 7-bit (0–127) or 10-bit (0–1023) address that the I2C module uses while it is a slave.

This parameter corresponds to bits 9–0 (OAR) of the I2C own address register (I2COAR).

Bit count

In `Slave` mode, sets the number of bits in each *data byte* the I2C module transmits and receives. This value must match that of the I2C master.

This parameter corresponds to bits 2–0 (BC) of the I2C mode register (I2CMDR).

Module clock prescaler (IPSC: 0 to 255)

In `Master` mode, configures the module clock frequency by entering a value 0–255, inclusive.

Module clock frequency = *I2C input clock frequency* / (*Module clock prescaler* + 1)

The I2C specifications require a module clock frequency between 7 MHz and 12 MHz.

The *I2C input clock frequency* depends on the DSP input clock frequency and the value of the PLL control register divider (PLLCR). For more information on setting the PLLCR, see the documentation for your digital signal controller.

The **Module clock prescaler (IPSC: 0 to 255)** corresponds to bits 7–0 (IPSC) of the I2C prescaler register (I2CPSC).

I2C Module clock frequency (SYSCLKOUT / (IPSC+1)) in Hz

Display the frequency the I2C module uses internally. To set this value, change the **Module clock prescaler**.

For more information about this value, see the “Formula for the Master Clock Period” section in the *TMS320x280x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721, on the Texas Instruments website.

I2C Master clock frequency (Module Clock Freq/(ICCL+ICCH+10)) in Hz

Display the master clock frequency.

For more information about this value, see the “Clock Generation” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721/ SPRUH22F/ SPRUHE8B, available on the Texas Instruments website.

Master clock Low-time divider (ICCL: 1 to 65535)

In `Master` mode, the divider determines the duration of the low state of the serial clock line (SCL) on the I2C bus.

The low-time duration of the master clock = Tmod x (ICCL + d).

For more information, see the “Formula for the Master Clock Period” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A/ SPRUH22F/ SPRUHE8B, available on the Texas Instruments website.

This parameter corresponds to bits 15–0 (ICCL) of the clock low-time divider register (I2CCLKL).

Master clock High-time divider (ICCH: 1 to 65535)

In `Master` mode, the divider determines the duration of the high state of the serial clock line (SCL) on the I2C bus.

The high-time duration of the master clock = Tmod x (ICCL + d).

For more information about this value, see the “Formula for the Master Clock Period” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A, SPRUH22f, SPRUHE8B, available on the Texas Instruments website.

This parameter corresponds to bits 15–0 (ICCH) of the clock high-time divider register (I2CCLKH).

Enable loopback

In `Master` mode, enables or disables digital loopback mode. In digital loopback mode, I2CDXR transmits data over an internal path to I2CDRR, which receives the data after a configurable delay.

The delay, measured in DSP cycles, equals (I2C input clock frequency/module clock frequency) x 8.

While **Enable loopback** is enabled, free data format addressing is not supported.

This parameter corresponds to bit 6 (DLB) of the I2C mode register (I2CMDR).

SDA pin assignment

Select a GPIO pin as I2C data bidirectional port.

This parameter is not available for TI C2000 F280x, F28044, F2833x, and C2834x processors.

SCL pin assignment

Select a GPIO pin as I2C clock bidirectional port.

This parameter is not available for TI C2000 F280x, F28044, F2833x, and C2834x processors.

Enable Tx interrupt

This parameter corresponds to bit 5 (TXFFIENA) of the I2C transmit FIFO register (I2CFFTX).

Tx FIFO interrupt level

This parameter corresponds to bits 4–0 (TXFFIL4-0) of the I2C transmit FIFO register (I2CFFTX).

Enable Rx interrupt

This parameter corresponds to bit 5 (RXFFIENA) of the I2C receive FIFO register (I2CFFRX).

Rx FIFO interrupt level

This parameter corresponds to bit 4–0 (RXFFIL4-0) of the I2C receive FIFO register (I2CFFRX).

Enable system interrupt

Select this parameter to configure the five basic I2C interrupt request parameters in the interrupt enable register (I2CIER):

- Enable AAS interrupt
- Enable SCD interrupt
- Enable ARDY interrupt
- Enable NACK interrupt
- Enable AL interrupt

Enable AAS interrupt

Enable the addressed-as-slave interrupt.

When enabled, the I2C module generates an interrupt (AAS bit = 1) upon receiving one of the following:

- Its **Own address register** value
- A general call (all zeros)
- A data byte in free data format

When enabled, the I2C module clears the interrupt (AAS = 0) upon receiving one of the following:

- Multiple START conditions (7-bit addressing mode only)
- A slave address that is different from **Own address register** (10-bit addressing mode only)
- A NACK or a STOP condition

This parameter corresponds to bit 6 (AAS) of the interrupt enable register (I2CIER).

Enable SCD interrupt

Enable STOP condition detected interrupt.

When enabled, the I2C module generates an interrupt (SCD bit = 1) after the CPU detects a stop condition on the I2C bus.

When enabled, the I2C module clears the interrupt (SCD = 0) upon one of the following events:

- The CPU reads I2CISRC while it indicates a stop condition
- A reset of the I2C module
- Someone manually clears the interrupt

This parameter corresponds to bit 5 (SCD) of the interrupt enable register (I2CIER).

Enable ARDY interrupt

Enable register-access-ready interrupt enable bit.

When enabled, the I2C module generates an interrupt (ARDY bit = 1) after the previous address, data, and command values in the I2C module registers have been used. New values can be written to the I2C module registers.

This parameter corresponds to bit 2 (ARDY) of the interrupt enable register (I2CIER).

Enable NACK interrupt

Enable no acknowledgment interrupt enable bit.

When enabled, the I2C module generates an interrupt (NACK bit = 1) when the module operates as a transmitter in master or slave mode and receives a NACK condition.

This parameter corresponds to bit 1 (NACK) of the interrupt enable register (I2CIER).

Enable AL interrupt

Enable arbitration-lost interrupt.

When enabled, the I2C module generates an interrupt (AL bit = 1) when the I2C module operates as a master transmitter and looses an arbitration contest with another master transmitter.

This parameter corresponds to bit 0 (AL) of the interrupt enable register (I2CIER).

For more information about the I2C parameters, see the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A/ SPRUH22F/ SPRUHE8B available on the Texas Instruments website.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-SCI_A, C28x-SCI_B, C28x-SCI_C, C28x_SCI_D

You can set the following parameters for serial communications interface (SCI):

Enable loopback

Enable the loopback function for self-test and diagnostics. When this function is enabled, a C28x DSP Tx pin is internally connected to its Rx pin, and the DSP can transmit data from its output port to its input port to check the integrity of the transmission.

Suspension mode

The type of suspension to be used while debugging your program with Code Composer Studio. When your program encounters a breakpoint, the suspension mode determines whether to perform the program instruction. The available options are:

- `Hard_abort`—Stops the program immediately.
- `Soft_abort`—Stops when the current receive/transmit sequence is complete.
- `Free_run`—Continues running regardless of the breakpoint.

Number of stop bits

Specify the number of stop bits transmitted.

Parity mode

The type of parity to be used. The available options are:

- `None`—Disables parity.
- `Odd`—Sets the parity bit to one if you have an odd number of ones in your bytes, such as 00110010.
- `Even`—Sets the parity bit to one if you have an even number of ones in your bytes, such as 00110011.

Character length bits

Length in bits of each transmitted or received character. The default value is 8.

Desired baud rate in bits/sec

Specify the desired baud rate.

Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD))

The value using which SCI baud rate is scaled. This value is based on LSPCLK.

Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec

The closest achievable baud rate calculated based on LSPCLK and BRR.

Communication mode

Raw data is unformatted and sent whenever the transmitting side is ready to send, regardless of whether the receiving side is ready or not. Without a wait state, deadlock conditions do not occur and data transmission is asynchronous. With this mode, the receiving side could miss data, but if the data is noncritical, using raw data mode can avoid blocking processes.

When you select protocol mode, handshaking between the host and the processor occurs. The transmitting side sends $SND to indicate it is ready to transmit. The receiving side sends back $RDY to indicate it is ready to receive. The transmitting side then sends data and, when the transmission is completed, it sends a checksum.

Advantages of using protocol mode are:

- Avoids deadlock
- Determines whether data is received without errors (checksum)
- Determines whether data is received by the processor
- Determines time consistency; each side waits for its turn to send or receive

Note Deadlocks can occur if an SCI Transmit block tries to communicate with multiple SCI Receive blocks on different COM ports while both transmit and receive blocks are in blocking mode. Deadlocks cannot occur on the same COM port.

Blocking mode

If this option is enabled, the system waits until data is available to read (when data length is reached). If this option is disabled, the system checks FIFO periodically (in polling mode) for data to read. If data is present, the block reads and outputs the contents. If data is not present, the block outputs the last value and continues.

Data byte order

Select an option to match the endianness of the data being moved.

Pin assignment (Tx)

Assign the SCI transmit pin to be used with the SCI module.

Pin assignment (Rx)

Assign the SCI receive pin to be used with the SCI module.

Note All SCI modules are not available for all TI C2000 processors.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-SPI_A, C28x-SPI_B, C28x-SPI_C, C28x-SPI_D

You can set the following parameters for the serial peripheral interface (SPI):

Mode

Configure the SPI module as `Master` or `Slave`.

Desired baud rate in bits/sec

Specify the desired baud.

Baud rate factor (SPIBRR: between 3 and 127)

The value used to calculate the baud. To set the **Baud rate factor**, search for “Baud Rate Determination” and “SPI Baud Rate Register (SPIBRR) Bit Descriptions” in *TMS320x28xx, 28xxx DSP Serial Peripheral Interface (SPI) Reference Guide*, Literature Number: SPRU059, available on the Texas Instruments website.

Closest achievable baud rate (LSPCLK/(SPIBRR+1)) in bits/sec

The closest achievable baud rate calculated based on LSPCLK and SPIBRR.

Suspension mode

The type of suspension to be used while debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. The available options are:

- `Hard_abort`—Stops the program immediately.
- `Soft_abort`—Stops when the current receive/transmit sequence is complete.
- `Free_run`—Continues running regardless of the breakpoint.

Enable loopback

Enable the loopback function for self-test and diagnostics. When this function is enabled, the Tx pin on a C28x DSP is internally connected to its Rx pin, and the DSP can transmit data from its output port to its input port to check the integrity of the transmission.

Enable 3-wire mode

Enable SPI communication over three pins instead of the normal four pins.

Enable Tx interrupt

Enable SPI transmit interrupt operation.

FIFO interrupt level (Tx)

Set level for transmit FIFO interrupt.

Enable Rx interrupt

Enable SPI receive interrupt operation.

FIFO interrupt level (Rx)

Set level for receive FIFO interrupt.

FIFO transmit delay

FIFO transmit delay (in processor clock cycles) to pause between data transmissions. Enter an integer.

SIMO pin assignment

Assign the SPI (SIMO) to a GPIO pin.

SOMI pin assignment

Assign the SPI value (SOMI) to a GPIO pin.

CLK pin assignment

Assign the CLK pin to a GPIO pin.

CLK pin assignment is not available for TI Concerto F28M35x/F28M36x processors.

STE pin assignment

Assign the SPI value (STE) to a GPIO pin.

STE pin assignment is not available for TI Concerto F28M35x/ F28M36x processors.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-eQEP

Assign eQEP pins to GPIO pins.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-Watchdog

When enabled, if the software fails to reset the watchdog counter within a specified interval, the watchdog resets the processor or generates an interrupt. This feature enables the processor to recover from faults.

For more information, see the *Data Manual* or *System Control and Interrupts Reference Guide* for your processor on the Texas Instruments website.

Enable watchdog

Enable the watchdog timer module.

This parameter corresponds to bit 6 (WDDIS) of the watchdog control register (WDCR) and bit 0 (WDOVERRIDE) of the system control and status register (SCSR).

Counter clock

Set the watchdog timer period relative to OSCCLK/512.

This parameter corresponds to bits 2–0 (WDPS) of the watchdog control register (WDCR).

Note Depending on the processor type, the default value of the watchdog clock (WDCLK) can be based on the internal oscillator (INTOSC1) or external oscillator (OSCCLK).

Timer period ((1/Counter clock)*256) in seconds

Display the timer period in seconds. This value automatically gets updated when you change the **Counter clock** parameter.

Time out event

Configure the watchdog to reset the processor or generate an interrupt when the software fails to reset the watchdog counter. The available options are:

- `Chip reset`—Generates a signal (WDRST) that resets the processor and disables the watchdog interrupt signal (WDINT).
- `Raise WD Interrupt`—Generates a watchdog interrupt signal (WDINT) and disables the reset processor signal (WDRST). The WDINT signal can be used to wake the device from an idle or standby low-power mode.

This parameter corresponds to bit 1 (WDENINT) of the system control and status register (SCSR).

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-GPIO

Use the GPIO pins for digital input or output by connecting to one of the three peripheral I/O ports.

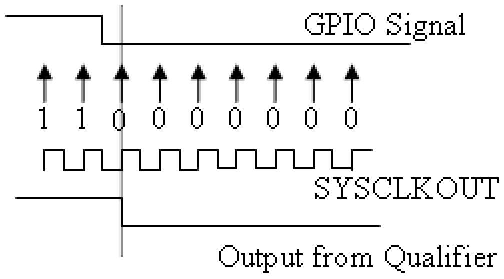
The GPIO pins available for various processors are:

Processors	GPIO Pin Values
C281x	GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, and GPIOG.
F2803x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, and GPIO40_44.
F2805x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, and GPIO40_42.
F2806x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_44, GPIO50_55, and GPIO56_58.
F2823x, F2833x, and C2834x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, and GPIO56_63.
C2801x, F2802x, F28044, F280x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, and GPIO32_34.
F28M35x (C28x)	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, GPIO56_63, GPIO68_71, and GPIO128_135.
F28M36x (C28x)	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, and GPIO192_199.
F2807x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, and GPIO128_135.
F2837xD	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.

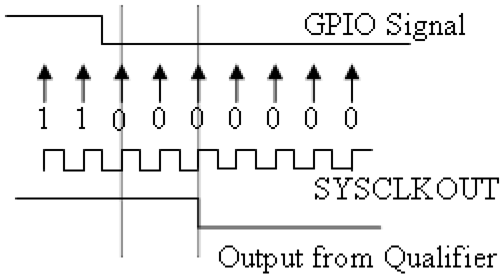
Processors	GPIO Pin Values
F2837xS	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.
F2838x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.
F28004x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, and GPIO56_63.

Each pin selected for input offers four signal qualification types:

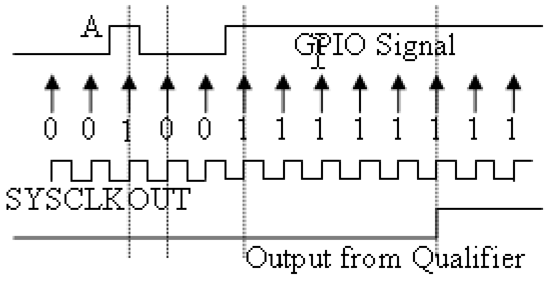
- `Synchronize to SYSCLKOUT only`—This setting is the default for all pins at reset. Using this qualification type, the input signal is synchronized to the system clock, SYSCLKOUT. The following figure shows the input signal measured on each tick of the system clock, and the resulting output from the qualifier.



- `Qualification using 3 samples`—This setting requires three consecutive cycles of the same value for the output value to change. The following figure shows that, in the third cycle, the GPIO value changes to `0`, but the qualifier output is still `1` because it waits for three consecutive cycles of the same GPIO value. The next three cycles have a value of `0`, and the output from the qualifier changes to `0` immediately after the third consecutive value is received.



- `Qualification using 6 samples`—This setting requires six consecutive cycles of the same GPIO input value for the output from the qualifier to change. In the following figure, glitch **A** does not alter the output signal. When the glitch occurs, the counting begins, but as the next measurement is low again, the count is ignored. The output signal does not change until six consecutive samples of the high signal are measured.

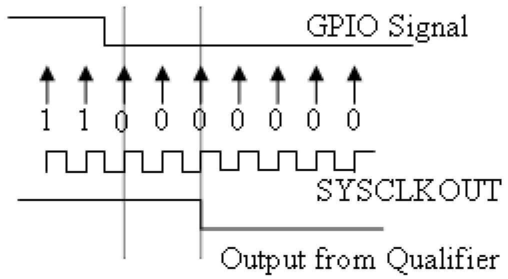


Note These GPIO settings are supported for the F2837xD dual core processor only when you select `CPU1` in **Build options** >**Select CPU**.

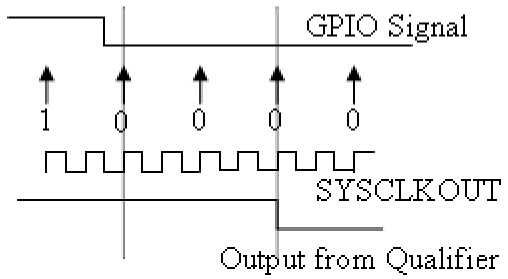
Qualification sampling period

Visible only when the `Qualification using # samples` option is selected. The qualification sampling period, with possible values of `0`–`255`, inclusive, calculates the frequency of the qualification samples or the number of system clock ticks per sample. The formula for calculating the qualification sampling frequency is *SYSCLKOUT/(2 * Qualification sampling period)*, except for zero. When **Qualification sampling period=**`0`, a sample is taken every SYSCLKOUT clock tick. For example, a setting of `0` means that a sample is taken on each SYSCLKOUT tick.

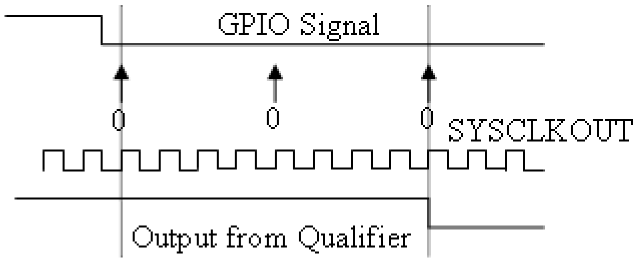
The following figure shows the SYSCLKOUT ticks, a sample taken every clock tick, and the **Qualification type** set to `Qualification using 3 samples`. In this case, the **Qualification sampling period**=`0`:



In the next figure **Qualification sampling period**=`1`. A sample is taken every two clock ticks, and the **Qualification type** is set to `Qualification using 3 samples`. The output signal changes much later than if **Qualification sampling period**=`0`.



In the following figure, **Qualification sampling period**=`2`. A sample is taken every four clock ticks, and the **Qualification type** is set to `Qualification using 3 samples`.



- `Asynchronous`—Using this qualification type, the signal is synchronized to an asynchronous event initiated by the software (CPU) via control register bits.

GPIOA, GPIOB, GPIOD, GPIOE input qualification sampling period

GPIO# Pull Up Disabled

Select this check box to disable the GPIO pull-up register. This option is available only for TI Concerto F28M35x/F28M36x processors.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-DMA_ch#

The Direct Memory Access (DMA) module transfers data directly between peripherals and memory using a dedicated bus, increasing overall system performance. In this case, # represents the DMA channel number.

You can individually enable and configure each DMA channel.

The DMA module services are event driven. Using the **Interrupt source** parameter, you can configure a wide range of peripheral interrupt event triggers. For more information, see the technical reference manual of your processor.

You can set the following parameters for DMA:

Enable DMA channel

Enable this parameter to edit the configuration of a specific DMA channel. This parameter does not have a corresponding bit or register.

Data size

Select the size of the data bit transfer.

The DMA read/write data buses are 32 bits wide. 32-bit transfers have twice the data throughput of a 16-bit transfer.

When providing DMA service to McBSP, set **Data size** to `16 bit`.

The following parameters are based on a 16-bit word size. If you set **Data size** to `32 bit`, double the value of the following parameters:

- Size: Burst
- Source: Burst step
- Source: Transfer step
- Source: Wrap step
- Destination: Burst step
- Destination: Transfer step
- Destination: Wrap step

Data size corresponds to bit 14 (DATASIZE) in the mode register (MODE).

Interrupt source

Select the peripheral interrupt that triggers a DMA burst for the specified channel.

Different C2000 processors have different interrupt trigger options that can be configured to trigger the DMA. Depending on the processor, the trigger sources include peripheral interrupts from ePWM, ADC, SPI, timer, and external interrupt. Some of these interrupt triggers such as TINT0 may require manual configuration. For external interrupt using GPIO, the configuration is done in the **External Interrupt** tab.

The **Interrupt source** parameter corresponds to bit 4–0 (PERINTSEL) in the mode register (MODE).

Burst size

Specify the number of 16-bit words in a burst, from 1 to 32. The DMA module must complete a burst before it can service the next channel.

Set the burst size for the peripheral DMA module services. For the ADC, the value equals the number of ADC registers used, up to 16. For multichannel buffered serial ports (McBSP), which lack FIFOs, the value is 1.

For RAM, the value can range from 1–32.

This parameter corresponds to bits 4–0 (BURSTSIZE) in the burst size register (BURST_SIZE).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Transfer size

Specify the number of bursts in a transfer, from 1–65536.

This parameter corresponds to bits 15–0 (TRANSFERSIZE) in the transfer size register (TRANSFER_SIZE).

Source begin address

Set the starting address for the current source address pointer. The DMA module points to this address at the beginning of a transfer and returns to it as specified by the **SRC wrap** parameter.

This parameter corresponds to bits 21–0 (BEGADDR) in the active source begin register (SRC_BEG_ADDR).

Destination begin address

Set the starting address for the current destination address pointer. The DMA module points to this address at the beginning of a transfer and returns to it as specified by the **DST wrap** parameter.

This parameter corresponds to bits 21–0 (BEGADDR) in the active destination begin register (DST_BEG_ADDR).

Source burst step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next burst. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Burst step** to 0. For example, because McBSP does not use FIFO, configure DMA to maintain the sequence of the McBSP data by moving each word of the data individually.

Accordingly, when you use DMA to transmit or receive McBSP data, set **Burst size** to 1 word and **Burst step** to 0.

This parameter corresponds to bits 15-0 (SRCBURSTSTEP) in the source burst step size register (SRC_BURST_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination burst step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next burst. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Burst step** to 0. For example, because McBSP does not use FIFO, configure DMA to maintain the sequence of the McBSP data by moving each word of the data individually. Accordingly, when you use DMA to transmit or receive McBSP data, set **Burst size** to 1 word and **Burst step** to 0.

This parameter corresponds to bits 15–0 (DSTBURSTSTEP) in the destination burst step size register (DST_BURST_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Source transfer step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next transfer. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Transfer step** to 0.

This parameter corresponds to bits 15–0 (SRCTRANSFERSTEP) source transfer step size register (SRC_TRANSFER_STEP).

If DMA is configured to perform memory wrapping (**SRC wrap** enabled), the corresponding source **Transfer step** does not alter the results.

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination transfer step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next transfer. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Transfer step** to 0.

This parameter corresponds to bits 15–0 (DSTTRANSFERSTEP) destination transfer step size register (DST_TRANSFER_STEP).

If DMA is configured to perform memory wrapping (**DST wrap** enabled), the corresponding destination **Transfer step** does not alter the results.

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Source wrap size

Specify the number of bursts before returning the current source address pointer to the **Source Begin Address** value. To disable wrapping, enter a value that is greater than the **Transfer** value.

This parameter corresponds to bits 15-0 (SRC_WRAP_SIZE) in the source wrap size register (SRC_WRAP_SIZE).

Destination wrap size

Specify the number of bursts before returning the current destination address pointer to the **Destination Begin Address** value. To disable wrapping, enter a value that is greater than the **Transfer** value.

This parameter corresponds to bits 15-0 (DST_WRAP_SIZE) in the destination wrap size register (DST_WRAP_SIZE).

Source wrap step

Set the number of 16-bit words using which the SRC_BEG_ADDR address pointer is incremented or decremented when a wrap event occurs. Enter a value from –4096 (decrement) to 4095 (increment).

This parameter corresponds to bits 15–0 (WRAPSTEP) in the source wrap step size registers (SRC_WRAP_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination wrap step

Set the number of 16-bit words using which the DST_BEG_ADDR address pointer is incremented or decremented when a wrap event occurs. Enter a value from –4096 (decrement) to 4095 (increment).

This parameter corresponds to bits 15–0 (WRAPSTEP) in the destination wrap step size registers (DST_WRAP_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Set channel 1 to highest priority

This parameter is available only for DMA_ch1.

Enable this option when DMA channel 1 is configured to handle high-bandwidth data, such as ADC data, and the other DMA channels are configured to handle lower-priority data. When enabled, the DMA module services each enabled channel sequentially until it receives a trigger from channel 1. Upon receiving the trigger, DMA interrupts its service to the current channel at the end of the current word, services the channel 1 burst that generated the trigger, and then continues servicing the current channel at the beginning of the next word.

Disable this channel to give each DMA channel equal priority, or if DMA channel 1 is the only enabled channel. When disabled, the DMA module services each enabled channel sequentially.

This parameter corresponds to bit 0 (CH1PRIORITY) in the priority control register 1 (PRIORITYCTRL1).

Enable first DMA event to trigger the full transfer (one shot mode)

Enable this parameter to have the DMA channel complete an entire *transfer* in response to an interrupt event trigger.

This option allows a single DMA channel and peripheral to dominate resources, and may streamline processing, but it also creates the potential for resource conflicts and delays.

Disable this parameter to have DMA complete one *burst* per channel per interrupt.

This parameter appears only when **Set channel 1 to highest priority** is disabled.

Synchronize ADC interrupt event triggers to DMA wrap counter (sync mode)

Enable this parameter to reset the DMA wrap counter when the **Interrupt source** is set to SEQ1INT and sends the ADCSYNC signal to the DMA wrap counter. This way, the wrap counter and the ADC channels remain synchronized with each other.

If **Interrupt source** is not set to SEQ1INT, **Sync enable** does not alter the results.

This parameter corresponds to bit 12 (SYNCE) of the mode register (MODE).

Do not disable the DMA channel after the transfer is complete (continuous mode)

Select this parameter to leave the DMA channel enabled upon completing a transfer. The channel waits for the next interrupt event trigger.

Clear this parameter to disable the DMA channel upon completing a transfer. The DMA module disables the DMA channel by clearing the RUNSTS bit in the control register when it completes the transfer. To use the channel again, first reset the RUN bit in the control register.

Enable destination sync mode

Enabling this parameter resets the destination wrap counter (DST_WRAP_COUNT) when **Sync enable** is enabled and the DMA module receives the SEQ1INT interrupt/ ADCSYNC signal.

Disabling this parameter resets the source wrap counter (SCR_WRAP_COUNT) when the DMA module receives the SEQ1INT interrupt/ADCSYNC signal.

This parameter is associated with bit 13 (SYNCSEL) in the mode register (MODE).

This parameter appears only when **Synchronize ADC interrupt event triggers to DMA wrap counter (sync mode)** is selected.

Generate interrupt

Enable this parameter to have the DMA channel send an interrupt to the CPU via the PIE at the beginning or end of a data transfer.

This parameter corresponds to bit 15 (CHINTE) and bit 9 (CHINTMODE) in the mode register (MODE).

Enable overflow interrupt

Enable this parameter to have the DMA channel send an interrupt to the CPU via PIE if the DMA module receives a peripheral interrupt while a previous interrupt from the same peripheral is waiting to be serviced.

This parameter is used for debugging during the development phase of a project.

The **Enable overflow interrupt** parameter corresponds to bit 7 (OVRINTE) of the mode register (MODE), and involves the overflow flag bit (OVRFLG) and peripheral interrupt trigger flag bit (PERINTFLG).

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

C28x-EMIF

Use the external memory interface (EMIF) to connect the C2000 processor to an external synchronous or asynchronous memory.

For C2000 processors, the EMIF is supported for these memory devices:

- Synchronous memory interface — JESD21-C SDR SDRAM
- Asynchronous memory interface — SRAM, NOR Flash, or any external device

Based on the processors, the number of EMIF modules supported varies. When you configure the EMIF interface based on the memory used, the GPIO pins required for interacting with the memory through EMIF are also configured. You must ensure that these GPIO pins are not used with other peripherals or as input/output because these pins are not included in the existing conflict check.

The EMIF1 pin configuration for synchronous and asynchronous memory is:

- GPIO38 – GPIO52 (except GPIO42 and GPIO43) are configured as address pins A0 – A12.
- GPIO86 and GPIO87 are configured as address pins A13 and A14 only when asynchronous memory is selected. GPIO86 and GPIO87 are configured as row and column address select (RAS and CAS) when synchronous memory is selected.
- GPIO69 – GPIO85 are configured as data pins D15 – D0. GPIO53 – GPIO68 are configured as data pins D31 – D16 only for 32-bit memory configuration.
- GPIO88 – GPIO91 are configured as data mask pins DQM0 – DQM3. You can manually configure these pins using custom code as address pins A15 – A18 when the EMIF is configured only for 8-bit asynchronous memory.
- GPIO92 and GPIO93 are configured as banks BA1 and BA0.
- GPIO28 – GPIO37 are configured as chip select (CS2, CS3, and CS4), clock enable (SDCKE), clock (CLK), write enable (WE), read and write control (RNW), wait pin (WAIT), and enable pin (OE).

The EMIF2 pin configuration for synchronous and asynchronous memory is:

- GPIO98 – GPIO109 are configured as address pins A0 – A11.
- GPIO53 – GPIO68 are configured as data pins D15 – D0.

An error message is displayed if the EMIF2 CS# is selected when the EMIF1 is configured for 32-bit data width because the same GPIO pins are used as D31 – D16 for the EMIF1 in 32-bit configuration.

- GPIO96 – GPIO97 are configured as data mask pins DQM0 – DQM1.
- GPIO111 and GPIO112 are configured as banks BA1 and BA0.
- GPIO110 and GPIO113 – GPIO121 are configured as chip select (CS0 and CS2), row and column address select (RAS and CAS), clock enable (SDCKE), clock (CLK), write enable (WE), read and write control (RNW), wait pin (WAIT), and enable pin (OE).

You can set these parameters for the EMIF:

EMIF clock divider (EMIF#CLKDIV)

Select the clock divider for the EMIF# module clock generation. In this case, # represents the number of the EMIF module. The EMIF clock frequency is based on SYSCLKOUT.

Enable CS0 for Synchronous memory

Select the chip select (CS0) to interface with the synchronous dynamic RAM (SDRAM). Synchronous memory supports the following memory sizes and addresses:

- EMIF1_CS0 — Data memory of size 256M × 16 with an address range of 0x80000000 to 0x8FFFFFFF
- EMIF2_CS0 — Data memory of size 3M × 16 with an address range of 0x90000000 to 0x91FFFFFF

Creation and usage of variables in SDRAM require the use of volatile qualifier and far attribute. Use `#pragma` to place the variables in SDRAM memory sections. Custom storage classes EM1_CS0_MEMORY and EM2_CS0_MEMORY are created in the signal object class `tic2000demospkg.Signal` to handle these requirements. You can use these custom storage classes to create variables using the Data Store Memory blocks.

Enable CS# for Asynchronous memory

Select the chip select (CS2/CS3/CS4) to interface with the asynchronous memory (SRAM / NOR Flash). Asynchronous memory supports the following memory sizes and addresses:

- EMIF1_CS2 — Data memory of size 2M × 16 with an address range of 0x00100000 to 0x002FFFFF
- EMIF1_CS3 — Data memory of size 512k × 16 with an address range of 0x00300000 to 0x0037FFFF

- EMIF1_CS4 — Data memory of size 393k × 16 with an address range of 0x00380000 to 0x003DFFFF
- EMIF2_CS2 — Data memory of size 4k × 16 with an address range of 0x00002000 to 0x00002FFF

Use #*pragma* to place the variables in asynchronous memory sections. Custom storage classes EM1_CS2_MEMORY, EM1_CS3_MEMORY, EM1_CS4_MEMORY, and EM2_CS2_MEMORY are created in the signal object class `tic2000demospkg.Signal` to handle these requirements. You can use these custom storage classes to create variables using the Data Store Memory blocks.

SDRAM column address bits

Select the value of the column address bits, thereby selecting the required page size of the connected SDRAM. Column address bits 8, 9, 10, and 11 corresponding to 256, 512, 1024, and 2048-word pages are supported.

The parameter **Page size = (2^column address bits)** is calculated based on the **SDRAM column address bits** parameter value.

Number of internal SDRAM banks

Select the number of memory banks inside the connected SDRAM. SDRAM with 1, 2, and 4 banks are supported.

SDRAM data bus width in bits

Select the data bus width of the connected SDRAM. Data bus widths of 16- and 32-bit are supported.

Refresh to active command delay cycles (T_RFC)

The minimum number of EM#CLK cycles from the refresh or load mode command to the refresh or activate command in the connected SDRAM. In this case, # represents 1 or 2. Some devices refer to this parameter as minimum auto refresh period.

The parameter **t_rfc in ns = (T_RFC+1)/fEM#CLK** is calculated based on the **Refresh to active command delay cycles (T_RFC)** parameter value.

Row precharge to active command delay cycles (T_RP)

The minimum number of EM#CLK cycles required from the row precharge command to the activate or refresh command in the connected SDRAM.

The parameter **t_rp in ns = (T_RP+1)/fEM#CLK** is calculated based on the **Row precharge to active command delay cycles (T_RP)** parameter value.

Active to read or write command delay cycles (T_RCD)

The minimum number of EM#CLK cycles from the activate command to the read or write command in the connected SDRAM.

The parameter **t_rcd in ns = (T_RCD+1)/fEM#CLK** is calculated based on the **Active to read or write command delay cycles (T_RCD)** parameter value.

Last write to row precharge command delay cycles (T_WR)

The minimum number of EM#CLK cycles from the last write transfer or last data in command to the row precharge command in the connected SDRAM.

The parameter **t_wr in ns = (T_WR+1)/fEM#CLK** is calculated based on the **Last write to row precharge command delay cycles (T_WR)** parameter value.

Active to precharge command delay cycles (T_RAS)

The minimum number of EM#CLK cycles from the activate command to the row precharge command in the connected SDRAM.

The parameter **t_ras in ns = (T_RAS+1)/fEM#CLK** is calculated based on the **Active to precharge command delay cycles (T_RAS)** parameter value.

Active to active command delay cycles (T_RC)

The minimum number of EM#CLK cycles from an activate command to the next activate command in the same bank in the connected SDRAM. This is also known as the minimum auto refresh period.

The parameter **t_rc in ns = (T_RC+1)/fEM#CLK** is calculated based on the **Active to active command delay cycles (T_RC)** parameter value.

Active one bank to active another bank command delay cycles (T_RRD)

The minimum number of EM#CLK cycles from an activate command in one bank to an activate command in a different bank in the connected SDRAM.

The parameter **t_rrd in ns = (T_RRD+1)/fEM#CLK** is calculated based on the **Active one bank to active another bank command delay cycles (T_RRD)** parameter value.

Self-refresh exit to other command delay cycles (T_XSR)

The minimum number of EM#CLK cycles from the self refresh exit command to any other command in the connected SDRAM.

The parameter **t_xsr in ns = (T_XSR+1)/fEM#CLK** is calculated based on the **Self-refresh exit to other command delay cycles (T_XSR)** parameter value.

SDRAM refresh period (tRefreshPeriod) in ms

REFRESH_RATE for SDRAM defines the rate at which the connected SDRAM refreshes. SDRAM refresh rate depends on the values of **SDRAM refresh period (tRefreshPeriod) in ms** and **SDRAM refresh cycle (ncycles)**. Enter the SDRAM refresh period and SDRAM refresh cycles from the SDRAM datasheet. The SDRAM refresh rate is calculated based on the formula *tRefreshPeriod * EMIF clock frequency / ncycles*.

SDRAM CAS Latency

Select the CAS latency required to access the connected SDRAM. SDRAM devices with CAS latencies of 2 and 3 are supported.

Asynchronous mode

Select the asynchronous mode for the connected asynchronous memory. These are the available modes:

- Normal — The byte enable will be active during the entire asynchronous cycle.
- Strobe — The byte enable will be active only during the strobe period of the access cycle mode.

Asynchronous data bus width in bits

Select the data bus width of the connected asynchronous memory. Asynchronous memory data bus width of 8-, 16-, and 32-bit are supported.

Read strobe setup cycles (R_SETUP)

The number of EM#CLK cycles from the EMIF chip select to the pin enable for asynchronous memory assert.

The parameter **t_r_setup in ns = (R_SETUP+1)/fEM#CLK** is calculated based on the **Read strobe setup cycles (R_SETUP)** parameter value.

Read strobe duration cycles (R_STROBE)

The number of EM#CLK cycles during which the pin enable for the asynchronous memory is held active.

The parameter **t_r_strobe in ns = (R_STROBE+1)/fEM#CLK** is calculated based on the **Read strobe duration cycles (R_STROBE)** parameter value.

Read strobe hold cycles (R_HOLD)

The number of EM#CLK cycles during which the EMIF chip select is held active after pin enable for the asynchronous memory is deasserted.

The parameter **t_r_hold in ns = (R_HOLD+1)/fEM#CLK** is calculated based on the **Read strobe hold cycles (R_HOLD)** parameter value.

Write strobe setup cycles (W_SETUP)

The number of EM#CLK cycles from the EMIF chip select to the write enable for the asynchronous memory assert.

The parameter **t_w_setup in ns = (W_SETUP+1)/fEM#CLK** is calculated based on the **Read strobe hold cycles (R_HOLD)** parameter value.

Write strobe duration cycles (W_STROBE)

The number of EM#CLK cycles during which the write enable for the asynchronous memory is held active.

The parameter **t_w_strobe in ns = (W_STROBE+1)/fEM#CLK** is calculated based on the **Write strobe duration cycles (W_STROBE)** parameter value.

Write strobe hold cycles (W_HOLD)

The number of EM#CLK cycles during which the EMIF chip select is held active after write enable for the asynchronous memory is deasserted.

The parameter **t_w_hold in ns = (W_HOLD+1)/fEM#CLK** is calculated based on the **Write strobe hold cycles (W_HOLD)** parameter value.

Turn around cycles (TA)

The number of EM#CLK cycles between the end of one asynchronous memory access and the start of another asynchronous memory access. This delay is not incurred between a read followed by a read or a write followed by a write to the same chip select.

Enable extended wait mode

Select this option to enable the extended wait option for the asynchronous memory. This option can be used if extended asynchronous wait cycles are required based on the EM#WAIT pin.

Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0-255]

This option is enabled if extended wait for any of the asynchronous memory CS# is enabled. Based on the value entered, the EMIF waits for *(MAX_EXT_WAIT+1) * 16* clock cycles before the asynchronous cycle is terminated.

Pin polarity of extended wait

Select the option to make the EMIF wait if the pin is low or high. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable wait rise interrupt

Select this option to get an interrupt based on the detection of a rising edge on the EM#WAIT pin. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable timeout interrupt

Select this option to get an interrupt when the EM#WAIT pin does not become inactive within the number of cycles defined in **Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255]**. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable line trap interrupt

Select this option to get an interrupt when there is an invalid cache line size or illegal memory access.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

C28x-LIN

You can configure the LIN Transmit and LIN Receive blocks within a model.

For detailed information on LIN module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments website.

LIN Module clock frequency (LM_CLK = SYSCLKOUT/2) in MHz

Display the frequency of the LIN module clock in MHz.

Enable loopback

Enables LIN loopback testing. When this option is enabled, the LIN module does the following:

- Internally redirects the LINTX output to the LINRX input.
- Places the external LINTX pin in a high state.
- Places the external LINRX pin in a high impedance state.

The default is disabled.

Suspension mode

Use this option to configure how the LIN state machine behaves while you debug the program on an emulator. The available options are:

- `Hard_abort`—Halts the transmissions and counters when you enter the LIN debug mode. The transmissions and counters resume when you exit LIN debug mode.
- `Free_run`—Allows completion of the current transmit and receive functions when you enter the LIN debug mode.

Parity mode

Use this option to configure parity checking. The available options are:

- `None`—Disables parity.
- `Odd`—Sets the parity bit to one if you have an odd number of ones in your bytes, such as 00110010.
- `Even`—Sets the parity bit to one if you have an even number of ones in your bytes, such as 00110011.

The default is `None`.

For **ID parity error interrupt** in the LIN Receive block to generate interrupts, enable **Parity mode**.

Frame length bytes

Set the number of data bytes in the response field, from 1–8 bytes.

The default is `8` bytes.

Baud rate prescaler (P: 0-16777215)

To set the LIN baud rate manually, enter a prescaler value, from 0–16777215. Click **Apply** to update the **Baud rate** display.

The default is `15`.

For more information, see the “Baud Rate” topic in the Texas Instruments document, *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2.

Baud rate fractional divider (M: 0–15)

To set the LIN baud rate manually, enter a fractional divider value, from 0–15. Click **Apply** to update the **Baud rate** display.

The default is `4`.

For more information, see the “Baud Rate” topic in the Texas Instruments document, *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2.

Baud rate (FLINCLK = LM_CLK/(16*(P+1+M/16)) in bits/sec

Display the baud rate. For more information, see “Setting the LIN baud rate”.

Communication mode

Enable or disable the LIN module from using the ID-field bits ID4 and ID5 for length control.

The default is `ID4 and ID5 not used for length control`.

Data byte order

Set the “endianness” of the LIN message data bytes.

The default is `Little_Endian`.

Data swap width

Set the width for data swap. If you set **Data byte order** to `Big_Endian`, the only available option for **Data swap width** is `8_bits`.

Pin assignment (Tx)

Map the LINTX output to a specific GPIO pin.

The default is `GPIO9`.

Pin assignment (Rx)

Map the LINRX input to a specific GPIO pin.

The default is `GPIO11`.

LIN mode

Set the LIN module as a master or a slave. The default is `Slave`.

In master mode, the LIN node can transmit queries and commands to slaves. In slave mode, the LIN module responds to queries or commands from a master.

This option corresponds to the CLK_MASTER field in the SCI Global Control Register (SCIGCR1).

ID filtering

Select the type of mask filtering comparison the LIN module performs.

If you select `ID byte`, the module uses the RECID and ID-BYTE fields in the LINID register to detect a match. If you select this option and enter `0xFF` for LINMASK, the LIN module does not report matches.

If you select `ID slave task`, the module uses the RECID an ID-SlaveTask byte to detect a match. If you select this option and enter `0xFF` for LINMASK, the LIN module reports matches.

The default is `ID slave task byte`.

ID byte

If you set **ID filtering** as `ID byte`, use this option to set the ID BYTE, also known as the “LIN mode message ID”.

In master mode, the CPU writes this value to initiate a header transmission. In slave mode, the LIN module uses this value to perform message filtering.

The default is `0x3A`.

ID slave task byte

If you set **ID filtering** to `ID slave task byte`, use this option to set the ID-SlaveTask BYTE. The LIN node compares this byte with the received ID and determines whether to transmit or receive.

The default is `0x30`.

Checksum type

If you select `Classic`, the LIN node generates the checksum field from the data fields in the response.

If you select `Enhance`, the LIN node generates the checksum field from both the ID field in the header and data fields in the response. LIN 1.3 supports classic checksums only. LIN 2.0 supports both classic and enhanced checksums.

The default is `Classic`.

Enable multibuffer mode

When you select this check box, the LIN node uses transmit and receive buffers instead of just one register. This setting affects various other LIN registers, such as: checksums, framing errors, transmitter empty flags, receiver ready flags, and transmitter ready flags.

The default is selected.

Enable baud rate adapt mode

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the slave node automatically adjusts its baud rate to match that of the master node. For this feature to work, first set the **Baud rate prescaler** and **Baud rate fractional divider**.

If you disable this option, the LIN module sets a static baud rate based on the **Baud rate prescaler** and **Baud rate fractional divider**.

The default is not selected.

Inconsistent synch field error interrupt

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the slave node generates interrupts when it detects irregularities in the synch field. This option is only relevant if you enable **Enable adapt mode**.

The default is `Disabled`.

No response error interrupt

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the LIN module generates an interrupt if it does not receive a complete response from the master node within a timeout period.

The default is `Disabled`.

Timeout after 3 wakeup signals interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt when it sends three wakeup signals to the master node and does not receive a header in response. The slave waits 1.5 seconds before sending another series of wakeup signals.

This interrupt indicates that the master node is having a problem recovering from low-power or sleep mode.

The default is `Disabled`.

Timeout after wakeup signal interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt when it sends a wakeup signal to the master node and does not receive a header in response. The slave waits 150 milliseconds before sending another series of wakeup signals.

This interrupt indicates that the master node is delayed recovering from low-power or sleep mode.

The default is `Disabled`.

Timeout interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt after 4 seconds of inactivity on the LIN bus.

The default is `Disabled`.

Wakeup interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When you enable this option:

- In low-power mode, a LIN slave node generates a wakeup interrupt when it detects the falling edge of a wake-up pulse or a low level on the LINRX pin.
- A LIN slave node that is “awake” generates a wakeup interrupt if it receives a request to enter low-power mode while it receives data.
- A LIN slave node that is “awake” does not generate a wakeup interrupt if it receives a wakeup pulse.

The default is `Disabled`.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

External Interrupt

External interrupts can be generated using GPIO pins.

XINT# GPIO

Select the GPIO pins for triggering external interrupts. In this case, # represents the number of the external interrupt.

XINT# Polarity

Select the polarity of external interrupts. In this case, # represents the number of the external interrupt.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

External Mode

Allows you to do external mode settings for your model.

Communication interface

Use the `serial` option to run your model in external mode with serial communication.

Serial port

Enter the COM port used by the target hardware.

To know the COM port used by the target hardware on your computer, see “Monitor and Tune over Serial Communication”.

Verbose

Select this to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.

See Also

More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

Execution profiling

Number of profiling samples to collect

Enter the number of profiling samples to collect. Using execution profiling, you can record the execution time, count the assembler instruction, and the high-level statement in the generated code.

See Also

More About

- "Hardware Implementation Pane: Texas Instruments C2000 Processors" on page 1-2

SD Card Logging

Use the SD card logging to log signals to SD card mounted on C2000 hardware.

Enable MAT-file logging on SD card

Enables the MAT-file logging for SD card.

SPI module

Select the desired interface on which the SD card is connected to hardware board.

C2000 hardware boards allow SD card to be interfaced through SPI. The SPI module will run in **Master** mode. For the SPI module, **Clock polarity** will be automatically set to **Falling_edge**, **Clock phase** value will be automatically set to **No_delay** and **Data bits** will be automatically set to **8**. It is advisable not to use the SPI module selected for SD card to perform any other operations through SPI master write, SPI transmit and SPI receive blocks as these may create issues in data logging on SD card.

Note For the selected SPI module, ensure that:

- Select the appropriate GPIO pins for **SIMO pin assignment**, **SOMI pin assignment**, **CLK pin assignment** and **STE pin assignment** in the corresponding **SPI_x** pane.
- GPIO pins are not used with other peripherals or as input/output because these pins are not included in the existing conflict check.

SPI baud rate

Select the desired option for the SPI interface used by the SD card.

The default is `Maximum achievable supported by the inserted SD Card`.

See Also

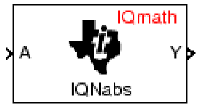
More About

- “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2

Blocks — Alphabetical List

C2000 Absolute IQN

Absolute value



Absolute IQN

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block computes the absolute value of an IQ number input. The output is also an IQ number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks® code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

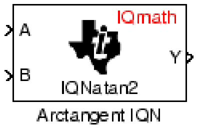
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

c2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Arctangent IQN

Four-quadrant arc tangent



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

The Arctangent IQN block computes the four-quadrant arc tangent of the IQ number inputs and produces IQ number output.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The Texas Instruments function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Type of arc tangent to calculate:

- `atan2` — Compute the four-quadrant arc tangent with output in radians with values from -pi to +pi.
- `atan2PU` — Compute the four-quadrant arc tangent per unit. If `atan2(B,A)` is greater than or equal to 0, `atan2PU(B,A) = atan2(B,A)/2*pi`. Otherwise, `atan2PU(B,A) = atan2(B,A)/2*pi+1`. The output is in per-unit radians with values from 0 to 2*pi radians.

Note The order of the inputs to the Arctangent IQN block correspond to the Texas Instruments convention, with argument 'A' at the top and 'B' at bottom.

References

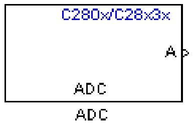
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C280x/C2833x ADC

Analog-to-Digital Converter (ADC)



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C280x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2833x

Description

The ADC block configures the ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices. With the C2833x, you can configure the ADC to use the processor DMA module to move data directly to memory without using the CPU. This frees the CPU to perform other tasks and increases overall system capacity.

Output

The output of the ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the ADC is 12-bit converter.

Modes

The ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module `A` or module `B` can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module `A` and module `B` are used for a single ADC block.

Parameters

ADC Control Pane

Module

Specifies which DSP module to use:

- `A` — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- `B` — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- `A and B` — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7).

Conversion mode

Type of sampling to use for the signals:

- `Sequential` — Samples the selected channels sequentially.
- `Simultaneous` — Samples the corresponding channels of modules A and B at the same time.

Start of conversion

Type of signal that triggers conversions to begin:

- `Software` — Signal from software. Conversion values are updated at each sample time.
- `ePWM#A` / `ePWM#B` / `ePWM#A_ePWM#B` — Start of conversion is controlled by user-defined PWM events.
- `XINT2_ADCSOC` — Start of conversion is controlled by the `XINT2_ADCSOC` external signal pin.

The choices available in **Start of conversion** depend on the **Module** setting. The following table summarizes the available choices. For each set of **Start of conversion** choices, the default is given first.

Module Setting	Start of Conversion Choices
`A`	`Software`, `ePWM#A`, `XINT2_ADCSOC`
`B`	`ePWM#B`, `Software`

Module Setting	Start of Conversion Choices
`A and B`	`Software`, `ePWM#A`, `ePWM#B`, `ePWM#A_ePWM#B`, `XINT2_ADCSOC`

Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. To execute this block asynchronously, set **Sample Time** to `-1`, check the **Post interrupt at the end of conversion** box.

To set different sample times for different groups of ADC channels, you must add separate ADC blocks to your model and set the desired sample times for each block.

Data type

Date type of the output data. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.

Post interrupt at the end of conversion

Select this check box to post an asynchronous interrupt at the end of the set of conversions. The interrupt is posted at the end of conversion. To execute this block asynchronously, set **Sample Time** to `-1`.

Input Channels Pane

Number of conversions

Number of ADC channels to use for analog-to-digital conversions.

Conversion no

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence. To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

Use multiple output ports

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/
F2837xS/F2838x/F28004x ePWM

“Configuring Acquisition Window Width for ADC Blocks”

C280x/C2802x/C2803x/C2805x/C2806x/ C2833x/C2834x/F28M3x/F2807x/F2837xD/ F2837xS/F2838x/F28004x ePWM

Generate enhanced Pulse Width Modulated (ePWM) waveforms

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2805x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C280x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2833x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2834x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2807x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xD

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xS

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2838x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F28004x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Use this block to generate ePWM waveforms. Multiple ePWM modules are available on C28x devices. Each module has two outputs, ePWMA and ePWMB.

When you enable the High-Resolution Pulse Width Modulator (HRPWM), the ePWM block uses the Scale Factor Optimizing (SFO) software library . The SFO library can “dynamically determine the number of MEP steps per SYSCLKOUT period.” For more information, consult *TMS320x28xx, 28xxx High-Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Literature Number SPRU924, available at the Texas Instruments Web site.

This block is common to various C28x devices. Some parameters may appear or disappear depending on which library the block is taken from. The blue label on the top right corner of the block displays the family that will be used to show parameters. As the block is a superset of functionalities available on different devices, some parameter selection maybe meaningless.

Parameters

General Pane

Allow use of 16 HRPWMs (for C28044) instead of 6 PWMs

Enable all 16 High-Resolution PWM modules (HRPWM) on the C28044 digital signal controller when the PWM resolution is too low.

For example, the Spectrum Digital eZdsp™ F28044 board has a system clock of 100 MHz (200-kHz switching). At these frequencies, conventional PWM resolution is too low—approximately 9 bits or 10 bits. By comparison, the HRPWM resolution for the same board is 14.8 bits.

All the ePWM blocks in your model become HRPWM blocks, Thus, when you enable this parameter:

- Use the HRPWM parameters under the ePWMA tab to make additional configuration changes.
- Most of the configuration parameters under the ePWMB tab are unavailable.
- Your model can contain up to 16 C280x/C2803x/C2833x ePWM blocks, provided you configure each one for a separate module. (For example, **Module** is `ePWM1`, `ePWM2`, and so on.)

For processors other than the C28044, deselect (disable) **Allow use of 16 HRPWMs (for C28044) instead of 6 PWMs**. To enable HRPWM for other processors, first determine how many HRPWM modules are available. Consult the Texas Instruments documentation for your processor, and then use the HRPWM parameters under the ePWMA tab to enable and configure HRPWM.

Module

Specify which ePWM module to use.

ePWMLink TBPRD

Select an ePWM module to which you want to link the current ePWM module for timer period. When you link timer period of an ePWM module with another, the **Timer period** value of the linked ePWM module is used in the current module. The **Timer period units**, **Specify timer period via** and the **Timer period** parameters do not appear when you select another ePWM module for linking.

However, the linking has no effect when you link an ePWM module to a module that does not exist in your model. This parameter is available only with some of the TI's C2000™ processors.

Timer period units

Specify the units of the **Timer period** or **Timer initial period** as `Clock cycles` (the default) or `Seconds`. When **Timer period units** is set to `Seconds`, the software converts the **Timer period** or **Timer initial period** from a value in seconds to a value in clock cycles. For best results, select `Clock cycles`. Doing so reduces calculations and rounding errors.

Note If you set **Timer period units** to `Seconds`, enable support for floating-point numbers. In the model window, select **Simulation > Model Configuration Parameters**.

In the Configuration Parameters dialog box, select Code Generation > Interface. Under **Software Environment**, enable **floating-point numbers**.

Specify timer period via

Configure the source of the timer period value. When you set this parameter to the `Input port` option, the **Timer period** parameter changes to **Timer initial period** and creates a timer period input port, **T**, on the block.

Timer period

Set the period of the ePWM counter waveform. The resultant ePWM waveform period depends on the settings of the **Action when counter=** parameters in the **ePWMx** tab.

When you enable HRPWM, you can enter a high-precision floating point value. The time-base period high resolution register (TBPRDHR) stores the high-resolution portion of the timer period value.

The timer period is calculated based on the **Counting mode** selection and **Timer period units**, as shown:

Count Mode	Timer period units	Calculation	Example
Up or Down	Clock cycles	The value entered in clock cycles is used to calculate time-base period (TBPRD) for the ePWM timer register. The period of the ePWM timer, T_{CTR} = (TBPRD + 1) * TBCLK. Where, T_{CTR} is the timer period in seconds, and TBCLK is the time-base clock.	For EPWMCLK frequency = 200 MHz, TBCLK = 5 ns. **Note** EPWMCLK will be equal to SYSCLKOUT or SYSCLKOUT/2 depending on the **EPWM clock divider (EPWMCLKDIV)** parameter setting. When the timer period is entered in clock cycles, TBPRD = 9999, and the ePWM timer period is calculated as, T_{CTR} = 50 µs. For the default action settings in the **ePWMx** tab, the ePWM period = 50 µs.

Count Mode	Timer period units	Calculation	Example
	Seconds	The value entered in seconds is used to calculate the time-base period (TBPRD) for the ePWM timer register. The TBPRD value entered in the register, TBPRD = (T_{CTR} / TBCLK) – 1. Where, T_{CTR} is the timer period in seconds, and TBCLK is the time-base clock. For the default action settings in the **ePWMx** tab, the ePWM period is the same as the timer period (in seconds) entered.	For EPWMCLK frequency = 200 MHz, TBCLK = 5 ns. When the timer period is entered in seconds, TBPRD = 9999, and the ePWM timer period is calculated as, T_{CTR} = 50 µs. For the default action settings in the **ePWMx** tab, the ePWM period = 50 µs.
Up-Down	Clock cycles	The value entered in clock cycles is used to calculate the time-base period (TBPRD) for the ePWM timer register. The period of the ePWM timer, T_{CTR} = 2 * TBPRD * TBCLK. Where, T_{CTR} is the timer period in seconds, and TBCLK is the time-base clock.	For EPWMCLK frequency = 200 MHz, TBCLK = 5 ns. When the timer period is entered in clock cycles, TBPRD = 10000, and the ePWM timer period is calculated as, T_{CTR} = 100 µs. For the default action settings in the **ePWMx** tab, the ePWM period = 100 µs.

Count Mode	Timer period units	Calculation	Example
	Seconds	The value entered in seconds is used to calculate the time-base period (TBPRD) for the ePWM timer register. The TBPRD value entered in the register, TBPRD = T_{CTR} / TBCLK. Where, T_{CTR} is the timer period in seconds, and TBCLK is the time-base clock. For the default action settings in the **ePWMx** tab, the ePWM period is two times the timer period (in seconds) entered.	For EPWMCLK frequency = 200 MHz, TBCLK = 5 ns. When the timer period is entered in seconds, TBPRD = 10000, and the ePWM timer period is calculated as, T_{CTR} = 50 µs. For the default action settings in the **ePWMx** tab, the ePWM period = 100 µs.

Timer initial period

The initial period of the waveform from the time the PWM peripheral starts operation until the ePWM input port, **T**, receives a new value for the period. Use **Timer period units** to measure the period in clock cycles or in seconds. The timer period is calculated similar to the **Timer period** parameter.

This parameter appears only when you set the **Specify timer period via** parmeter to the `Input port` option.

Reload for time base period register (PRDLD)

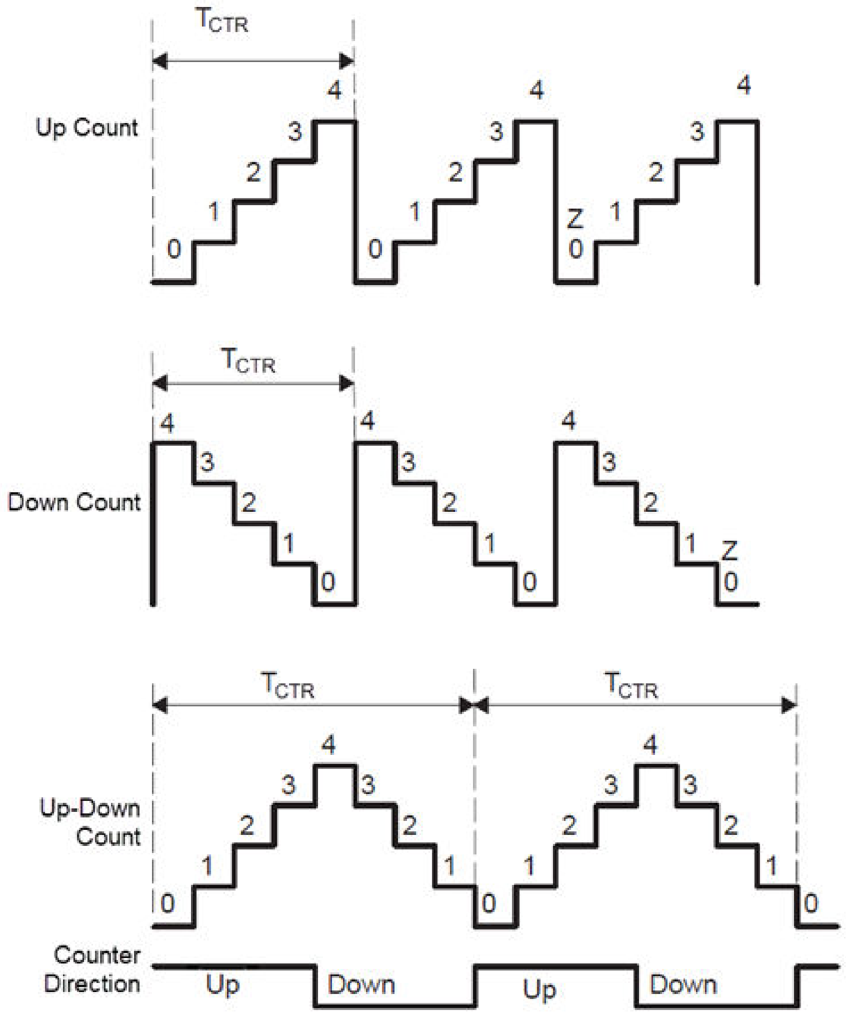
The time at which the counter period is reset.

- `Counter equals to zero` The counter period refreshes when the value of the counter is 0.
- `Immediate without using shadow` The counter period refreshes immediately.

Counting mode

Specify the counting mode in which to operate. This PWM module can operate in three distinct counting modes: `Up`, `Down`, and `Up-Down`. The `Down` option is not compatible with HRPWM. To avoid an error when you build the model, do not set the **Counting mode** parameter to `Down` and select the **Enable HRPWM (Period)** parameter checkbox.

The following illustration shows the waveforms that correspond to these three modes:



Synchronization action

Specify the source of a phase offset to apply to the Time-base synchronization input signal, EPWMxSYNCI from the **SYNC** input port. Selecting `Set counter to phase value specified via dialog` creates the **Phase offset value** parameter. Selecting `Set counter to phase value specified via input port` creates a phase input port, **PHS**, on the block. Selecting `Disable`, the default value prevents the application of phase offsets to the TB module.

Counting direction after phase synchronization

This parameter appears when **Counting mode** is `Up-Down` and **Synchronization action** is `Set counter to phase value specified via dialog` or `Input port`. Configure the timer to count up or down, following synchronization. This parameter corresponds to the PHSDIR field of the Time-base Control Register (TBCTL).

Phase offset value (TBPHS)

This field appears when you select `Set counter to phase value specified via dialog` in **Synchronization action**.

The offset value will be loaded in the Time Base Counter on a Synchronization event.

Note Enter the **Phase offset value (TBPHS)** in TBCLK cycles, from 0 to 65535. While using HRPWM, you may enter decimal values.

This parameter corresponds to the Time-Base Phase Register (TBPHS).

Specify software synchronization via input port (SWFSYNC)

Create an input port, **SYNC**, for a Time-base synchronization input signal, EPWMxSYNCI. You can use this option to achieve precise synchronization across multiple ePWM modules by daisy-chaining multiple Time-base (TB) submodules.

Enable digital compare A event1 synchronization (DCAEVT1)

This parameter only appears for specific C28x devices.

Synchronize the ePWM time base to a DCAEVT1 digital compare event. Use this feature to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**. This option is not compatible with HRPWM. Enabling HRPWM disables this option.

Enable digital compare B event1 synchronization (DCBEVT1)

This parameter only appears for specific C28x devices.

Synchronize the ePWM time base to a DCBEVT1 digital compare event. Use this feature to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset**

value. This option is not compatible with HRPWM. Enabling HRPWM disables this option.

Synchronization output (SYNCO)

This parameter corresponds to the SYNCOSEL field in the Time-Base Control Register (TBCTL).

Use this parameter to specify the event that generates a Time-base synchronization output signal, EPWMxSYNCO, from the Time-base (TB) submodule.

The available choices are:

- `Pass through (EPWMxSYNCI or SWFSYNC)` — a Synchronization input pulse or Software forced synchronization pulse, respectively. You can use this option to achieve precise synchronization across multiple ePWM modules by daisy chaining multiple Time-base (TB) submodules.
- `Counter equals to zero (CTR=Zero)` — Time-base counter equal to zero (TBCTR = 0x0000)
- `Counter equals to compare B (CTR=CMPB)` — Time-base counter equal to counter-compare B (TBCTR = CMPB)
- `Disable` — Disable the EPWMxSYNCO output (the default)

Time base clock (TBCLK) prescaler divider

Use the **Time base clock (TBCLK) prescaler divider** (CLKDIV) and the **High speed time base clock (HSPCLKDIV) prescaler divider** (HSPCLKDIV) to configure the Time-base clock speed (TBCLK) for the ePWM module. Calculate TBCLK using the following equation:

TBCLK in Hz = PWM clock in Hz/(HSPCLKDIV * CLKDIV)

For example, the default values of both CLKDIV and HSPCLKDIV are `1`, and the default frequency of PWM clock is 100 MHz, so:

TBCLK in Hz = 100 MHz/(1 * 1) = 100 MHz

TBCLK in seconds = 1/TBCLK in Hz = 1/100 MHz = 0.01 μs

The choices for the **Time base clock (TBCLK) prescaler divider** are: `1`, `2`, `4`, `8`, `16`, `32`, `64`, and `128`.

The **Time block clock (TBCLK) prescaler divider** parameter corresponds to the CLKDIV field of the Time-base Control Register (TBCTL).

Note The PWM clock is the SYSCLKOUT or a clock derived from SYSCLKOUT using the PWM Clock divider. For a few TI C2000 processors, there may be a PWM clock divider that divides the SYSCLKOUT to derive the PWM module clock. Check your processor's technical reference manual to know more details.

The frequency of SYSCLKOUT depends on the oscillator frequency and the configuration of PLL-based clock module. Changing the value of SYSCLOCKOUT affects the timing of all ePWM modules. If there is a PWM clock prescale available in the processor, changing its value also affects the PWM timing.

High speed time base clock (HSPCLKDIV) prescaler divider

See the **Time base clock (TBCLK) prescaler divider** topic for an explanation of the role of this value in setting the speed of the Time-base Clock. Choices are to divide by `1`, `2`, `4`, `6`, `8`, `10`, `12`, and `14`. Selecting **Enable high resolution PWM (HRPWM – period)** forces this option to `1`.

This parameter corresponds to the HSPCLKDIV field of the Time-base Control Register (TBCTL).

Enable swap module A and B

This parameter only appears for specific C28x devices.

Swap the ePWMA and ePWMB outputs. This option outputs the ePWMA signals on the ePWMB outputs and the ePWMB signals on the ePWMA outputs.

ePWMA and ePWMB panes

Each ePWM module has two outputs, ePWMA and ePWMB. The **ePWMA output** pane and **ePWMB output** pane include the same settings, although the default values vary in some cases, as noted.

Enable ePWM#x

Enables the ePWMA and/or ePWMB output signals for the ePWM module selected on the **General** pane. In this case, # represents the ePWM module and x represents A or B. By default, **Enable ePWM#A** is enabled, and **Enable ePWM#B** is disabled.

Note When you select **Enable ePWM#A** or **Enable ePWM#B**, enable support for floating-point numbers by browsing to **Configuration Parameters > Code Generation > Interface > Software Environment**.

CMPx initial value

This field appears when you set **CMPx source** to `Input port`. In this case, x represents A or B. Enter the initial pulse width of CMPA or CMPB that the PWM peripheral uses when it starts operation. Subsequent inputs to the **WA** or **WB** ports change the CMPA or CMPB pulse width.

Action when counter=ZEROAction when counter=period (PRD)Action when counter=CMPA on up-count (CAU)Action when counter=CMPA on down-count (CAD)Action when counter=CMPB on up-count (CBU)Action when counter=CMPB on down-count (CBD)

These settings along with the other remaining settings in the **ePWMA output** and **ePWMB output** panes, determine the behavior of the Action Qualifier (AQ) submodule. The AQ module determines which events are converted into various action types, producing the required switched waveforms of the ePWM#A and ePWM#B output signals.

For each of these four fields, the available choices are `Do nothing`, `Clear`, `Set`, and `Toggle`.

The default values for these fields vary between the **ePWMA output** and **ePWMB output** panes.

The following table shows the defaults for each of these panes when you set **Counting mode** to `Up` or `Up-Down`:

Action when counter =...	ePWMA output pane	ePWMB output pane
ZERO	Set	Clear
period (PRD)	Clear	Set
CMPA on up-count (CAU)	Clear	Set
CMPA on down-count (CAD)	Set	Do nothing
CMPB on up-count (CBU)	Do nothing	Clear
CMPB on down-count (CBD)	Do nothing	Set

The following table shows the defaults for each of these panes when you set **Counting mode** to `Down`:

Action when counter =...	ePWMA output pane	ePWMB output pane
ZERO	Do nothing	Do nothing
period (PRD)	Clear	Clear
CMPA on down-count (CAD)	Set	Do nothing
CMPB on down-count (CBD)	Do nothing	Set

For a detailed discussion of the AQ submodule, consult the *TMS320x280x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide* (SPRU791), available on the Texas Instruments Web site.

Compare value reload conditionAdd continuous software force input portContinuous software force logicReload condition for software force

These four settings determine how the action-qualifier (AQ) submodule handles the S/W force event, an asynchronous event initiated by software (CPU) via control register bits.

Compare value reload condition determines if and when to reload the Action-qualifier S/W Force Register from a shadow register. Choices are `Load on counter equals to zero (CTR=Zero)` (the default), `Load on counter equals to period (CTR=PRD)`, `Load on either`, and `Freeze`.

Add continuous software force input port creates an input port, **SFA**, which you can use to control the software force logic. Send one of the following values to **SFA** as an unsigned integer data type:

- 0 = `Forcing disable`: Do nothing. The default option.
- 1 = `Forcing low`: Clear low
- 2 = `Forcing high`: Set high

If you did not create the **SFA** input port, you can use **Continuous software force logic** to select which type of software force logic to apply. The choices are:

- `Forcing disable`: Do nothing. The default.
- `Forcing low`: Clear low
- `Forcing high`: Set high

Reload condition for software force — Choices are `Zero` (the default), `Period`, `Either period or zero`, and `Immediate`.

Inverted version of ePWM#A

This parameter only appears for specific C28x devices.

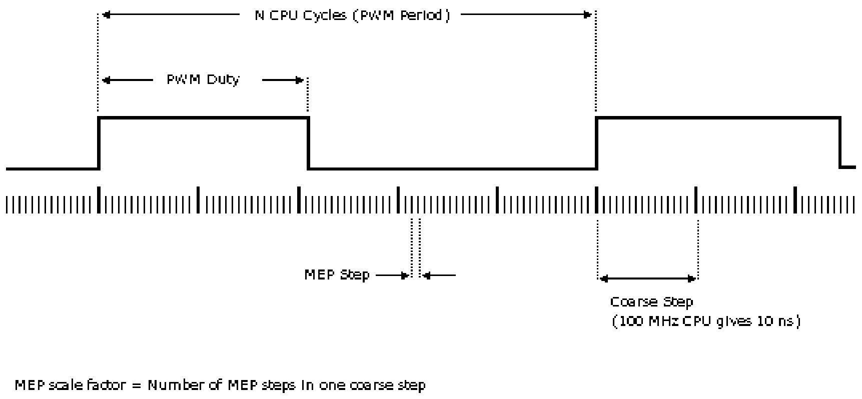
Invert the ePWM#A signal and output it on the ePWM#B outputs.

This parameter sets the SELOUTB field in the HRPWM Configuration Register (HRCNFG).

Enable high resolution PWM (HRPWM)

This parameter appears at this position in the C280x and C2833x ePWM blocks.

Select to enable High Resolution PWM settings. When the effective resolution for conventionally generated PWM is insufficient, consider High Resolution PWM (HRPWM). The resolution of PWM is normally dependent upon the PWM frequency and the underlying system clock frequency. To address this limitation, HRPWM uses **Micro Edge Positioner (MEP)** technology to position edges more finely by dividing each coarse system clock. The accuracy of the subdivision is on the order of `150ps`. The following figure shows the relationship between one system clock and edge position in terms of **MEP** steps:


N CPU Cycles (PWM Period)
PWM Duty
MEP Step
Coarse Step
(100 MHz CPU gives 10 ns)
MEP scale factor = Number of MEP steps in one coarse step

High resolution PWM (HRPWM) loading mode

This parameter appears at this position in the C280x and C2833x ePWM blocks.

Determine when to transfer the value of the CMPAHR shadow to the active register:

- Counter equals to zero (CTR=ZERO): Transfer the value when the time base counter equals zero (TBCTR = 0x0000).
- Counter equals to period (CTR=PRD): Transfer the value when the time base counter equals the period (TBCTR = TBPRD).
- CTR=Zero or CTR=PRD Transfer the value when either case is true.

High resolution PWM (HRPWM) control mode

This parameter appears at this position in the C280x and C2833x ePWM blocks.

Select which register controls the Micro Edge Positioner (MEP) step size. The **High resolution PWM (HRPWM) control mode** option configures the CTLMODE "Control Mode Bits".

- `Duty control mode` uses the Extension Register for HRPWM Duty (CMPAHR) or the Extension Register for HRPWM Period (TBPRDHR) to control the MEP edge position.
- Select `Phase control mode` to use the Time Base Period High-Resolution Register (TBPRDHR) to control the MEP edge position.

The **High resolution PWM (HRPWM) control mode** option configures the CTLMODE "Control Mode Bits" in the HRPWM Configuration Register (HRCNFG).

High resolution (HRPWM) edge control mode

This parameter appears at this position in the C280x and C2833x ePWM blocks.

Swap the ePWMA and ePWMB outputs. This parameter sets the SWAPAB field in the HRPWM Configuration Register (HRCNFG).

Use scale factor optimizer (SFO) software

Enable scale factor optimizing (SFO) software with HRPWM. This software dynamically determines the scaling factor for the Micro Edge Positioner (MEP) step size. The step size varies depending on operating conditions such as temperature and voltage. The SFO software reduces variability due to these conditions. For more information, see the "Scale Factor Optimizing Software (SFO)" section of the *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Literature Number: SPRUGE8.

Counter Compare Pane

The four compare registers—CMPA, CMPB, CMPC, and CMPD, are compared with the time-base counter value to generate appropriate events. CMPA and CMPB events are used for controlling PWM duty cycle by selecting appropriate actions in **ePWMA** and **ePWMB** panes. These events can also be used to generate an interrupt to the CPU and/or a start of conversion pulse to the ADC. You can refer to the **Event Trigger** pane to select the events to be triggered.

ePWMLink CMPx

Select an ePWM module to which you want to link the current ePWM module for counter value. In this case, x represents A, B, C, or D. When you link counter value of an ePWM module with another, the **CMPx value** value of the linked ePWM module is used in the current module. The **CMPx**, **Specify CMPx via**, and **CMPx value** parameters do not appear when you select another ePWM module for linking.

However, the linking has no effect when you link an ePWM module to a module that does not exist in your model. This parameter is available only with some of the TI's C2000 processors.

CMPx units

Specify the units used by the compare register: `Percentages` (the default) or `Clock cycles`. In this case, x represents A, B, C, or D.

Notes

- The term *clock cycles* refers to the time-base clock on the processor. See the **TB clock prescaler divider** topic for an explanation of time-base clock speed calculations.
- Percentages use additional computation time in generated code and can decrease accuracy of the results.
- If you set **CMPx units** to `Percentages`, enable support for floating-point numbers by browsing to **Configuration Parameters > Code Generation > Interface > Software Environment**.

Specify CMPx via

Specify the source of the pulse width. If you select `Specify via dialog` (the default), enter a value for the **CMPx value** parameter. If you select `Input port`, set the value using the input port, **Wx** on the block. If you select `Input port`, make sure to set the **CMPx initial value** parameter. In this case, x represents A, B, C, or D.

CMPx value

This field appears when you set **CMPx source** to `Specify via dialog`. Enter a value that specifies the pulse width, in the units specified in **CMPx units**. In this case, x represents A, B, C, or D.

Reload for compare x Register (SHDWxMODE)

The time at which the counter period is reset. In this case, x represents A, B, C, or D.

- `Counter equals to zero` — refreshes the counter period when the value of the counter is 0.
- `Immediate without using shadow` — refreshes the counter period immediately.

Deadband Unit Pane

The **Deadband unit** pane lets you specify parameters for the Dead-Band Generator (DB) submodule.

Use deadband for ePWM#AUse deadband for ePWM#B

Enables a deadband area of Rising Edge Delay or Falling Edge Delay cycles without signal overlap between pairs of ePWM output signals. This check box is cleared by default.

Enable half-cycle clocking

This parameter only appears for specific C28x devices.

To double the deadband resolution, enable half-cycle clocking. This option clocks the deadband counters at TBCLK*2. When you disable this option, the deadband counters use full-cycle clocking (TBCLK*1).

Deadband polarity

Configure the deadband polarity as `Active high (AH)` (the default option), `Active low (AL)` , `Active high complementary (AHC)` or `Active low complementary (ALC)`. During the Deadband time, both the ePWMA and ePWMB outputs have to set to an inactive state. Depending on your hardware settings, the inactive states can correspond to a high or a low logic value. Active high means that the system is active when the ePWM output is set to a high logic value. Active low means that the system is active when the ePWM output is set to a low logic value. Use the Complementary option when the B signal needs to be the inverse of A. For more information, refer to the ePWM Technical Reference guide of your processor.

Signal source for rising edge (RED)

Select the signal source to which rising edge delay (RED) has to be applied. By default ePWM#A signal is selected.

Signal source for falling edge (FED)

Select the signal source to which falling edge delay (FED) has to be applied. By default ePWM#A signal is selected.

Deadband period source

Specify the source of the control logic. Choose `Specify via dialog` (the default) to enter explicit values, or `Input port` to use a value from the input port.

Deadband Rising edge (RED) deadband period (0~16383)

The value you enter in the field specifies the dead band delay in time-base clock (TBCLK) cycles.

Deadband Falling edge (FED) deadband period (0~16383)

The value you enter in the field specifies the dead band delay in time-base clock (TBCLK) cycles.

Event Trigger Pane

Configure ADC Start of Conversion (SOC) by one or both of the ePWMA and ePWMB outputs.

Enable ADC start of conversion for module A

When you select this option, ADC Start of Conversion Event (ePWMSOCxA) is generated when the event selected in the **Start of conversion for module A event selection** parameter occurs.

Number of event for start of conversion for Module A (SOCA) to be generated

When you select **Enable ADC start of conversion for module A**, this field specifies the number of the event that triggers ADC Start of Conversion for Module A (SOCA): `First event` triggers ADC start of conversion with every event (the default). `Second event` triggers ADC start of conversion with every second event. `Third event` triggers ADC start of conversion with every third event.

Start of conversion for module A event selection

When you select **Enable ADC start of conversion for module A**, this field specifies the counter match condition that triggers an ADC start of conversion event. The choices are:

Digital Compare Module A Event 1 start of conversion (DCAEVT1.soc) and Digital Compare Module B Event 1 start of conversion (DCBEVT1.soc) (For specific C28x devices only)

When the ePWM asserts a DCAEVT1 or DCBEVT1 digital compare event. Use this feature to synchronize the selected PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**.

Counter equals to zero (CTR=Zero)

When the ePWM counter reaches zero (the default).

Counter equals to period (CTR=PRD)

When the ePWM counter reaches the period value.

Counter equals to zero or period (CTR=Zero or CTR=PRD)

When the time base counter reaches zero (TBCTR = 0x0000) or when the time base counter reaches the period (TBCTR = TBPRD).

Counter is incrementing and equals to the compare x register (CTRU=CMPx)

The ePWM counter reaches the compare value x on the way up. In this case, x represents A, B, C, or D.

Counter is decrementing and equals to the compare x register (CTRD=CMPx)

The ePWM counter reaches the compare value x on the way down. In this case, x represents A, B, C, or D.

Enable ADC start of conversion for module B

When you select this option, ADC Start of Conversion Event (ePWMSOCxB) is generated when the event selected in the **Start of conversion for module B event selection** parameter occurs.

Number of event for start of conversion for Module B (SOCB) to be generated

When you select **Enable ADC start module B**, this field specifies the number of the event that triggers ADC start of conversion: `First event` triggers ADC start of conversion with every event (the default), `Second event` triggers ADC start of conversion with every second event, and `Third event` triggers ADC start of conversion with every third event.

Start of conversion for module B event selection

When you select **Enable ADC start of conversion for module B**, this field specifies the counter match condition that triggers an ADC start of conversion event. The choices are the same as for **Module A counter match event condition**.

Enable ePWM interrupt

Select this option to generate interrupts based on different events defined by **Number of event for interrupt to be generated** and **Interrupt counter match event condition**. By default, the software clears (disables) this option.

Number of event for interrupt to be generated

When you select **Enable ePWM interrupt**, this field specifies the number of the event that triggers the ePWM interrupt: `First event` triggers ePWM interrupt with every event (the default), `Second event` triggers ePWM interrupt with every second event, and `Third event` triggers ePWM interrupt with every third event.

Interrupt counter match event condition

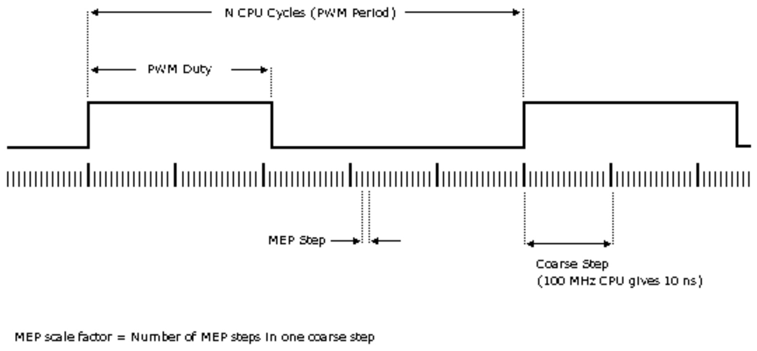
When you select **Enable ePWM interrupt**, this field specifies the counter match condition that triggers ePWM interrupt. The choices are the same as for **Module A counter match event condition**.

HRPWM Pane

Enable high resolution period on ePWM#A (HRPWM - period)Enable high resolution period on ePWM#B (HRPWM - period)

This parameter only appears for specific C28x devices.

When the effective resolution for conventionally generated PWM is insufficient, consider using High Resolution PWM (HRPWM). The resolution of PWM is normally dependent upon the PWM frequency and the underlying system clock frequency. To address this limitation, HRPWM uses **Micro Edge Positioner (MEP)** technology to position edges more finely by dividing each coarse system clock. The accuracy of the subdivision is on the order of `150ps`. The following figure shows the relationship between one system clock and edge position in terms of **MEP** steps:


N CPU Cycles (PWM Period)
PWM Duty
MEP Step
Coarse Step
(100 MHz CPU gives 10 ns)
MEP scale factor = Number of MEP steps in one coarse step

When this parameter is enabled, decimal values will be accepted for the timer period of the ePWM Module. The Extension Register for the HRPWM Period (TBPRDHR) provides an 8 bit representation of the decimal part of the **Timer period** value. This parameter enables the **Enable high resolution PWM (HRPWM - duty)** option, and displays the **HRPWM loading mode**, **HRPWM control mode**, and **HRPWM edge control** mode options. Also configure **HRPWM control mode**.

Selecting Enable HRPWM (Period) forces **TB clock prescaler divider** and **High Speed TB clock prescaler divider** to 1. These settings match the HRPWM time base clock with the SYSCLKOUT frequency.

The Down option in the **Counting mode** parameter is not compatible with HRPWM. To avoid an error when you build the model, do not set the **Counting mode** parameter to Down and select the **Enable HRPWM (Period)** parameter checkbox.

Enable HRPWM Duty on ePWM#A(HRPWM - duty)

This parameter only appears for specific C28x devices.

When this parameter is enabled, decimal values will be accepted for the Compare A value (CMPA) of the ePWM Module. The Extension Register for the HRPWM Compare A (CMPAHR) provides an 8 bit representation of the decimal part of the **Compare** value.

This parameter also enables **HRPWM control mode**.

High resolution PWM (HRPWM) loading mode on ePWM#A

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected. Determine when to transfer the value of the CMPAHR shadow to the active register:

- `Counter equals to zero (CTR=ZERO)` — transfers the value when the time base counter equals zero (TBCTR = 0x0000).
- `Counter equals to period (CTR=PRD)` — transfers the value when the time base counter equals the period (TBCTR = TBPRD).
- `Counter equals to either zero or period (CTR=ZERO or CTR=PRD)` — transfers the value when either case is true.

This option configures the HRLOAD “Shadow Mode Bit” in the HRPWM Configuration Register (HRCNFG).

High resolution PWM (HRPWM) control mode on ePWM#A

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected. Select which register controls the Micro Edge Positioner (MEP) step size. The **High resolution PWM (HRPWM) Control mode** option configures the CTLMODE “Control Mode Bits”.

- `Duty control mode` — uses the Extension Register for HRPWM Duty (CMPAHR) or the Extension Register for HRPWM Period (TBPRDHR) to control the MEP edge position.
- `Phase control mode` — uses the Time Base Phase High Resolution Register (TBPHSHR) to control the MEP edge position.

The **High resolution PWM (HRPWM) control mode** — configures the CTLMODE “Control Mode Bits” in the HRPWM Configuration Register (HRCNFG).

High resolution PWM (HRPWM) edge control mode

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected.

Select the register that controls the Micro Edge Positioner (MEP) step size.

- `Rising Edge` — MEP control of rising edge
- `Falling Edge` — MEP control of falling edge
- `Both Edge` — MEP control of both edges

The **High resolution PWM (HRPWM) Edge Control mode** option configures the EDGMODE “Edge Mode Bits” in the HRPWM Configuration Register (HRCNFG).

Use scale factor optimizer (SFO) software

This parameter is enabled, if the **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected.

Enable scale factor optimizing (SFO) software with HRPWM. This software dynamically determines the scaling factor for the Micro Edge Positioner (MEP) step size. The step size varies depending on operating conditions such as temperature and voltage. The SFO software reduces variability due to these conditions. For more information, see the “Scale Factor Optimizing Software (SFO)” section of the *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Literature Number: SPRUGE8.

Enable auto convert

This parameter only appears for specific C28x devices and if **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected.

Apply the scaling factor calculated by the SFO software to the controlling period or duty cycle. (Use the **HRPWM duty mode** to select controlling period or duty cycle.) This parameter sets the AUTOCONV field in the HRPWM Configuration Register (HRCNFG).

PWM Chopper Control Pane

The **PWM chopper control** pane lets you specify parameters for the PWM-Chopper (PC) submodule. The PC submodule uses a high-frequency carrier signal to modulate the PWM waveform generated by the AQ and DB modules.

Chopper module enable

Select to enable the chopper module. Use of the chopper module is optional, so this check box is cleared by default.

Chopper frequency divider

Set the prescaler value that determines the frequency of the chopper clock. The system clock speed is divided by this value to determine the chopper clock frequency. Choose an integer value from `1` to `8`.

Chopper clock cycles width of first pulse

Choose an integer value from `1` to `16` to set the width of the first pulse. This feature provides a high-energy first pulse for a hard and fast power switch turn on.

Chopper pulse duty cycle

The duty cycles of the second and subsequent pulses are also programmable. The duty cycle can be varied in steps of 12.5% from `12.5%` to `87.5%`.

Trip Zone Unit Pane

The **Trip Zone unit** pane lets you specify parameters for the Trip-zone (TZ) submodule. Each ePWM module receives TZ signals from the GPIO MUX. The number of Trip zone signals are different for C28x processor families. These signals can be used to force the ePWM output into a specific state based on an event like an external fault. Use the settings in this pane to program the ePWM outputs to respond to external events.

Trip zone source

Specify the source of the control logic for the Trip Zone signals. Select `Specify via dialog` (the default) to enable specific Trip-zone signals in the block dialog. Choose `Input port` to enable specific Trip-zone signals using a block input port, **TZSEL**.

If you select `Input port`, use the following bit operation to determine the value of the 16-bit integer to send to the **TZSEL** input port:

$\texttt{TZSEL INPUT VALUE = (OSHT6*}2^{13}\texttt{ + OSHT5*}2^{12}\texttt{ + OSHT4*}2^{11}\texttt{ + OSHT3*}2^{10}\texttt{ + OSHT2*}2^{9}\texttt{ + OSHT1*}2^{8}\texttt{ + CBC6*}2^{5}\texttt{ + CBC5*}2^{4}\texttt{ + CBC4*}2^{3}\texttt{ + CBC3*}2^{2}\texttt{ + CBC2*}2^{1}\texttt{ + CBC1*}2^{0}\texttt{)}$

The software uses the higher 8 bits for the **One shot TZ1-TZ6** (OSHT1–6) and the lower 8 bits for **Cyclic TZ1-TZ6** (CBC1–6). You can set up a group of TZ sources (1~6), use a bit operation to combine them into an integer, and then feed the integer to TZSEL.

For example, to enable One Shot TZ6 (OSHT6) and One Shot TZ5 (OSHT5) as trip zone sources, set OSHT6 and OSHT5 to “1” and leave the remaining values as “0”.

TZSEL INPUT VALUE = $(1*2^{13} + 1*2^{12} + 0*2^{11} \ldots)$

TZSEL INPUT VALUE = (8192 + 4096 + 0 …)

TZSEL INPUT VALUE = 12288

When the block receives this value, it applies it to the TZSEL register as a binary value: `11000000000000`.

For more information, see the ”Trip-Zone Submodule Control and Status Registers” section of the *TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRU791 on www.ti.com

Enable One-Shot Trip zone# (TZ#)

This option is only available when the **Trip zone source** is `Specify via dialog`. Select this check box to enable the corresponding Trip-zone signal in One-Shot Mode. In this mode, when the trip event is active, the Trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. You can unlatch the condition using software control.

Enable one-shot digital compare A event 1 (DCAEVT1)Enable one-shot digital compare B event 1 (DCBEVT1)

This option is only available when the **Trip zone source** is `Specify via dialog`. Select these check boxes to enable the corresponding event signal as a OST trip source for event 1. In this mode, if the digital compare A or digital compare B event 1 is active, the Trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. You can unlatch the condition using the software control. This parameter is available only for specific C28x processors.

Enable Cyclic Trip zone# (TZ#)

This option is only available when the **Trip zone source** is `Specify via dialog`. Select this check box to enable the corresponding Trip-zone signal in Cycle-by-Cycle Mode. In this mode, when the trip event is active, the Trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. In Cycle-by-Cycle Mode, the Trip zone module automatically clears condition when the ePWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, every ePWM cycle resets or clears the trip event.

Enable cyclic digital compare A event 2 (DCAEVT2)Enable cyclic digital compare B event 2 (DCBEVT2)

This option is only available when the **Trip zone source** is `Specify via dialog`. Select these check boxes to enable the corresponding event signal as a cyclic trip source for event 2. In this mode, if the digital compare A or digital compare B event 2 is active, the Trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. In Cycle-by-Cycle Mode, the Trip zone module automatically clears condition when the ePWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, every ePWM cycle resets or clears the trip event. This parameter is available only for specific C28x processors.

Enable Trip-zone One-Shot interrupt (OST)

Generate an interrupt when any of the enabled one shot (OST) triggering events occur.

Enable Trip-zone Cycle-by-Cycle interrupt (CBC)

Generate an interrupt when any of the enabled cyclic or cycle-by-cycle (CBC) triggering events occur.

Digital comparator output A/B event 1/2 interrupt enable (DCAEVT1, DCAEVT2, DCBEVT1, DCBEVT2)

These parameters are available only for specific C28x processors. Generate an interrupt when Digital Comparator Output A or Digital Comparator Output B for event 1 or 2 occurs.

ePWM#A forced (TZ) toePWM#B forced (TZ) toePWM#A forced (DCAEVT#) toePWM#B forced (DCBEVT#) to

These parameters decide the actions to take on the ePWM outputs upon a trip-zone condition. The Trip zone module overrides and forces the ePWM#A and/or ePWM#B (TZ or DCAEVTx) output to one of the following states: `No action` (the default), `High`, `Low`, or `Hi-Z (High Impedance)`.

Digital Compare

This pane is available only for specific C28x processors.

Use the **Digital Compare** pane to configure the Digital Compare (DC) submodule parameters.

Each digital compare (DC) submodule receives three TZ signals (TZ1 to TZ3) from the GPIO MUX, and three COMP signals from the COMP (For specific C28x devices only). These signals indicate fault or trip conditions that are external to the ePWM submodule. Use the settings in this pane to output specific DC events in response to those external signals. These DC events feed directly into the Time-base, Trip-zone, and Event-trigger submodules.

For more information, see the “Digital Compare (DC) Submodule” section of the *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRUGE9.

Source for digital compare A high signal (DCAH), **Source for digital compare B high signal (DCBH)**

If the TZ or COMP signal selected is true, DCAH/DCBH will be set to a high logic value. Use the **DCAEVT# source select**, **DCBEVT# source select** options to determine the impact of DCAH/DCBH on DCAEVT# and DCBEVT#.

Source for digital compare A low signal (DCAL), **Source for digital compare B low signal (DCBL)**

If the TZ or COMP signal selected is true, DCAL/DCBL will be set to a high logic value. Use the **DCAEVT# source select**, **DCBEVT# source select** options to determine the impact of DCAL/DCBL on DCAEVT# and DCBEVT#.

Digital compare output A event # selection (DCAEVT#), **Digital Compare output B event # selection (DCBEVT#)**

Qualify the signals that generate DC events, such as DCAEVT# or DCBEVT#. Select the states of **Source for digital compare A high signal DCAH**, **Source for digital compare B high signal DCBH**, **Source for digital compare A low signal (DCAL)**, and **Source for digital compare B low signal (DCBL)** that generate the event. To disable this feature, choose the `Event disabled` option.

DCAEVT# source select, **DCBEVT# source select**

This parameter controls two separate aspects of triggering DC events:

- Triggering filtered or unfiltered DC event.
  - Configures DCACTL[EVT1SRCSEL] or DCACTL[EVT2SRCSEL]
  - Configures DCBCTL[EVT1SRCSEL] or DCBCTL[EVT2SRCSEL]
- Trigger the DC event synchronously or asynchronously.
  - Configures DCACTL[EVT1FRCSYNCSEL] or DCACTL[EVT2FRCSYNCSEL]
  - Configures DCBCTL[EVT1FRCSYNCSEL] or DCBCTL[EVT2FRCSYNCSEL]

Filtering

- Options that begin with `DCAEVT# with sync` or `DCAEVT# with async` do not apply filtering to DC events. Qualified signals trigger DC events.
- Options that begin with `DCEVTFILT sync` apply filtering to DC events. Qualified signals pass through filtering circuits before triggering DC events. This filtering is not configurable in the ePWM block. For more information, refer to the “Event Filtering” section of the *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, Literature Number: SPRUGE9.

Synchronizing

- Options that end with `async` trigger DC events asynchronously. When the qualified or filtered signals exist, the DC submodule triggers the DC event immediately.

- Options that end with `sync` trigger DC events synchronously. Once the qualified or filtered signals exist, the DC submodule triggers the DC event in sync with the TBCLK signal.

Note The following fields appear when you select `DCEVTFILT with sync` or `DCEVTFILT with async` for the **DCAEVTX source select** or **DCBEVTX source select**.

For more details about the following parameters, refer to the sections:*TMS320x2806x Piccolo processor: 3.2.9.3.2 (Event Filtering) and Table 56 of Technical Reference Manual (SPRUH18C). TMS320x2802x/03x Piccolo processors : 2.9.3.2 (Event Filtering) and Table 56 of Enhanced Pulse Width Modulator (ePWM) Module Reference Guide (SPRUGE9E) for TMS320x2802x and TMS320x2803x Piccolo processors.*

Pulse select

The blanking window which filters out event occurrences on the signal while active, is aligned to either a CTR = PRD pulse or a CTR = 0 pulse.

Blanking window inverted

The option that allows you to Enable or Disable the Inverted Blanking window.

Blanking window offset

The number of TBCLK cycles required from the blanking window reference to the point when the blanking window is applied.

Blanking window width

The duration of the blanking window in terms of TBCLK.

Filter source select

The option that allows you to select a source for Filtering.

The available options are:

- Filtered version of DCAEVT1 (DCAEVT1FILT)
- Filtered version of DCAEVT2 (DCAEVT2FILT)
- Filtered version of DCBEVT1 (DCBEVT1FILT)
- Filtered version of DCBEVT2 (DCBEVT2FILT)

Enable counter capture

The option that allows you to Enable or Disable the time-base counter capture.

References

For more information, consult the following references, available at the Texas Instruments Web site:

- *TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, literature number SPRU791
- *TMS320x280x, 2801x, 2804x High Resolution Pulse Width Modulator Reference Guide*, literature number SPRU924
- *TMS320x2802x, 2803x Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*, literature number SPRUGE9
- *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, literature number SPRUGE8
- *TMS320x2805x Piccolo Technical Reference Manual*, literature number SPRUHE5
- *TMS320x2806x Piccolo Technical Reference Manual*, literature number SPRUH18
- *TMS320x28M35x Concerto Technical Reference Manual*, literature number SPRUH22
- *TMS320x28M36x Concerto Technical Reference Manual*, literature number SPRUHE8
- *Using the ePWM Module for 0% - 100% Duty Cycle Control Application Report*, literature number SPRU791
- *Configuring Source of Multiple ePWM Trip-Zone Events*, literature number SPRAAR4
- *TMS320F2809, TMS320F2808, TMS320F2806 TMS320F2802, TMS320F2801 TMS320C2802, TMS320C2801, and TMS320F2801x DSPs Data Manual*, literature number SPRS230
- *TMS320F28044 Digital Signal Processor Data Manual*, literature number SPRS357
- *TMS320F28335/28334/28332 TMS320F28235/28234/28232 Digital Signal Controllers (DSCs) Data Manual*, literature number SPRS439

See Also

"ADC-PWM Synchronization Using ADC Interrupt"

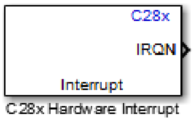
"Modify Duty Cycle of ePWM Using DMA"

C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F28004x ADC

“Photovoltaic Inverter with MPPT Using Solar Explorer Kit”

C28x Hardware Interrupt

Interrupt Service Routine to handle hardware interrupt on C28x processors



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Scheduling

Description

Execution scheduling models based on timer interrupts do not meet the requirements of some real-time applications to respond to external events. The C28x Hardware Interrupt block addresses this problem by allowing asynchronous processing of interrupts triggered by events managed by other blocks in the C280x/C2833x DSP Chip Support Library.

When the C28x Hardware Interrupt block has an external interrupt selection, the selection enables interrupts on the selected general-purpose I/O pins. To configure these pins, see the **Configuration Parameters** > **Hardware Implementation** > **Hardware board settings** > **Target hardware resources** > **External Interrupt** pane. For more information, see “Hardware Implementation Pane: Texas Instruments C2000 Processors” on page 1-2.

Vectorized Output

The output of this block is a function call. The size of the function call line equals the number of interrupts the block is set to handle. Each interrupt is represented by four parameters shown on the dialog box of the block. These parameters are a set of four vectors of equal length. Each interrupt is represented by one element from each parameter (four elements total), one from the same position in each of these vectors.

Each interrupt is described by:

- CPU interrupt numbers
- Peripheral Interrupts Expansion (PIE) interrupt numbers
- Task priorities
- Preemption flags

So, one interrupt is described by a CPU interrupt number, a PIE interrupt number, a task priority, and a preemption flag.

The CPU and PIE interrupt numbers together uniquely specify a single interrupt for a single peripheral or peripheral module.

The following table lists the PIE and CPU interrupt numbers for the c28x processors F280x, F2802x, F2803x, F2805x, F2806x, F2833x, F28M35x, and F28M36x that support 12×8 interrupts. The row headers 1–12 represent the CPU values, and the column headers 1–8 represent the PIE values.

PIE and CPU Interrupt Numbers for F280x, F2802x, F2803x, F2805x, F2806x, F2833x, F28M35x, and F28M36x Processors

PIE ⇒ / CPU ⇓	1	2	3	4	5	6	7	8
1	SEQ1INT (ADC) / ADCINT1	SEQ2INT (ADC) / ADCINT2	Reserved	XINT1	XINT2	ADCINT / ADCINT9	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	EPWM1_TZINT	EPWM2_TZINT	EPWM3_TZINT	EPWM4_TZINT	EPWM5_TZINT	EPWM6_TZINT	EPWM7_TZINT	EPWM8_TZINT
3	EPWM1_INT	EPWM2_INT	EPWM3_INT	EPWM4_INT	EPWM5_INT	EPWM6_INT	EPWM7_INT	EPWM8_INT
4	ECAP1_INT	ECAP2_INT	ECAP3_INT	ECAP4_INT	ECAP5_INT	ECAP6_INT	EPWM10_TZINT / HRCAP1_INT	EPWM9_TZINT / HRCAP2_INT
5	EQEP1_INT	EQEP2_INT	EQEP3_INT	HRCAP3_INT	HRCAP4_INT	Reserved	EPWM10_INT	EPWM9_INT
6	SPIRXINTA (SPI-A)	SPITXINTA (SPI-A)	SPIRXINTB (SPIB_RX) / MRINTB (McBSP-B)	SPITXINTB (SPIB_TX) / MXINTB (McBSP-B)	SPIRXINTC (SPI-C) / MRINTA (McBSP-A_RX)	SPITXINTC (SPI-C) / MXINTA (McBSP-A_TX)	SPIRXINTD (SPI-D) / EPWM12_TZINT	SPITXINTD (SPI-D) / EPWM11_TZINT
7	DINTCH1 (DMA1)	DINTCH2 (DMA2)	DINTCH3 (DMA3)	DINTCH4 (DMA4)	DINTCH5 (DMA5)	DINTCH6 (DMA6)	EPWM12_INT	EPWM11_INT
8	I2CINT1A	I2CINT2A	Reserved	Reserved	SCIRXINTC (SCI-C)	SCITXINTC (SCI-C)	Reserved	Reserved
9	SCIRXINTA (SCIA_RX)	SCITXINTA (SCIA_TX)	SCIRXINTB (SCIB_RX) / LINA_INT0	SCITXINTB (SCIB_TX) / LINA_INT1	ECAN0INTA (CANA_1)	ECAN1INTA (CANA_2)	ECAN0INTB (CANB_1)	ECAN1INTB (CANB_2)

PIE ⇒ / **CPU ⇓**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
10	EPWM9_TZINT / ADCINT1	EPWM10_TZINT / ADCINT2	EPWM11_TZINT / ADCINT3	EPWM12_TZINT / ADCINT4	EPWM13_TZINT / ADCINT5	EPWM14_TZINT / ADCINT6	EPWM15_TZINT / ADCINT7	EPWM16_TZINT / ADCINT8
11	CLA1_INT1 / EPWM9_INT7 / MTOCIPCINT1	CLA1_INT2 / EPWM10_INT / MTOCIPCINT2	CLA1_INT3 / EPWM11_INT / MTOCIPCINT3	CLA1_INT4 / EPWM12_INT / MTOCIPCINT4 /	CLA1_INT5 / EPWM13_INT	CLA1_INT6 / EPWM14_INT	CLA1_INT7 / EPWM15_INT	CLA1_INT8 / EPWM16_INT
12	XINT3	XINT4 / C28FLSINGERR	XINT5	XINT6 / C28RAMSINGERR	XINT7 / C28RAMACCVIOL	Reserved	LVF	LUF

The PIE and CPU interrupt numbers for the c28x processors F2807x, F2837xS, F2837xD, F2838x, and F28004x that support 12×16 interrupts are:

PIE and CPU Interrupt Numbers for F2807x, F2837xS, F2837xD, F2838x, and F28004x Processors

PIE ⇒ / **CPU ⇓**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	ADCA1	ADCB1	ADCC1	XINT1	XINT2	ADCD1	TIMER 0	WAKE / WDOG
2	EPWM1_TZ	EPWM2_TZ	EPWM3_TZ	EPWM4_TZ	EPWM5_TZ	EPWM6_TZ	EPWM7_TZ	EPWM8_TZ
3	EPWM1	EPWM2	EPWM3	EPWM4	EPWM5	EPWM6	EPWM7	EPWM8
4	ECAP1	ECAP2	ECAP3	ECAP4	ECAP5	ECAP6	ECAP7	Reserved
5	EQEP1	EQEP2	EQEP3	Reserved	CLB1	CLB2	CLB3	CLB4
6	SPIA_RX	SPIA_TX	SPIB_RX	SPIB_TX	MCBSPA_RX	MCBSPA_TX	MCBSPB_RX	MCBSPB_TX
7	DMA_CH1	DMA_CH2	DMA_CH3	DMA_CH4	DMA_CH5	DMA_CH6	Reserved	Reserved
8	I2CA	I2CA_FIFO	I2CB	I2CB_FIFO	SCIC_RX	SCIC_TX	SCID_RX	SCID_TX
9	SCIA_RX	SCIA_TX	SCIB_RX	SCIB_TX	CANA_0	CANA_1	CANB_0	CANB_1
10	ADCA_EVT	ADCA2	ADCA3	ADCA4	ADCB_EVT	ADCB2	ADCB3	ADCB4
11	CLA1_1	CLA1_2	CLA1_3	CLA1_4	CLA1_5	CLA1_6	CLA1_7	CLA1_8
12	XINT3	XINT4	XINT5	MPOST	FMC.DONE	VCU	FPU_OVERFLOW	FPU_UNDERFLOW

PIE ⇒ / **CPU ⇓**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	I2CA	SYS_ERR	ECATSYNC0 (CPU1 only)	ECATINTn (CPU1 only)	IPC0/ CIPC0	IPC1/ CIPC1	IPC2/ CIPC2	IPC3/ CIPC3
2	EPWM9_TZ	EPWM10_TZ	EPWM11_TZ	EPWM12_TZ	EPWM13_TZ	EPWM14_TZ	EPWM15_TZ	EPWM16_TZ

PIE ⇒ **CPU ⇓**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
3	EPWM9	EPWM10	EPWM11	EPWM12	EPWM13	EPWM14	EPWM15	EPWM16
4	FSITXA_INT1	FSITXA_INT2	FSITXB_INT1	FSITXB_INT2	FSIRXA_INT1	FSIRXA_INT2	FSIRXB_INT1	FSIRXB_INT2
5	SD1 / SDFM1	SD2/ SDFM1	ECATRSTINTn (CPU1 only)	ECATSYNC1 (CPU1 only)	SDFM1DR1	SDFM1DR2	SDFM1DR3	SDFM1DR4
6	SPIC_RX	SPIC_TX	SPID_RX	SPID_TX	SDFM2DR1	SDFM2DR2	SDFM2DR3	SDFM2DR4
7	FSIRXC_INT1	FSIRXC_INT2	FSIRXD_INT1	FSIRXD_INT2	FSIRXE_INT1	FSIRXE_INT2	FSIRXF_INT1	FSIRXF_INT2
8	LINA_0/ FSIRXG_INT1	LINA_1/ FSIRXG_INT2	FSIRXH_INT1	FSIRXH_INT2	PMBUSA/ CLB5	CLB6	UPPA (CPU1 only)/CLB7	CLB8
9	MCANSS_INT0(CPU1 only)	MCANSS_INT1 (CPU1 only)	MCANSS_ECC_CORR_PUL_INT (CPU1 only)	MCANSS_WAKE_AND_TS_PLS_INT (CPU1 only)	PMBUSA	CM_STATUS (CPU1 only)	USBA (CPU1 only)	Reserved
10	ADCC_EVT	ADCC2	ADCC3	ADCC4	ADCD_EVT	ADCD2	ADCD3	ADCD4
11	CMTOCPUxIPCINTR0	CMTOCPUxIPCINTR1	CMTOCPUxIPCINTR2	CMTOCPUxIPCINTR3	CMTOCPUxIPCINTR4	CMTOCPUxIPCINTR5	CMTOCPUxIPCINTR6	CMTOCPUxIPCINTR7
12	EMIF_ ERROR	RAM_CORRECTABLE_ERROR/ ECAP6INT2	FLASH_CORRECTABLE_ERROR/ ECAP7INT2	RAM_ACCESS_VIOLATION	SYS_PLL_SLIP/ CPUxCRC_INT	AUX_PLL_SLIP// CLA1CRC_INT	CLA OVERFLOW	CLA UNDERFLOW

The PIE and CPU interrupt numbers for the c281x processors are:

PIE and CPU Interrupt Numbers for C281x Processors

PIE ⇒	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
CPU ⇓								
1	PDPINTA (EV-A)	PDPINTB (EV-B)	Reserved	XINT1	XINT2	ADCINT (ADC)	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	CMP1INT (EV-A)	CMP2INT (EV-A)	CMP3INT (EV-A)	T1PINT (EV-A)	T1CINT (EV-A)	T1UFINT (EV-A)	T1OFINT (EV-A)	Reserved
3	T2PINT (EV-A)	T2CINT (EV-A)	T2UFINT (EV-A)	T2OFINT (EV-A)	CAPINT1 (EV-A)	CAPINT2 (EV-A)	CAPINT3 (EV-A)	Reserved
4	CMP4INT (EV-B)	CMP5INT (EV-B)	CMP6INT (EV-B)	T3PINT (EV-B)	T3CINT (EV-B)	T3UFINT (EV-B)	T3OFINT (EV-B)	Reserved
5	T4PINT (EV-B)	T4CINT (EV-B)	T4UFINT (EV-B)	T4OFINT (EV-B)	CAPINT4 (EV-B)	CAPINT5 (EV-B)	CAPINT6 (EV-B)	Reserved
6	SPIRXINTA (SPI)	SPITXINTA (SPI)	Reserved	Reserved	MRINT (McBSP)	MXINT (McBSP)	Reserved	Reserved
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	SCIRXINTA (SCI-A)	SCITXINTA (SCI-A)	SCIRXINTB (SCI-B)	SCITXINTB (SCI-B)	ECAN0INT (CAN)	ECAN1INT (CAN)	Reserved	Reserved
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

The task priority indicates the relative importance tasks associated with the asynchronous interrupts. If an interrupt triggers a higher-priority task while a lower-priority task is running, the execution of the lower-priority task is suspended while the higher-priority task is executed. The lowest value represents the highest priority. The default priority value of the base rate task is 40, so the priority value for each asynchronously triggered task must be less than 40 for these tasks to suspend the base rate task.

The preemption flag determines whether a given interrupt is pre-emptable. Preemption overrides prioritization, such that a preemptable task of higher priority can be preempted by a non-preemptable task of lower priority.

Parameters

CPU interrupt numbers

Enter a vector of CPU interrupt numbers for the interrupts you want to process asynchronously.

PIE interrupt numbers

Enter a vector of PIE interrupt numbers for the interrupts you want to process asynchronously.

Simulink task priorities

Enter a vector of task priorities for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 2-40 for an explanation of task priorities.

Preemption flags

Enter a vector of preemption flags for the interrupts you want to process asynchronously.

See the discussion of this block's "Vectorized Output" on page 2-40 for an explanation of preemption flags.

Enable simulation input

Select this check box if you want to be able to test asynchronous interrupt processing in the context of your Simulink software model.

Note Select this check box to enable you to test asynchronous interrupt processing behavior in Simulink software.

See Also

"ADC-PWM Synchronization Using ADC Interrupt"

"Asynchronous Scheduling"

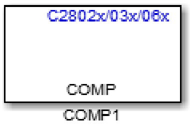
“Permanent Magnet Synchronous Motor Field-Oriented Control”

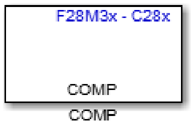
“Schedule a Multi-Rate Controller for a Permanent Magnet Synchronous Machine”

“Photovoltaic Inverter with MPPT Using Solar Explorer Kit”

C2802x/C2803x/C2806x/F28M3x COMP

Compare two input voltages on comparator pins





Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Configure the comparator module to output the comparison result on the comparator pins of the processor.

Parameters

Comparator module

Select the comparator module to configure. Use only one block per module.

Second input

Select `COMPxB` to compare the voltage of **Input Pin A** with **Input Pin B**

Select `Internal DAC` to compare the voltage of **Input Pin A** with the output of a DAC reference located in the comparator. For more information, see the "DAC Reference" section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator*.

The comparator source outputs 1, if **Input Pin A** has a value greater than **Input Pin B** or the 10-bit DAC reference. Otherwise, it outputs 0.

Inverter comparator output

Select this check box to apply a logical NOT to the output of the comparator source. For example, when the comparator source outputs 1, the inverter circuit changes it to 0.

Synchronization

Select `Asynchronous` to pass the asynchronous version of the comparator output. Select `Synchronous` to pass the synchronous version of the comparator output. Selecting `Synchronous` enables the **Qualification period** option.

Qualification period

Qualify changes in the comparator output before passing them along. The `Passed through` setting passes changes in the comparator value along without qualifying them. The `consecutive clocks` settings pass changes in the comparator value along after receiving the specified number of consecutive samples with the same value. Use this setting to prevent intermittent and spurious changes in the comparator output.

Sample time

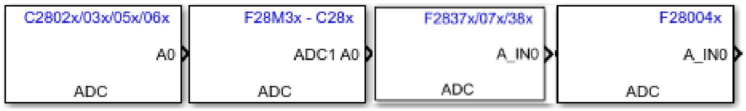
Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ADC

Configure ADC to sample analog pins and output digital data



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2805x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2807x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xD

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xS

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2838x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F28004x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Configures the ADC to output a constant stream of data collected from the ADC pins on the DSP.

An ADC block allows for reading one ADC channel. Use multiple ADC blocks to read multiple ADC channels.

Parameters

ADC Module

Select ADC Module 1 or ADC Module 2 for conversion.

Select ADC Module A through D for the processors that support Type 4 ADC.

Note The **ADC Module** parameter is available only for Texas Instruments C2000 processors that support Type 3, Type 4, or Type 5 ADC.

ADC Resolution

Select 12-bit (Single-ended input) or 16-bit (Differential inputs) ADC resolution options.

In 12-bit mode, only single-ended input is supported. In 16-bit mode, the input voltage to the converter is sampled through a pair of input pins, that means the differential inputs between the two channels is converted.

This parameter appears only for Texas Instruments C2000 F2807x, Texas Instruments C2000 F2837xD, Texas Instruments C2000 F2838x and Texas Instruments C2000 F2837xS processors.

Note The 16-bit (Differential inputs) ADC mode is not enabled by default in most of the processors.

Sampling mode

Select **Single sample mode** to sample signals sequentially. Select **Simultaneous sample mode** to sample pairs of signals. The hardware allows each signal of a pair to be sampled at the same time.

SOC trigger number

Identify the start-of-conversion trigger by number. In single sampling mode, you can select an individual trigger. In simultaneous sampling mode, you can select triggers in pairs.

SOCx acquisition window

Define the length of the acquisition period in ADC clock cycles. The value of this parameter depends on the SYSCLK and the minimum ADC sample time. For more information, see the ADC Acquisition (Sample and Hold) Window section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide*.

SOCx trigger source

Select the source that triggers the start of conversion. The following types of inputs are available:

- Software
- CPU Timers 0/1/2 interrupts
- XINT2 SOC
- ePWM1-9 SOCA and SOCB

If you set **SOCx trigger source** to `XINT2_XINT2SOC`, use the **XINT2SOC external pin / ADCEXTSOC external pin** parameter at **Hardware Implementation > Target hardware resources** to define the external GPIO pin that triggers the start of conversion.

ADCINT will trigger SOCx

At the end of conversion, use the `ADCINT1` or `ADCINT2` interrupt to trigger a start of conversion (SOC). This loop creates a continuous sequence of conversions. The default selection, `No ADCINT` disables this parameter. To set the interrupt, select the **Post interrupt at EOC trigger** option, and choose the appropriate interrupt.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

Data type

Select the data type of the digital output data. You can choose from the options `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

Post interrupt at EOC trigger

Post interrupts when the ADC triggers EOC pulses. When you select this option, the dialog box displays the **Interrupt selection** and **ADCINT# continuous mode** options. For more information, see the EOC and Interrupt Operation section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide*.

Interrupt selection

Select which interrupt the ADC posts after triggering an EOC pulse.

ADCINT1 continuous modeADCINT2 continuous mode

When the ADC generates an end of conversion (EOC) signal, generate an ADCINT# interrupt, whether the previous interrupt flag has been acknowledged or not.

Input Channels – Conversion channel

Select the input channel to which this ADC conversion applies. For Type 4 ADC, if you select 16-bit (differential inputs) mode, the differential voltage between the two channels is converted.

References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ePWM

“ADC-PWM Synchronization Using ADC Interrupt”

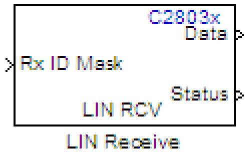
C28x Hardware Interrupt

“Configuring Acquisition Window Width for ADC Blocks”

“Photovoltaic Inverter with MPPT Using Solar Explorer Kit”

C2803x LIN Receive

Receive data via local interconnect network (LIN) module on target



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Description

The Local Interconnect Network (LIN) bus implements a serial communications protocol for distributed automotive and industrial applications. In particular, LIN serves low cost applications that do not require the bandwidth or robustness provided by the CAN protocol.

The LIN Receive block configures the target to receive scalar or vector data from the LINRX or LINTX pins.

Each C2803x target has one LIN module. Your model can only contain one LIN Transmit and one LIN Receive block per module.

The C2803x LIN Transmit block takes three inputs:

- **ID**: Set the value of the LIN ID for the LIN transmit node.
- **Tx ID Mask**: Set the value of the LIN ID mask for the LIN transmit node.
- **Data**: Connect this input to the data source.

For more information and examples, see:

- “Configuring LIN Communications”

- “LIN-Based Control of PWM Duty Cycle”

Note Many LIN-specific settings are located under **Peripherals** > **LIN** in Hardware Implementation -> Target Hardware Resources for your model. Verify that these settings meet the requirements of your application.

Parameters

Data type

Select the data type the LIN block outputs to the model. Available options are `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. To interpret the data, the data type and data length must match those of the data input to transmitting LIN node.

The default value is `int16`.

Data length

Set the length of the data the LIN block outputs to the model. This value is measured in multiples of the **Data type**. For example, if **Data type** is `int16` and **Data length** is `int16`, the LIN block outputs the data to the model in lengths of

1 x int16

If you set the **Data length** to a value greater than 1, the block outputs the data as vectors.

To interpret the data, the data type and data length must match those of the data input to transmitting LIN node.

The default value is 1.

Note In a loopback configuration, the maximum data length cannot exceed 8 bytes. If the sum of the incoming and the outgoing data exceeds the hardware buffer length of the LIN module, the module discards incoming bytes of data.

Initial output

Set the initial value the DATA port outputs to the model before the LIN node has received data.

The default value is `0`.

Action taken when connection times out

Specify what the LIN block outputs on the DATA port in response to a connection time-out. The choices are:

- `Output the last received value` — the DATA port outputs the last data value the LIN node received.
- `Output custom value` — the DATA port outputs the value defined by **Output value when connection times out**.

The default value is `Output the last received value`.

If the LIN node has not received data, and you set this parameter to `Output the last received value`, the DATA port outputs the **Initial output** value.

Output value when connection times out

Specify the custom value the DATA port outputs when **Action taken when connection times out** is set to `Output custom value` and a connection timeout occurs.

Enable blocking mode

If you enable (select) this checkbox, the target application stops and waits for the LIN node to receive data before continuing. If you disable this option, the application continues running and does not wait for data to arrive.

The default value is disabled (deselected).

Verify checksum

If you enable (select) this option, the LIN node verifies the checksum it receives.

The default value is disabled (deselected).

Output receiving status

Enabling (selecting) this checkbox adds a `status` output to the LIN Receive block, as shown in the following figure.

The `status` output reports the following values for each message the LIN node receives:

- `0`: No error.
- `-1`: A time-out occurred while the block was waiting to receive data.

- `-2`: Unable to receive.
- Other status values represent the highest 8 bits of the SCI Flags Register. Convert these values from decimal to binary. Then determine the meaning of these values by referring to Table 14. SCI Flags Register (SCIFLR) Field Descriptions in *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

Receive buffer interrupt

If you enable this option, the SCI node generates an interrupt after it receives a complete frame. The default value is `Disabled`.

Checksum error interrupt

If you enable this option, the LIN block generates an interrupt when the incoming message contains an invalid checksum.

The default value is `Disabled`.

The TXRX Error Detector Checksum Calculator verifies checksums for incoming messages. With the classic LIN implementation, the checksum only covers the data fields. For LIN 2.0–compliant messages, the checksum includes both the ID field and the data fields. If you enable this option, the Checksum Calculator generates interrupts when it detects checksum errors, such as those caused by LIN message collisions.

Framing error interrupt

If you enable this option, the LIN module generates interrupts when framing errors occur.

The default value is `Disabled`.

Overrun error interrupt

If you enable this option, the LIN module generates interrupt when overrun errors occur.

The default value is `Disabled`.

ID parity error interrupt

If you enable this option, the LIN module generates an ID-Parity interrupt when it receives an invalid ID.

The default value is `Disabled`.

If you enable this option, also enable **Parity mode** in Hardware Implementation -> Target Hardware Resources.

ID match interrupt

If you enable this option, the LIN module generates an interrupt when the LIN node validates the ID in messages it receives.

The default value is `Disabled`.

Sample time

Set the block's input sample time, T_s.

The default value is `0.1` seconds.

References

For detailed information on the LIN module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

See Also

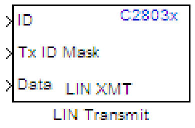
C2803x LIN Transmit (block reference)

“Configuring LIN Communications”

“LIN-Based Control of PWM Duty Cycle”

C2803x LIN Transmit

Transmit data from target via serial communications interface (SCI) to host



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Description

The Local Interconnect Network (LIN) bus implements a serial communications protocol for distributed automotive and industrial applications. In particular, LIN serves low cost applications that do not require the bandwidth or robustness provided by the CAN protocol.

The C2803x LIN Transmit block takes three inputs:

- **ID**: Set the value of the LIN ID for the LIN transmit node.
- **Tx ID Mask**: Set the value of the LIN ID mask for the LIN transmit node.
- **Data**: Connect this input to the data source.

Note Many LIN-specific settings are located under **Peripherals > LIN** in **Hardware Implementation > Target hardware resources** for your model. Verify that these settings meet the requirements of your application.

Parameters

Send checksum

Select this checkbox to include a checksum in the last data field of the checkbyte. LIN 2.0 implementations require this checksum.

The default value is unchecked (disabled).

Physical bus error interrupt

The LIN master node detects when the physical bus cannot convey a valid message. For example, if the bus had a short circuit to ground or to V_{BAT}. This raises a physical bus error flag in all of the LIN nodes on the network. If you enable **Physical bus error interrupt**, the LIN transmit node generates an interrupt in response to a physical bus error flag.

Bit error interrupt

If you enable this option, the LIN node compares the data it transmits and the data on the LIN bus.

The default value is `Disabled`.

The TXRX Error Detector Bit Monitor compares data bits on the LIN transmit (LINTX) and receive (LINRX) pins. If the data do not match, the Bit Monitor raises a bit-error flag. When you enable this option, the bit-error flag also produces a bit-error interrupt.

Transmit buffer interrupt

If you enable this option, the LIN node generates an interrupt while it is generating a checksum and setting the Transmitter buffer register ready flag.

The default value is `Disabled`.

References

For detailed information on the SCI module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

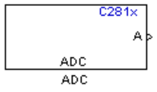
See Also

“Configuring LIN Communications”

“LIN-Based Control of PWM Duty Cycle”

C281x ADC

Analog-to-digital converter (ADC)



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Description

The C281x ADC block configures the C281x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices.

Triggering

The C281x ADC trigger mode depends on the internal setting of the source start-of-conversion (SOC) signal. In unsynchronized mode the ADC is usually triggered by software at the sample time intervals specified in the ADC block. For more information on configuring the specific parameters for this mode, see “Configuring Acquisition Window Width for ADC Blocks”.

In synchronized mode, the Event (EV) Manager associated with the same module as the ADC triggers the ADC. In this case, the ADC is synchronized with the pulse width modulator (PWM) waveforms generated by the same EV unit via the **ADC Start Event** signal setting. The **ADC Start Event** is set in the C281x PWM block. See that block for information on the settings.

Note The ADC cannot be synchronized with the PWM if the ADC is in cascaded mode (see below).

Output

The output of the C281x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

Modes

The C281x ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module `A` or module `B` can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module `A` and module `B` are used for a single ADC block.

Parameters

ADC Control Pane

Module

Specify which DSP module to use:

- `A` — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- `B` — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- `A and B` — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

Then, use the check boxes to select the desired ADC channels.

Conversion mode

Type of sampling to use for the signals:

- `Sequential` — Samples the selected channels sequentially
- `Simultaneous` — Samples the corresponding channels of modules A and B at the same time

Start of conversion

Specify the type of signal that triggers the conversion:

- `Software` — Signal from software

- `EVA` — Signal from Event Manager A (only for Module A)
- `EVB` — Signal from Event Manager B (only for Module B)
- `External` — Signal from external hardware

Sample time

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. To execute this block asynchronously, set **Sample Time** to `-1`, check the **Post interrupt at the end of conversion** box.

To set different sample times for different groups of ADC channels, you must add separate C281x ADC blocks to your model and set the desired sample times for each block.

Data type

Date type of the output data. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`.

Post interrupt at the end of conversion

Check this check box to post an asynchronous interrupt at the end of each conversion. The interrupt is posted at the end of conversion.

Input Channels Pane

Number of conversions

Number of ADC channels to use for analog-to-digital conversions.

Conversion no.

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence. To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

Use multiple output ports

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

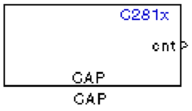
See Also

C281x PWM

C28x Hardware Interrupt

C281x CAP

Receive and log capture input pin transitions



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Description

The C281x CAP module provides input capture functionality for systems where precise timing of external events is important. The C281x CAP block sets parameters for the capture units (CAPs) of the Event Manager (EV) module. The capture units log transitions detected on the capture unit pins by recording the times of the input signal transitions into a two-level deep FIFO stack. You can set the capture unit pins to detect rising edge, falling edge, either type of transition, or no transition. The cnt output of the block gives the captured value of the EV running timer.

The C281x chip has six capture units — three associated with each EV module. Capture units 1, 2, and 3 are associated with EVA and capture units 4, 5, and 6 are associated with EVB. Each capture unit is associated with a capture input pin.

Each group of EV module capture units can use one of two general-purpose (GP) timers on the target board. EVA capture units can use GP timer 1 or 2. EVB capture units can use GP timer 3 or 4. When a transition occurs, the module stores the value of the selected timer in the two-level deep FIFO stack.

The C281x CAP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Note You can have up to two C281x CAP blocks in a model—one block for each EV module.

Outputs

This block has up to two outputs: a `cnt` (count) output and an optional, FIFO status `flag` output. The `cnt` output holds the value of the EV timer captured during the detected transitions. The `cnt` output gives the captured values of the running counter based on the value set in **Output data format** parameter. The status flag outputs are:

- `0` — The FIFO is empty. Either no captures have occurred or the previously stored captures have been read from the stack. (The binary version of this flag is `00`.)
- `1` — The FIFO has one entry in the top register of the stack. (The binary version of this flag is `01`.)
- `2` — The FIFO has two entries in the stack registers. (The binary version of this flag is `10`.)
- `3` — The FIFO has two entries in the stack registers and one or more captured values have been lost. This occurs because another capture occurred before the FIFO stack was read. This means that the FIFO stack is read when you execute the block as specified by your scheduling scheme synchronously, if a sample time is used or asynchronously, if triggered by an interrupt or an idle task. The new value is placed in the bottom register. The bottom register value is pushed to the top of the stack and the top value is pushed out of the stack. (The binary version of this flag is `11`.)

Parameters

Data Format Pane

Module

Select the Event Manager (EV) module to use:

- `A` — Use CAPs 1, 2, and 3.
- `B` — Use CAPs 4, 5, and 6.

Output overrun status flag

Select to output the status of the elements in the FIFO. The data type of the status flag is uint16.

Output data format

The type of data to output:

- `Send 2 elements (FIFO Buffer)` — Sends the latest two values. The output is updated when there are two elements in the FIFO, which is indicated by bit 13 or 11 or 9 being sent (CAP x FIFO). If the CAP is polled when fewer than two elements are captures, old values are repeated. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. The top value of the FIFO is read and stored in the output at index 0.
  3. The new top value of the FIFO (the previously stored bottom stack value) is read and stored in the output at index 1.

- `Send 1 element (oldest)` — Sends the older of the two most recent values. The output is updated when there is at least one element in the FIFO, which is indicated by the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. The top value of the FIFO is read and stored in the output.

- `Send 1 element (latest)` — Sends the most recent value. The output is updated when there is at least one element in the FIFO, which is indicated by the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. If the FIFO buffer contains two entries, the bottom value is read and stored in the output. If the FIFO buffer contains one entry, the top value is read and stored in the output.

Sample time

Time between outputs from the FIFO. If new data is not available, the previous data is sent.

Data type

Data type of the output data. Available options are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean`. The `auto` option uses the data type of a connected block that outputs data to this block. If this block does not receive an input, `auto` sets the data type to `double`.

Note The output of the C281x CAP block can be vectorized.

CAP Panes

The CAP panes set parameters for individual CAPs. The particular CAP affected by a CAP pane depends on the EV module you selected:

- **CAP1** controls CAP 1 or CAP 4, for EV module A or B, respectively.
- **CAP2** controls CAP 2 or CAP 5, for EV module A or B, respectively.
- **CAP3** controls CAP 3 or CAP 6, for EV module A or B, respectively.

Enable CAP

Select to use the specified capture unit pin.

Edge Detection

Type of transition detection to use for this CAP. Available types are `Rising Edge`, `Falling Edge`, `Both Edges`, and `No transition`.

Time Base

Select which target board GP timer the CAP uses as a time base. CAPs 1, 2, and 3 can use `Timer 1` or `Timer 2`. CAPs 4, 5, and 6 can use `Timer 3` or `Timer 4`.

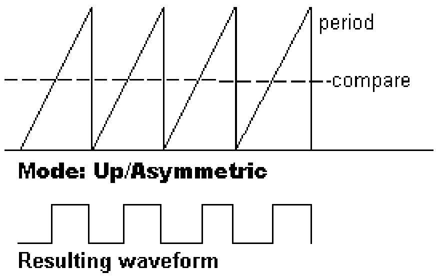
Clock source

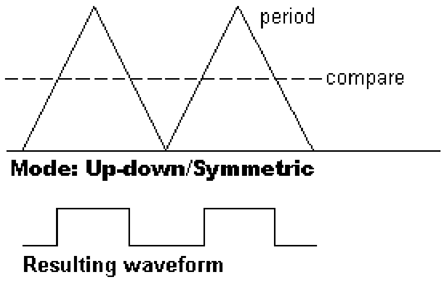
This option is available only for the CAP 3 pane. You can select `Internal` to use the internal time base. Also configure the **Counting mode**, **Timer prescaler**, and **Timer period source** for the internal time base.

Select `QEP circuit` to generate the input clock from the quadrature encoder pulse (QEP) submodule.

Counting mode

Select `Up` to generate an asymmetrical waveform output, or `Up-down` to generate a symmetrical waveform output, as shown in the following illustration.





The Counting mode is for the internal timer settings.

When you specify the **Counting mode** as `Up` (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as `Up-down` (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**
- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when you set **Clock source** to `QEP circuit`.

Timer Prescaler

Clock divider factor by which to prescale the selected GP timer to produce the desired timer counting rate. Available options are `none`, `1/2`, `1/4`, `1/8`, `1/16`, `1/32`, `1/64`, and `1/128`. The following table shows the rates that result from selecting each option.

Scaling	Resulting Rate (µs)
none	0.01334
1/2	0.02668
1/4	0.05336
1/8	0.10672
1/16	0.21344
1/32	0.42688
1/64	0.85376
1/128	1.70752

Note These rates assume a 75 MHz input clock.

Timer period source

Select `Specify via dialog` to enable the **Timer period** parameter. Select `Input port` to create a block input, **T1**, that accepts the timer period value.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `65535`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClock\mathrm{P}rescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 µs.

Post interrupt on CAP

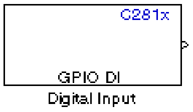
Check this check box to post an asynchronous interrupt on CAP.

See Also

C28x Hardware Interrupt

C281x GPIO Digital Input

General-purpose I/O pins for digital input



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F28004x

Description

This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital input. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

Note To avoid losing new settings, click **Apply** before changing the **IO Port** parameter.

Parameters

IO Port

Select the input/output port to use: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (There is no GPIOPC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DI blocks cannot share the same I/O port.

Note The input function of the digital I/O and the input path to the related peripheral are enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

The following tables show the shared pins.

GPIO A MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

GPIO B MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

Sample time

Time interval, in seconds, between consecutive input from the pins.

Data type

Data type of the data to obtain from the GPIO pins. The data is read as 16-bit integer data and then cast to the selected data type. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `boolean`.

Note The width of the vectorized data output by this block is determined by the number of bits selected in the **Block Parameters** dialog box.

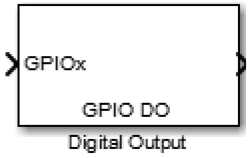
See Also

C281x GPIO Digital Output

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x GPIO Digital Input C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x GPIO Digital Output

C281x GPIO Digital Output

General-purpose I/O pins for digital output



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Embedded Coder Support Package for Texas Instruments C2000 Processors/F28004x

Description

This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital output. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

Note Fixed-point inputs are not supported for this block.

Note To avoid losing new settings, click **Apply** before changing the **IO Port** parameter.

Parameters

IO Port

Select the input/output port to use: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (There is no GPIOPC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DO blocks cannot share the same I/O port.

Note The input function of the digital I/O and the input path to the related peripheral are enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

The following tables show the shared pins.

GPIO A MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

GPIO B MUX

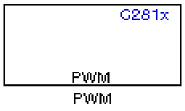
Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

See Also

C281x GPIO Digital Input

C281x PWM

Pulse width modulators (PWMs)



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Description

F2812 DSPs include a suite of pulse width modulators (PWMs) used to generate various signals. This block provides options to set the A or B module Event Managers to generate the waveforms you require. The twelve PWMs are configured in six pairs, with three pairs in each module.

The C281x PWM module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Note All inputs to the C281x PWM block must be scalar values.

Parameters

Timer Pane

Module

Specify which target PWM pairs to use:

- A — Displays the PWMs in module A (PWM1/PWM2, PWM3/PWM4, and PWM5/PWM6).

- `B` — Displays the PWMs in module B (PWM7/PWM8, PWM9/PWM10, and PWM11/PWM12).

Note PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3.

Waveform period source

Source from which the waveform period value is obtained. Select `Specify via dialog` to enter the value in **Waveform period** or select `Input port` to use a value from the input port.

Note All inputs to the C281x PWM block must be scalar values.

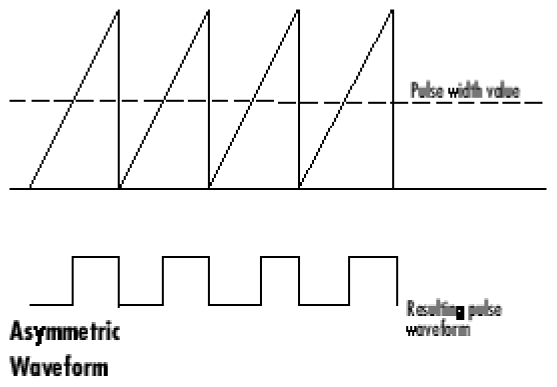
Waveform period

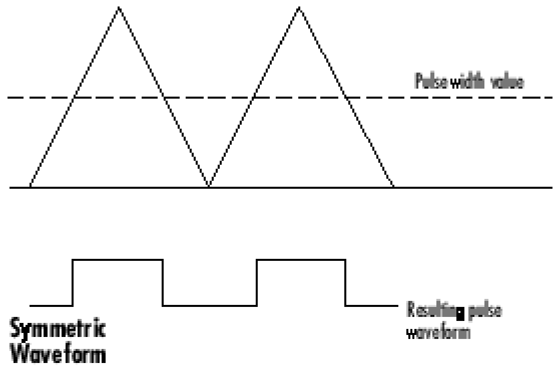
Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

Note The term *clock cycles* refers to the high-speed peripheral clock on the F2812 chip. This clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

Waveform type (counting mode)

Type of waveform to be generated by the PWM pair. The F2812 PWMs can generate two types of waveforms: `Asymmetric(Up)` and `Symmetric(Up-down)`. The following illustration shows the difference between the two types of waveforms.





Waveform period units

Units in which to measure the waveform period. Options are `Clock cycles`, which refer to the high-speed peripheral clock on the F2812 chip (75 MHz), or `Seconds`. Changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

Outputs Pane

Enable PWM#/PWM#

Check to activate the PWM pair. PWM1/PWM2 are activated via the Output 1 pane, PWM3/PWM4 are on Output 2, and PWM5/PWM6 are on Output 3.

Duty cycle source

Select `Specify via dialog` to use the dialog box to enter a **Duty cycle** value for the pair of PWM outputs. Select `Input port` to use the input port, **W#**, to enter a **Duty cycle** value for the pair of PWM outputs.

The input port **W1** corresponds to PWM1/PWM2. **W2** corresponds to PWM3/PWM4. **W3** corresponds to PWM5/6.

Note All inputs to the C281x PWM block must be scalar values.

Duty cycle

Set the ratio of the PWM waveform pulse duration to the PWM **Waveform period**.

Duty cycle units

Units for the duty cycle. Valid choices are `Clock cycles` and `Percentages`. Changing these units changes the **Duty cycle** value, and the **Waveform period** value and **Waveform period units** selection.

Note Using percentages can cause some additional computation time in generated code. This may or may not be noticeable in your application.

Logic Pane

Control logic source

Configure the control logic for all PWMs enabled on the Outputs tab. Valid settings are `Specify via dialog` (default setting) or to `Input port`.

`Specify via Dialog` enables **PWM control logic** settings for each PWM output:

- `Forced high` causes the pulse value to be high.

  `Active high` causes the pulse value to go from low to high.

`Active low` causes the pulse value to go from high to low.

`Forced low` causes the pulse value to be low.

`Input port` adds an input port to the PWM block for setting the C2000 ACTRx register. Each PWM uses 2 bits to set the following options:

- 00 Forced Low
- 01 Active Low
- 10 Active High
- 11 Forced High

Bits 11–0 of the 16–bit Compare Action Control Registers for module A control PWM1-6

Bits 11–0 of the 16–bit Compare Action Control Registers for module B control PWM1-6

For example: If a decimal value of 3222 is read at the input port while using PWM module A, the following PWM settings will be honored:

3222 = 0C96h = 110010010110b

So that:

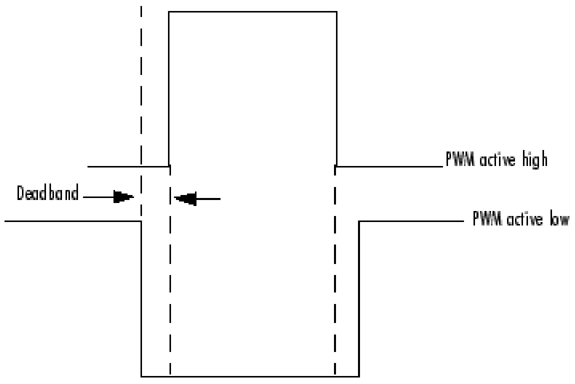
- PW1: Active High
- PW2: Active Low
- PW3: Active Low
- PW4: Active High
- PW5: Forced Low
- PW6: Forced High

For more information, see the section on Compare Action Control Registers (ACTRA and ACTRB) in the Texas Instruments™ document “TMS320x281x DSP Event Manager (EV) Reference Guide”, literature number SPRU065.

Deadband Pane

Use deadband for PWM#/PWM#

Enables a deadband area without signal overlap at the beginning of particular PWM pair signals. The following figure shows the deadband area.



Deadband prescaler

Number of clock cycles, which, when multiplied by the Deadband period, determines the size of the deadband. Selectable values are 1, 2, 4, 8, 16, and 32.

Deadband period source

Source from which the deadband period is obtained. Select `Specify via dialog` to enter the values in the **Deadband period** field or select `Input port` to use a value, in clock cycles, from the input port.

Note All inputs to the C281x PWM block must be scalar values.

Deadband period

Value that, when multiplied by the Deadband prescaler, determines the size of the deadband. Selectable values are from 1 to 15.

ADC Control Pane

ADC start event

Controls whether this PWM and ADC associated with the same EV module are synchronized. Select `None` to disable synchronization or select an event to generate the source start-of-conversion (SOC) signal for the associated ADC.

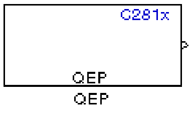
- `None` — The ADC and PWM are not synchronized. The EV does not generate an SOC signal and the ADC is triggered by software (that is, the A/D conversion occurs when the ADC block is executed in the software).
- `Underflow interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the board's general-purpose (GP) timer counter reaches a hexadecimal value of FFFF.
- `Period interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the value in GP timer matches the value in the period register. The value set in **Waveform period** above determines the value in the register.

Note If you select `Period interrupt` and specify a sampling time less than the specified **(Waveform period)/(Event timer clock speed)**, zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the timer is 64000/75 MHz = 8.5333e-004. If you enter a **Sample time** in the C281x ADC dialog box that is less than this result, it will cause zero-order hold interpolation.)

- `Compare interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the value in the GP timer matches the value in the compare register. The value set in **Duty cycle** above determines the value in the register.

C281x QEP

Quadrature encoder pulse circuit



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Description

Each F2812 Event Manager has three capture units, which can log transitions on its capture unit pins. Event Manager A (EVA) uses capture units 1, 2, and 3. Event Manager B (EVB) uses capture units 4, 5, and 6.

The quadrature encoder pulse (QEP) circuit decodes and counts quadrature encoded input pulses on these capture unit pins. QEP pulses are two sequences of pulses with varying frequency and a fixed phase shift of 90 degrees (or one-quarter of a period). The circuit counts both edges of the QEP pulses, so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is useful for obtaining speed and position information from a rotating machine. Logic in the QEP circuit determines the direction of rotation by which sequence is leading. For module A, if the QEP1 sequence leads, the general-purpose (GP) Timer counts up and if the QEP2 sequence leads, the timer counts down. The pulse count and frequency determine the angular position and speed.

The C281x QEP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Parameters

Module

Specify which QEP pins to use:

- `A` — Uses QEP1 and QEP2 pins.
- `B` — Uses QEP3 and QEP4 pins.

Counting mode

Specify how to count the QEP pulses:

- `Counter` — Count the pulses based on GP Timer 2 (or GP Timer 4 for EVB).
- `RPM` — Count the rotations per minute.

Positive rotation

Defines whether to use `Clockwise` or `Counterclockwise` as the direction to use as positive rotation. This field appears only if you select `RPM`.

Initial count

Initial value for the counter. The value defaults to `0`.

Encoder resolution (pulse/revolution)

Number of QEP pulses per revolution. This field appears only if you select `RPM`.

Enable QEP index

Reset the QEP counter to zero when the QEP index input on CAP3_QEPI1 transitions from low to high.

Enable index qualification mode

Qualify the QEP index input on CAP3_QEPI1. Check that the levels on CAP1_QEP1 and CAP2_QEP2 are high before asserting the index signal as valid.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `65535`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClock\mathrm{P}rescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 µs.

Sample time

Time interval, in seconds, between consecutive reads from the QEP pins.

Data type

Data type of the QEP pin data. The circuit reads the data as 16-bit data and then casts it to the selected data type. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `boolean`.

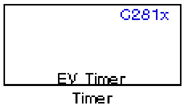
References

For more information on the QEP module, consult the following documents, available at the Texas Instruments Web site:

- *TMS320x280x, 2801x, 2804x Enhanced Quadrature Encoder Pulse (eQEP) Module Reference Guide*, Literature Number SPRU790
- *Using the Enhanced Quadrature Encoder Pulse (eQEP) Module in TMS320x280x, 28xxx as a Dedicated Capture Application Report*, Literature Number SPRAAH1

C281x Timer

Configure general-purpose timer in Event Manager module



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C281x

Description

The C281x contains two event-manager (EV) modules. Each module contains two general-purpose (GP) timers. You can use these timers as independent time bases for various applications.

Use the C281x Timer block to set the periodicity of one GP timer and the conditions under which it posts interrupts. Each model can contain up to four C281x Timer blocks.

The C281x Timer module configures GP Timers that other C281 blocks share. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Parameters

ModuleTimer no

Select which of four possible timers to configure. Setting **Module** to `A` lets you select `Timer 1` or `Timer 2` in **Timer no**. Setting **Module** to `B` lets you select `Timer 3` or `Timer 4` in **Timer no**.

Clock source

When **Timer no** has a value of `Timer 2` or `Timer 4`, use this parameter to select the clock source for the event timer. You can choose either `Internal` or `QEP circuit`.

When you select `Internal`, you can configure other options such as **Timer period source**, **Counting mode**, and **Timer prescaler**.

Timer period source

Select the source of the event timer period. Use `Specify via dialog` to set the period using **Timer period**. Select `Input port` to create an input, **T**, that accepts the value of the timer period in clock cycles, from `0` to `65535`. **Timer period source** becomes unavailable when **Clock source** is set to `QEP circuit`.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `10000`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClock\mathrm{P}rescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 μs.

Compare value source

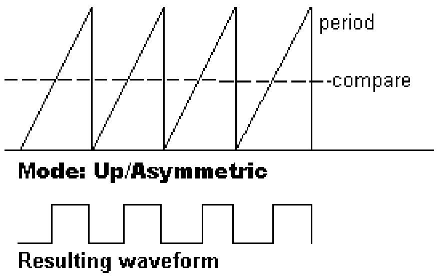
Select the source of the compare value. Use `Specify via dialog` to set the period using the **Compare value** parameter. Select `Input port` to create a block input, **W**, that accepts the value of the compare value, from `0` to `65535`.

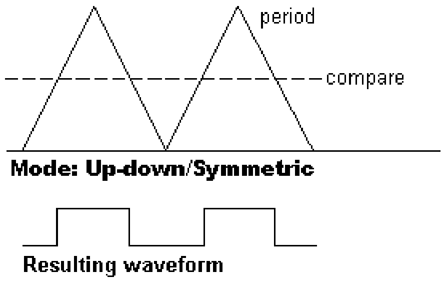
Compare value

Enter a constant value for comparison to the running timer value for generating interrupts. Enter a value from `0` to `65535`. The value defaults to `5000`. The timer only generates interrupts if you enable **Post interrupt on compare match**.

Counting mode

Select `Up` to generate an asymmetrical waveform output, or `Up-down` to generate a symmetrical waveform output, as shown in the following illustration.





When you specify the **Counting mode** as `Up` (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as `Up-down` (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**
- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when **Clock source** is set to `QEP circuit`.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

Timer prescaler becomes unavailable when **Clock source** is set to `QEP circuit`.

Post interrupt on period match

Generate an interrupt when the value of the timer reaches its maximum value as specified in **Timer period**.

Post interrupt on underflow

Generate an interrupt when the value of the timer cycles back to `0`.

Post interrupt on overflow

Generate an interrupt when the value of the timer reaches its maximum, `65535`. Also set **Timer period** to `65535` for this parameter to work.

Post interrupt on compare match

Generate an interrupt when the value of the timer equals **Compare value**.

References

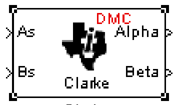
TMS320x281x DSP Event Manager (EV) Reference Guide, Literature Number: SPRU065, available from the Texas Instruments Web site.

See Also

C28x Hardware Interrupt

C2000 Clarke Transformation

Convert balanced three-phase quantities to balanced two-phase quantrature quantities



Clarke
Transformation

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/
Optimization/ C28x DMC

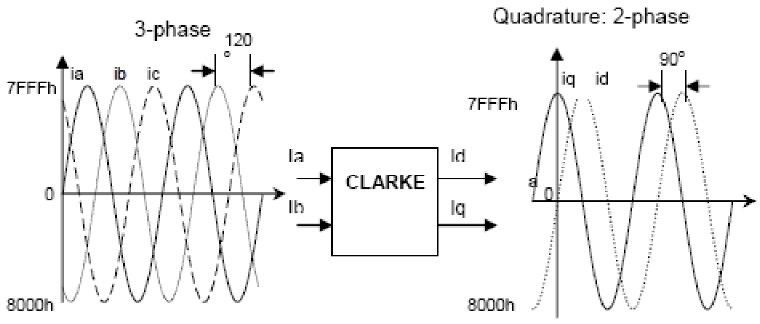
Description

This block converts balanced three-phase quantities into balanced two-phase quadrature quantities. The transformation implements these equations

$$Id = Ia$$

$$Iq = (2Ib + Ia)/\sqrt{3}$$

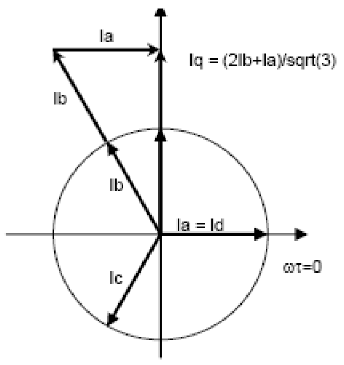
and is illustrated in the following figure.



The inputs to this block are the phase a (`As`) and phase b (`Bs`) components of the balanced three-phase quantities and the outputs are the direct axis (`Alpha`) component and the quadrature axis (`Beta`) of the transformed signal.

The instantaneous outputs are defined by the following equations and are shown in the following figure:

$$ia = I * \sin(\omega t)$$
$$ib = I * \sin(\omega t + 2\pi/3)$$
$$ic = I * \sin(\omega t - 2\pi/3)$$
$$id = I * \sin(\omega t)$$
$$iq = I * \sin(\omega t + \pi/2)$$



The variables used in the preceding equations and figures correspond to the variables on the block as shown in the following table:

	Equation Variables	Block Variables
Inputs	ia	As
	ib	Bs
Outputs	id	Alpha
	iq	Beta

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` CRL.

- The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

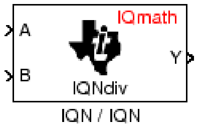
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Inverse Park Transformation, C2000 Park Transformation, C2000 PID Controller, C2000 Space Vector Generator, C2000 Speed Measurement

C2000 Division IQN

Divide IQ numbers



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block divides two numbers that use the same Q format, using the Newton-Raphson technique. The resulting quotient uses the same Q format at the inputs.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

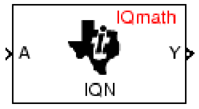
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, c2000 Arctangent IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Float to IQN

Convert floating-point number to IQ number



Float to IQN

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block converts a floating-point number to an IQ number. The Q value of the output is specified in the dialog.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the

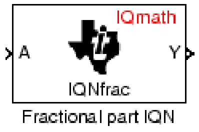
Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Fractional part IQN

Fractional part of IQ number



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block returns the fractional portion of an IQ number. The returned value is an IQ number in the same IQ format.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

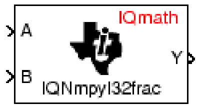
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Fractional part IQN x int32

Fractional part of result of multiplying IQ number and long integer



Fractional part
IQN x int32

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block multiplies an IQ input and a long integer input and returns the fractional portion of the resulting IQ number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

References

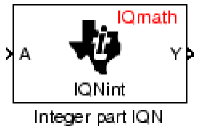
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Integer part IQN

Integer part of IQ number



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block returns the integer portion of an IQ number. The returned value is a long integer.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

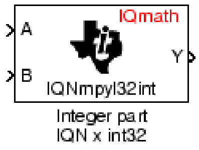
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Integer part IQN x int32

Integer part of result of multiplying IQ number and long integer



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block multiplies an IQ input and a long integer input and returns the integer portion of the resulting IQ number as a long integer.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

References

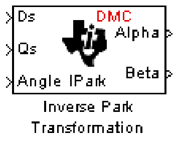
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Inverse Park Transformation

Convert rotating reference frame vectors to two-phase stationary reference frame



Library

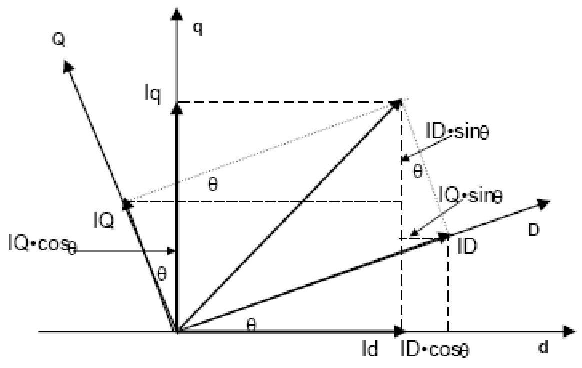
Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block converts vectors in an orthogonal rotating reference frame to a two-phase orthogonal stationary reference frame. The transformation implements these equations:

$$Id = ID * \cos\theta - IQ * \sin\theta$$
$$Iq = ID * \sin\theta + IQ * \cos\theta$$

and is illustrated in the following figure.



The inputs to this block are the direct axis (`Ds`) and quadrature axis (`Qs`) components of the transformed signal in the rotating frame and the phase angle (`Angle`) between the stationary and rotating frames.

The outputs are the direct axis (`Alpha`) and the quadrature axis (`Beta`) components of the transformed signal.

The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

	Equation Variables	**Block Variables**
Inputs	ID	Ds
	IQ	Qs
	θ	Angle
Outputs	id	Alpha
	iq	Beta

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.
- The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

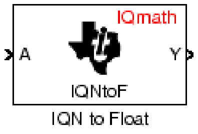
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Clarke Transformation, C2000 Park Transformation, C2000 PID Controller, C2000 Space Vector Generator, C2000 Speed Measurement

C2000 IQN to Float

Convert IQ number to floating-point number



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block converts an IQ input to an equivalent floating-point number. The output is a single floating-point number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

References

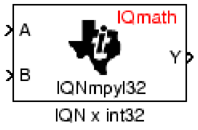
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN x int32

Multiply IQ number with long integer



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block multiplies an IQ input and a long integer input and produces an IQ output of the same Q value as the IQ input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

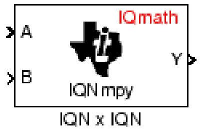
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN x IQN

Multiply IQ numbers with same Q format



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block multiplies two IQ numbers. Optionally, it can also round and saturate the result.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Multiply option

Type of multiplication to perform:

- `Multiply` — Multiply the numbers.
- `Multiply with Rounding` — Multiply the numbers and round the result.
- `Multiply with Rounding and Saturation` — Multiply the numbers and round and saturate the result to the maximum value.

References

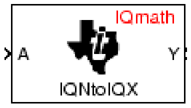
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN1 to IQN2

Convert IQ number to different Q format



IQN1 to IQN2

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block converts an IQ number in a particular Q format to a different Q format.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

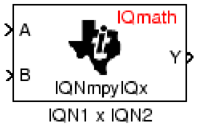
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN1 x IQN2

Multiply IQ numbers with different Q formats



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block multiples two IQ numbers when the numbers are represented in different Q formats. The format of the result is specified in the dialog box.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the

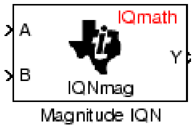
Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Magnitude IQN

Magnitude of two orthogonal IQ numbers



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block calculates the magnitude of two IQ numbers using

$$\sqrt{a^2 + b^2}$$

The output is an IQ number in the same Q format as the input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

References

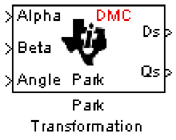
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Park Transformation

Convert two-phase stationary system vectors to rotating system vectors



Library

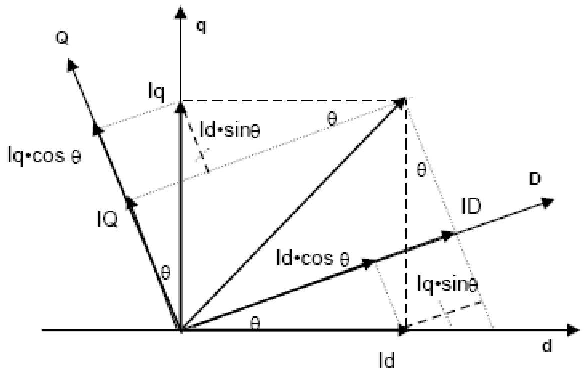
Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block converts vectors in balanced two-phase orthogonal stationary systems into an orthogonal rotating reference frame. The transformation implements these equations

$$ID = Id * \cos\theta + Iq * \sin\theta$$
$$IQ = -Id * \sin\theta + Iq * \cos\theta$$

and is illustrated in the following figure.



The variables used in the preceding figure and equations correspond to the block variables as shown in the following table:

	Equation Variables	Block Variables
Inputs	id	Alpha
	iq	Beta
	θ	Angle
Outputs	ID	Ds
	IQ	Qs

The inputs to this block are the direct axis (`Alpha`) and the quadrature axis (`Beta`) components of the transformed signal and the phase angle (`Angle`) between the stationary and rotating frames.

The outputs are the direct axis (`Ds`) and quadrature axis (`Qs`) components of the transformed signal in the rotating frame.

The instantaneous inputs are defined by the following equations:

$$id = I * \sin(\omega t)$$
$$iq = I * \sin(\omega t + \pi/2)$$

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.
- The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

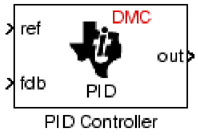
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Clarke Transformation, C2000 Inverse Park Transformation, C2000 PID Controller, C2000 Space Vector Generator, C2000 Speed Measurement

C2000 PID Controller

Digital PID controller

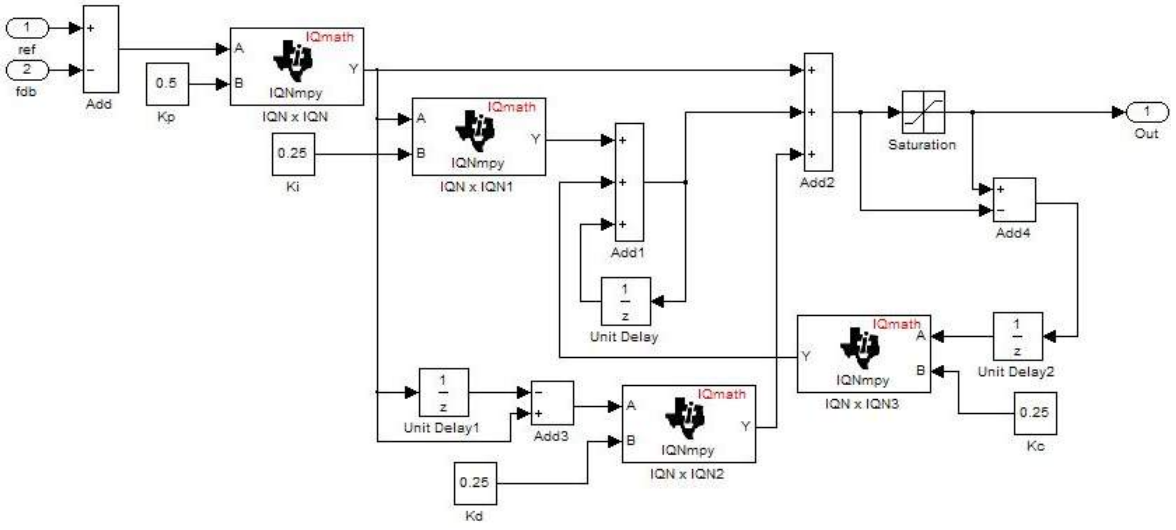


Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block implements a 32-bit digital PID controller with antiwindup correction. The inputs are a reference input (`ref`) and a feedback input (`fdb`) and the output (`out`) is the saturated PID output. The following diagram shows a PID controller with antiwindup.



The differential equation describing the PID controller before saturation that is implemented in this block is

$$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t)$$

where u_{presat} is the PID output before saturation, u_p is the proportional term, u_i is the integral term with saturation correction, and u_d is the derivative term.

The proportional term is

$$u_p(t) = K_p e(t)$$

where K_p is the proportional gain of the PID controller and $e(t)$ is the error between the reference and feedback inputs.

The integral term with saturation correction is

$$u_i(t) = \int_0^t \left\{ \frac{K_p}{T_i} e(\tau) + K_c\big(u(\tau) - u_{presat}(\tau)\big) \right\} d\tau$$

where K_c is the integral correction gain of the PID controller.

The derivative term is

$$u_d(t) = K_p T_d \frac{de(t)}{dt}$$

where T_d is the derivative time of the PID controller. In discrete terms, the derivative gain is defined as $K_d = T_d/T$, and the integral gain is defined as $K_i = T/T_i$, where T is the sampling period and T_i is the integral time of the PID controller.

Using backward approximation, the preceding differential equations can be transformed into the following discrete equations.

$$u_p[n] = K_p e[n]$$

$$u_i[n] = u_i[n-1] + K_i K_p e[n] + K_c\big(u[n-1] - u_{presat}[n-1]\big)$$

$$u_d[n] = K_d K_p (e[n] - e[n-1])$$

$$u_{presat}[n] = u_p[n] + u_i[n] + u_d[n]$$

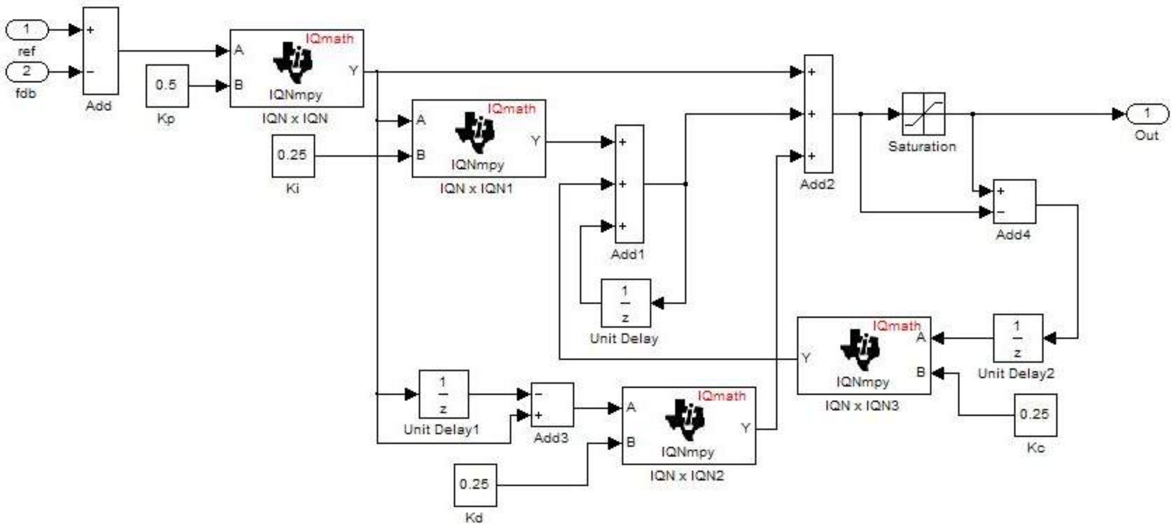
$$u[n] = SAT\big(u_{presat}[n]\big)$$

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.

  The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

This block implements a 32-bit digital PID controller with antiwindup correction. The inputs are a reference input (`ref`) and a feedback input (`fdb`) and the output (`out`) is the saturated PID output. The following diagram shows a PID controller with antiwindup.



The differential equation describing the PID controller before saturation that is implemented in this block is

$$u_{presat}(t) = u_p(t) + u_i(t) + u_d(t)$$

where u_{presat} is the PID output before saturation, u_p is the proportional term, u_i is the integral term with saturation correction, and u_d is the derivative term.

The proportional term is

$$u_p(t) = K_p e(t)$$

where K_p is the proportional gain of the PID controller and *e(t)* is the error between the reference and feedback inputs

$$u_i(t) = \int_0^t \left\{ \frac{K_p}{T_i} e(\tau) + K_c\left(u(\tau) - u_{presat}(\tau)\right) \right\} d\tau$$

where K_c is the integral correction gain of the PID controller.

The derivative term is

$$u_d(t) = K_p T_d \frac{de(t)}{dt}$$

where T_d is the derivative time of the PID controller. In discrete terms, the derivative gain is defined as $K_d = T_d/T$, and the integral gain is defined as $K_i = T/T_i$, where T is the sampling period and T_i is the integral time of the PID controller.

Using backward approximation, the preceding differential equations can be transformed into the following discrete equations.

$$u_p[n] = K_p e[n]$$

$$u_i[n] = u_i[n-1] + K_i K_p e[n] + K_c\left(u[n-1] - u_{presat}[n-1]\right)$$

$$u_d[n] = K_d K_p (e[n] - e[n-1])$$

$$u_{presat}[n] = u_p[n] + u_i[n] + u_d[n]$$

$$u[n] = SAT\left(u_{presat}[n]\right)$$

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.

  The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Proportional gain

Amount of proportional gain (K_p) to apply to the PID

Integral gain

Amount of gain (K_i) to apply to the integration equation

Integral correction gain

Amount of correction gain (K_c) to apply to the integration equation

Derivative gain

Amount of gain (K_d) to apply to the derivative equation.

Minimum output

Minimum allowable value of the PID output

Maximum output

Maximum allowable value of the PID output

References

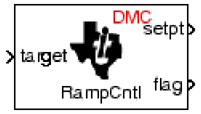
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Clarke Transformation, C2000 Inverse Park Transformation, C2000 Park Transformation, C2000 Space Vector Generator, C2000 Speed Measurement, “Simulation of FOC Using PMSM Model”, “Permanent Magnet Synchronous Motor Field-Oriented Control”

C2000 Ramp Control

Create ramp-up and ramp-down function



Ramp Control

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block implements a ramp-up and ramp-down function. The input is a `target` value and the outputs are the set point value (`setpt`) and a `flag`. The `flag` output is set to `7FFFFFFFh` when the output `setpt` value reaches the input `target` value. The `target` and `setpt` values are signed 32-bit fixed-point numbers with Q values between 16 and 29. The flag is a long number.

The `target` value is compared with the `setpt` value. If they are not equal, the output `setpt` is adjusted up or down by a fixed step size (0.0000305).

If the fixed step size is relatively large compared to the `target` value, the output may oscillate around the `target` value.

Parameters

Maximum delay rate

Value that is multiplied by the sampling loop time period to determine the time delay for each ramp step. Valid values are integers greater than 0.

Minimum limit

Minimum allowable ramp value. If the input falls below this value, it will be saturated to this minimum. The smallest value you can enter is the minimum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value below this minimum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its minimum value is -4.

Maximum limit

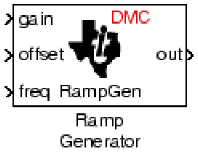
Maximum allowable ramp value. If the input goes above this value, it will be reduced to this maximum. The largest value you can enter is the maximum value that can be represented in fixed-point data format by the input and output blocks to which this Ramp Control block is connected in your model. If you enter a value above this maximum, an error occurs at the start of code generation or simulation. For example, if your input is in Q29 format, its maximum value is 3.9999....

See Also

C2000 Ramp Generator

C2000 Ramp Generator

Generate ramp output



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block generates ramp output (`out`) from the slope of the ramp signal (`gain`), DC offset in the ramp signal (`offset`), and frequency of the ramp signal (`freq`) inputs. All of the inputs and output are 32-bit fixed-point numbers with Q values between 1 and 29.

Algorithm

The block's output (`out`) at the sampling instant k is governed by the following algorithm:

`out`(k) = angle(k) * `gain`(k) + `offset`(k)

For `out`(k) > 1, `out`(k) = `out`(k) - 1. For `out`(k) < -1, `out`(k) = `out`(k) + 1.

Angle(k) is defined as follows:

angle(k) = angle(k-1) + `freq`(k) * **Maximum step angle**

for angle(k) > 1, angle(k) = angle(k) - 1

for angle(k) < -1, angle(k) = angle(k) + 1

The frequency of the ramp output is controlled by a precision frequency generation algorithm that relies on the modulo nature of the finite length variables. The frequency of the output ramp signal is equal to

f = (**Maximum step angle** * sampling rate) / 2^m

where m represents the fractional length of the data type of the inputs.

All math operations are carried out in fixed-point arithmetic, where the fixed-point fractional length is determined by the block's inputs.

Note To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.

Parameters

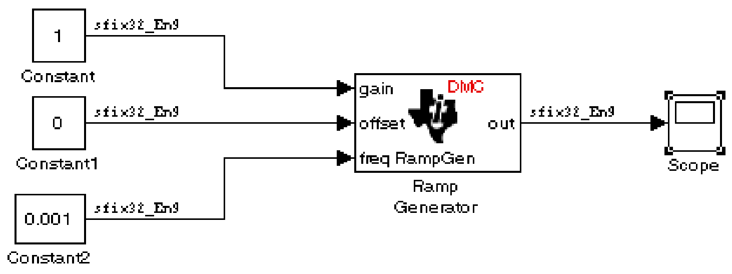
Maximum step angle

The maximum step size, which determines the rate of change of the output (i.e., the minimum period of the ramp signal).

When you enter double-precision floating-point values for parameters in the IQ Math blocks, the software converts them to single-precision values that are compatible with the behavior on c28x processor.

Examples

The following model demonstrates the Ramp Generator block. The Constant and Scope blocks are available in Simulink Commonly Used Blocks.



In your model, select **Simulation > Model Configuration Parameters**. On the **Solver** pane, set **Type** to `Fixed-step` and **Solver** to `Discrete (no continuous states)`. Set the parameter values for the blocks as shown in the following table.

Block	Connects to	Parameter	Value

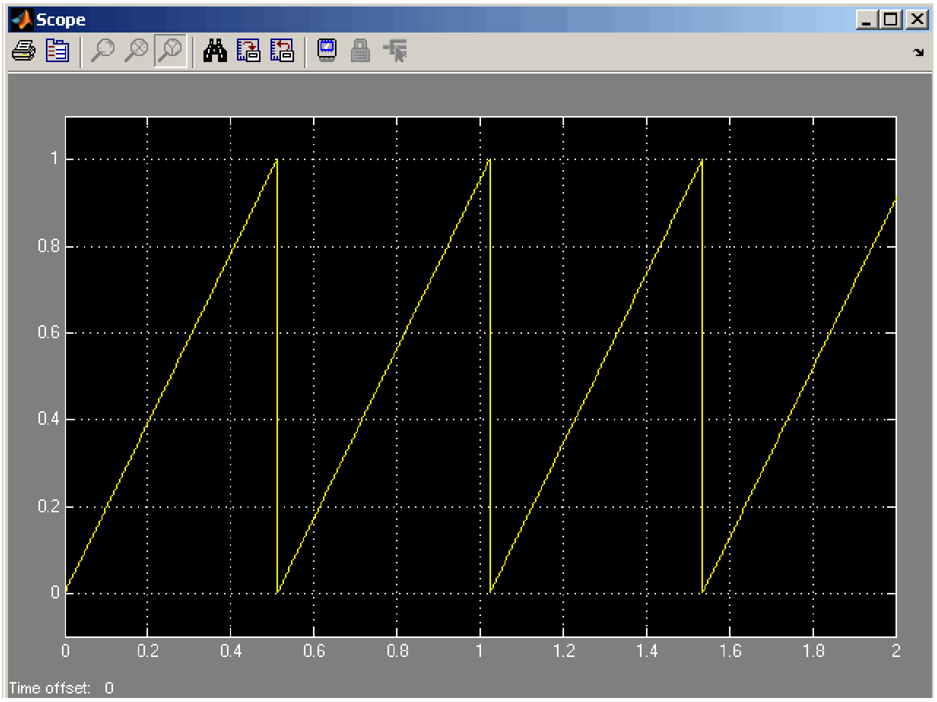
Constant	Ramp Generator - `gain`	`Constant value`
		`Sample time`
		`Output data type`
		`Output scalig value`
			`1`
			`0.001`
			`sfix(32)`
			`2^-9`

Constant	Ramp Generator - `offset`	`Constant value`
		`Sample time`
		`Output data type`
		`Output scalig value`
			`0`
			`inf`
			`sfix(32)`
			`2^-9`

Constant	Ramp Generator - `freq`	`Constant value`
		`Sample time`
		`Output data type`
		`Output scalig value`
			`0.001`
			`inf`
			`sfix(32)`
			`2^-9`

C2000 Ramp Generator	Scope and Floating Scope (Simulink block)	`Maximum step angle`	`1`

When you run the model, the Scope block generates the following output (drag a zoom box around a portion of the output to change the display).



With fixed point calculations in IQMath, for a given frequency input on the block, **f_input**, the equation is:

f = (Maximum step angle * f_input * sampling rate) / 2^m

For example, if f_input = 0.001, the real value, 1, counts as fixed point with a fractional length of 9:

f = (1 * 1 * (1/0.001)) / 2^9 = 1.9531 Hz

Where 0.001 is the block sample time.

If we use normal math, and f_input is a non-fixed point real value, then:

f = (Maximum step angle * f_input * sampling rate) / 1

For example, if we are using floating point calculation:

f = (1 * 0.001 * (1/0.001)) / 1 = 1 Hz

When using fixed point with fractional length 9, the expected period becomes:

$T = 1/f = 1/1.9531 \text{ Hz} = 0.5120 \text{ s}$

This result is what the above Scope output shows.

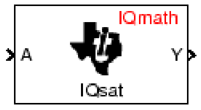
Note If you use different fractional lengths for the fixed point calculations, the output frequency varies depending on the precision.

See Also

C2000 Ramp Control

C2000 Saturate IQN

Saturate IQ number



Saturate IQN

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block saturates an input IQ number to the specified upper and lower limits. The returned value is an IQ number of the same Q value as the input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Upper Limit

Maximum real-world value to which to saturate

Lower Limit

Minimum real-world value to which to saturate

References

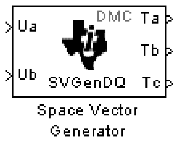
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Space Vector Generator

Duty ratios for stator reference voltage



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block calculates duty ratios that generate a given stator reference voltage using space vector PWM technique. Space vector pulse width modulation is a switching sequence of the upper three power devices of a three-phase voltage source inverter and is used in applications such as AC induction and permanent magnet synchronous motor drives. The switching scheme results in three pseudosinusoidal currents in the stator phases. This technique approximates a given stator reference voltage vector by combining the switching pattern corresponding to the basic space vectors.

The inputs to this block are

- Alpha component — the reference stator voltage vector on the direct axis stationary reference frame (`Ua`)
- Beta component — the reference stator voltage vector on the direct axis quadrature reference frame (`Ub`)

The alpha and beta components are transformed via the inverse Clarke equation and projected into reference phase voltages. These voltages are represented in the outputs as the duty ratios of the PWM1 (`Ta`), PWM3 (`Tb`), and PWM5 (`Tc`).

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See "Using the IQmath Library" for more information.

References

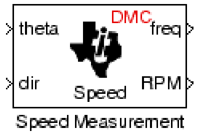
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, Literature Number SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Clarke Transformation, C2000 Inverse Park Transformation, C2000 Park Transformation, C2000 PID Controller, C2000 Speed Measurement

C2000 Speed Measurement

Calculate motor speed



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x DMC

Description

This block calculates the motor speed based on the rotor position when the direction information is available. The inputs are the electrical angle (`theta`) and the direction of rotation (`dir`) from the encoder. The outputs are the speed normalized from 0 to 1 in the Q format (`freq`) and the speed in revolutions per minute (`rpm`).

Note

- To generate optimized code from this block, enable the `TI C28x` or `TI C28x (ISO)` Code Replacement Library.
- This block does not call the corresponding Texas Instruments library function during code generation. Instead, the MathWorks code uses the TI functions global Q setting to adjust dynamically the Q format based on the block input. See “Using the IQmath Library” for more information.

Understanding the Theta Input to the Block

To indicate the rotational position of your motor, the block expects a 32-bit, fixed-point value that varies from 0 to 1.

Block input theta is defined by the following relations:

- A theta input signal equal to 0 indicates 0 degrees of rotation.
- A theta input signal equal to 1 indicates 360 degrees of rotation (one full rotation).

When the motor spins at a constant speed, theta (in counts) from your position sensor (encoder) should increase linearly from 0 to 1 and then abruptly return to 0, like a saw-shaped signal. Adjust the theta signal output from your encoder to get the input signal range for the Speed Measurement block. Then, convert your encoder signal to 32-bit fixed-point Q format that meets your resolution needs.

For example, if you are using a position sensor that generates 8000 counts for one full revolution of the motor, (0.0450 degrees per count), you need to reset your counter to 0 after your counter reaches 8000. Each time you read your encoder position, you need to convert the position to a 32-bit, fixed-point Q format value knowing that 8000 is represented as a 1.0. In this example your format could be Q31.

The Base Speed Parameter

Base speed is the maximum motor rotation rate to measure. This value is probably not the maximum speed the motor can achieve.

The Speed Measurement block calculates motor speed from two successive *theta* readings of the motor position, $theta_{new}$ and $theta_{old}$ (the base speed of the motor; and the time between readings). The maximum speed the block can calculate occurs when the difference between two successive samples [abs($theta_{new}$-$theta_{old}$)] is 1.0—one full motor revolution occurs between theta samples.

Therefore, the value you provide for the Base speed (in revolutions per minute) parameter is the speed, in revolutions per minute, at which your motor position signal reports one full revolution during one sample time. While the motor may spin faster than the base speed, the block cannot calculate the rotation rate in that case. If the motor completes more than one revolution in one sample time, the calculated speed may be wrong. The block does not know that between samples $theta_{new}$ and $theta_{old}$, *theta* wrapped from 1 back to 0 and started counting up again.

The time difference between the two theta readings is the sample time. The Speed Measurement block inherits the sample time from the upstream block in your model. You set the sample time in the upstream block and then the Speed Measurement block uses that sample time to calculate the rotation rate of the motor.

The Sample Time Calculation

Motor speed measurements depend on the sample time you set in the model. Your sample time must be short enough to measure the full speed of the motor.

Two parameters drive your sample time—motor base speed and encoder counts per revolution. To be able to measure the maximum rotation rate, you must take at least one sample for each revolution. For a motor with base speed equal to 1000 rpm, which is 16.67 rps, you need to sample at 1/16.67 s, which is 0.06 s/sample. This sample rate of 16.67 samples per second is the maximum sample time (lowest sample rate) so that you can measure the full speed of the motor.

Using the same sample rate assumption, the minimum speed the block can measure depends on the encoder counts per revolution. At the minimum measurable motor speed, the encoder generates one count per sample period—16.67 counts per second. For an encoder that generates 8000 counts per revolution, this results in being able to measure a speed of [(16.67 counts/s) * (0.045 degrees/count)] = 0.752 degrees per second, or about 45 degrees per minute—one-eighth RPM.

The Differentiator Constant

The differentiator constant is a scalar value applied to the block output. For example, setting it to `1` does not alter the output. Setting the constant to `1/4` multiplies the frequency and revolutions per minute outputs by 0.25. This setting can be useful when your motor has multiple pole pairs, and one electrical revolution is not equal to one mechanical revolution. The constant lets you account for the difference between electrical and mechanical rotation rates.

The Low-Pass Filter Constant

This block includes filtering capability if your position signal is noisy. Setting the filter constant to `0` disables the filter. Setting the filter constant to `1` filters out the entire signal and results in a block output equal to 0. Use a simulation to determine the best filter constant for your system. Your goal is to filter enough to remove the noise on your signal but not so much that the speed measurements cannot react to abrupt speed changes.

Parameters

Base speed

Maximum speed of the motor to measure in revolutions per minute.

Differentiator constant

Constant used in the differentiator equation that describes the rotor position.

Low-pass filter constant

Constant to apply to the lowpass filter. This constant is $1/(1+T*(2\pi f_c))$, where T is the sampling period and f_c is the cutoff frequency. The $1/(2\pi f_c)$ term is the lowpass filter time constant. This block uses a lowpass filter to reduce noise generated by the differentiator.

Example

The following example demonstrates how you configure the Speed Measurement block.

Configuring the Speed Measurement Block to Measure Motor Speed

Use the following process to set up the Speed Measurement block parameters.

1 Add the block to your model.
2 Open the block dialog box to view the block parameters.
3 Set the value for **Base Speed** to the maximum speed to measure, in revolutions per minute.
4 Enter values for **Differentiator** and **Low-Pass Filter Constant**.
5 Click **OK** to close the dialog box.

Setting the Sample Time to Measure Motor Speed

Use the following process to set the sample time for measuring the motor speed.

1 Open the block dialog box for the block before the Speed Measurement block in your model (the upstream or driving block).

2 Set the sample time parameter in the upstream block according to the sample time guidelines described in “The Sample Time Calculation” on page 2-145.

3 Click **OK** to close the dialog box.

References

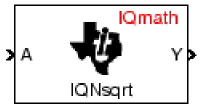
For detailed information on the DMC library, see *C/F 28xx Digital Motor Control Library*, SPRC080, available at the Texas Instruments Web site.

See Also

C2000 Clarke Transformation, C2000 Inverse Park Transformation, C2000 Park Transformation, C2000 PID Controller, C2000 Space Vector Generator

C2000 Square Root IQN

Square root or inverse square root of IQ number



Square Root IQN

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

Negative inputs to this block return a value of zero.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Whether to calculate the square root or inverse square root

- `Square root (_sqrt)` — Compute the square root.
- `Inverse square root (_isqrt)` — Compute the inverse square root.

References

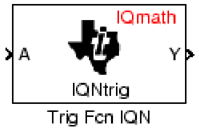
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, Fractional part IQN x int32, C2000 Integer part IQN, Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Trig Fcn IQN

C2000 Trig Fcn IQN

Sine, cosine, or arc tangent of IQ number



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ Optimization/ C28x IQmath

Description

This block calculates basic trigonometric functions and returns the result as an IQ number. Valid Q values for `_IQsinPU` and `_IQcosPU` are 1 to 30. For all others, valid Q values are from 1 to 29.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Type of trigonometric function to calculate:

- `_IQsin` — Compute the sine `(sin(A))`, where A is in radians.
- `_IQsinPU` — Compute the sine per unit `(sin(2*pi*A))`, where A is in per-unit radians.
- `_IQcos` — Compute the cosine `(cos(A))`, where A is in radians.

- `_IQcosPU` — Compute the cosine per unit `(cos(2*pi*A))`, where A is in per-unit radians.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

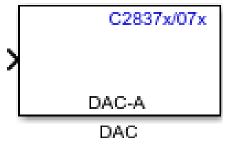
See Also

C2000 Absolute IQN, C2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN

C2837x/07x DAC

Configures the DAC to generate an analog output on the specified DAC channel A/B/C (12-bit) of F2837x/F2807x processor

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xS



Description

Generate an analog output on the specified DAC channel A, B, or C for F2837x/F2807x processors. The block accepts a 12–bit value as an input in the range 0 to 4095. You can saturate a value higher than 4095 to 4095 with the **Saturate on input outflow** option.

The output pins of this block are multiplexed with the ADC block input. When you use a DAC block in your model, the corresponding channels of the ADC cannot be used as input. If used, the ADC samples the DAC output.

The pins that are shared between ADC and DAC are DACOUTA/ADCINA0, DACOUTB/ADCINA1, and DACOUTC/ADCINB1. The input to the DAC block can be double, float, int, and uint. The block typecasts the input to uint16.

Ports

Input

`Port_1` — Input signal
real, scalar

Parameters

`DAC channel` — The DAC channel to generate analog output
12 (default)

Enter the DAC channel on which to generate the analog signal. The DAC channels that you can select are A, B, or C.

`Saturate on input overflow (>4095)` — The option to saturate the input value on input overflow

Select this check box to saturate the input value to 4095 when there is an input value is higher than 4095.

See Also

Topics

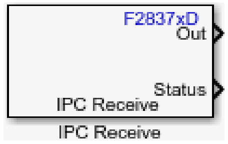
C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ePWM
C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x ADC

Introduced in R2016b

F2837xD IPC Receive

Receive data from either CPU

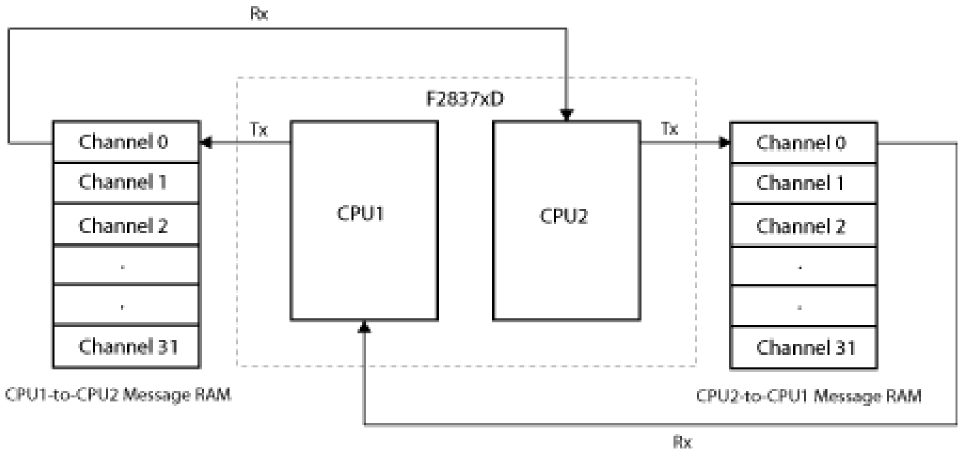
Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD



Description

The IPC Receive block receives and outputs data sent from one CPU to the other.

CPU1 transmits data to its allocated memory (CPU1-to-CPU2 Message RAM) and receives data from the allocated memory of CPU2 (CPU2-to-CPU1 Message RAM). CPU2 transmits data to CPU2-to-CPU1 Message RAM and receives data from CPU1-to-CPU2 Message RAM.



A hardware interrupt block can be used along with the IPC Receive block for receiving data based on hardware interrupts. Channels 0, 1, 2, and 3 are configured for hardware interrupts — IPC0, IPC1, IPC2, and IPC3. These hardware interrupts can be set in the hardware interrupt block using these parameters: **CPU interrupt number** 1 and **PIE interrupt numbers** 13, 14, 15, and 16 respectively.

Ports

Output

`Out` — IPC receive
vector | scalar

Data read from the other CPU.

`Status` — IPC receive status
0 | 1 | 2 | 3

The status port outputs one of these values:

- 0 — No errors
- 1 — Data not available
- 2 — Data type mismatch
- 3 — Data length mismatch

Parameters

`Channel` — Channel selected to receive data
`0` (default) | 0–31

The channel at which you want to receive data. Each channel is a separate memory location in the shared memory.

Note The transmitter and receiver have 32 channels each to transmit and receive data. For data transmission and reception, the transmitter and receiver must be set to the same channel number.

`Data type` — Type of data to be received
`uint16` (default) | single | int8 | uint8 | int16 | int32 | uint32 | boolean

The type of data the block receives.

Vector data is stored in the global shared RAM, and the address of the data is stored in the MSGRAM.

`Data length` — Size of data to be received
`1` (default) | positive integer

The number of data units received at each sample time. If the data length is 1, the block interprets each incoming piece of data as a scalar value; if the data length is greater than 1, the block interprets each incoming piece of data as a vector with length equal to **Data length**. The maximum size for scalar and vector data is 32 bits.

`Enable blocking` — Specify if CPU must wait to read data
`off` (default) | on

When enabled, the CPU waits until data is available from the other CPU.

`Sample time` — Interval at which block reads data
`0.001` (default) | `–1` | positive scalar

The time between data samples, measured in seconds. When you set this parameter to `-1`, Simulink determines the best sample time for the block based on the block context within the model.

See Also

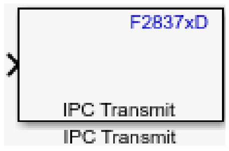
F2837xD IPC Transmit

Introduced in R2018a

F2837xD IPC Transmit

Transmit data to either CPU

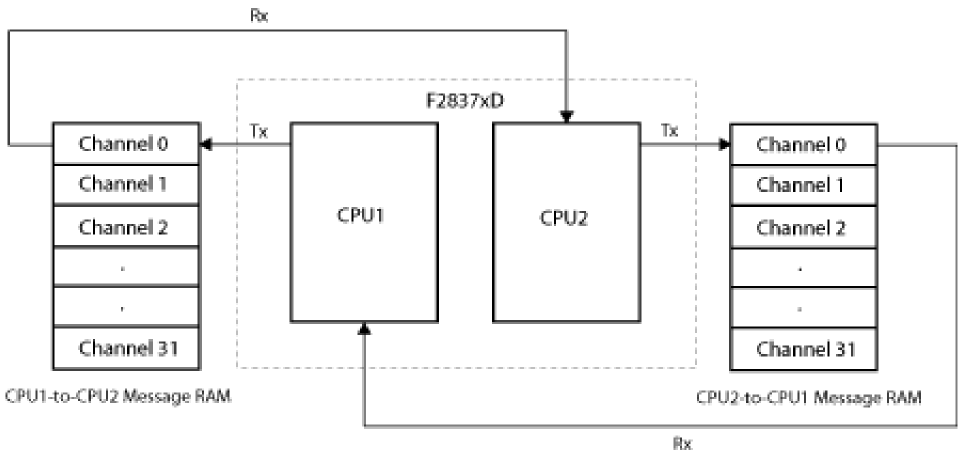
Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD



Description

The IPC Transmit block transmits data from one CPU to the other.

CPU1 transmits data to its allocated memory (CPU1-to-CPU2 Message RAM) and receives data from the allocated memory of CPU2 (CPU2-to-CPU1 Message RAM). CPU2 transmits data to CPU2-to-CPU1 Message RAM and receives data from CPU1-to-CPU2 Message RAM.



Ports

Input

`Input` — Data to be send to the other CPU
uint16 | single | int8 | uint8 | int16 | int32 | uint32 | boolean

The port accepts data to be transmitted to the other CPU as a vector or scalar.

Parameters

`Channel` — Channel selected to transmit data
`0` (default) | 0–31

The channel from which you want to transmit data. Each channel is a separate memory location in the shared memory.

Note The transmitter and receiver have 32 channels each to transmit and receive data. For data transmission and reception, the transmitter and receiver must be set to the same channel number.

`Enable blocking` — Specify if CPU must wait until sent data is read
`off` (default) | on

When enabled, after sending data, the CPU waits until the other CPU reads the data.

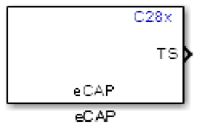
See Also

F2837xD IPC Receive

Introduced in R2018a

C28x eCAP

Receive and log transitions on capture input pin or configure auxiliary pulse width modulator



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The eCAP block captures the timing of important external events, such as Hall sensor signals in speed measurements of rotating machinery. When not used in capture mode, the block can be used in APWM mode, which is a single-channel, asymmetric pulse width modulator (APWM). You can add one eCAP block to your model for each capture pin. You cannot assign the same eCAP pin to two eCAP blocks in a model. eCAP and APWM modes use the same pins. In eCAP mode, the pins are used as input to capture the transitions. In APWM mode, the pins are used to output a PWM waveform.

Input/Output Ports

Input

SI — Synchronization input from software

scalar

The input from the software used to synchronize the eCAP counter. The synchronization occurs when the synchronization input value is 1.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > eCAP** or **APWM**.
- On the **General** tab, you select **Enable counter Sync-In mode** and **Enable software-forced counter synchronizing input**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

RA — One-Shot capture sequence

scalar

Starts a One-Shot capture sequence.

A 2-bit stop register is used to compare the Mod4 counter output, and when the register and counter values are equal the Mod4 counter is stopped.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > eCAP**.

- On the **eCAP** tab, you set **Select mode control** > **One-Shot** and select **Enable One-Shot re-arming control input**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`T` — APWM period
scalar

Period of the APWM.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode** > **APWM**.
- On the **APWM** tab, you select **Waveform period source** > **Input port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`W` — APWM width
scalar

Width of the APWM.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode** > **APWM**.
- On the **APWM** tab, you select **Duty cycle source** > **Input port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Output

The output ports appear only in eCAP mode.

`TS` — Output timestamps of capture events
vector

The signal width can take a value of 1, 2, 3, or 4 depending on the capture event selected in **Stop value after** on the **eCAP** tab. Use the **Enable reset counter after capture event # time-stamp** option to reset the counter after an event. This option is useful for finding the time difference between the events.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`CF` — Status of capture event
vector

The status of the capture event. `0` indicates that no event has occurred. `1` indicates that the event specified by the **Stop value after** parameter has occurred at the eCAP pin.

Dependencies

The port appears only when you select **Enable capture event status flag output** on the **General** tab.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`OF` — Status of overflow
scalar

The status of overflow. `0` indicates that no event has occurred. `1` indicates that the counter has overflowed from the highest value to 0.

Dependencies

The port appears only when you select **Enable overflow status flag output** on the **General** tab.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

General

`Operating mode` — Select eCAP or APWM mode
`eCAP` (default) | `APWM`

When you select `eCAP`, the block captures and logs pin transitions for each capture unit to a FIFO buffer. When you select `APWM`, the block generates asymmetric pulse width modulation (APWM) waveforms for driving downstream systems.

`eCAPx pin` — Select capture unit pin

`eCAP1` (default) | `eCAP2` | `eCAP3` | `eCAP4` | `eCAP5` | `eCAP6` | `eCAP7`

Select the required eCAP module to have a dedicated eCAP pin for capturing the external events.

The pin selection for the eCAP module can be done by browsing to **Hardware Implementation** > **Target hardware resources**. The selection option is provided only if the module has more than one pin that can be configured for an eCAP module.

`Counter phase offset value (0 ~ 4294967295)` — Time base for event captures

`0` (default) | integer in [0 4294967295]

This value provides the time base for event captures, clocked by the system clock. A phase register is used to synchronize with other counters via software- or hardware-forced synchronization. For information about software- or hardware-forced synchronization, see the **Enable counter Sync-In mode** parameter. This value is useful in APWM mode when you need a phase offset between capture modules. Set the phase offset to an integer from 0 to 42949667295 (2^{32}) counts.

`Enable counter Sync-In mode` — Enable TSCTR counter to load from TSCTR register

`off` (default) | `on`

Synchronization can be done using the SYNCI event or the software. When synchronization occurs, the shadow register CTRPHS is loaded into the active counter TSCTR in the current eCAP module and the eCAP modules downstream.

`Enable software-forced counter synchronizing input` — Synchronize one or more eCAP time bases

`off` (default) | `on`

A software method for synchronizing one or more eCAP time bases. The synchronization occurs when the synchronization input value is 1.

Dependencies

This parameter appears only when **Enable counter Sync-In mode** is selected.

`Sync output selection` — Synchronize eCAP counter with other eCAP counters
`CTR=PRD` (default) | `Pass through` | `Disabled`

Synchronizes an eCAP counter with other eCAP counters. The options are:

- `CTR=PRD` — Triggers the sync-out signal when the counter value equals the period.
- `Pass through` — The sync-in event is passed through as the sync-out signal.
- `Disabled` — Disables the sync-out signal.

`Sample time` — Frequency at which block reads input pin value
`0.001` (default)

Sample time for the block in seconds.

eCAP

To enable configuration parameters on the eCAP tab, set **Operating mode** to `eCAP` on the **General** tab.

`Event prescaler (integer from 0 to 31)` — Prescales input signal in mutiples of 2
`0` (default) | scalar integer in [0 31]

The input signal is prescaled by twice the value of this parameter. For example, if you enter `1`, the input is prescaled by 2, and for `31`, the input is prescaled by 62. Entering `0` bypasses the input prescaler, leaving the input capture signal unchanged.

`Select mode control` — Mode of capture
`Continuous` (default) | `One-Shot`

The `Continuous` option performs continuous timestamp captures (events 1 through 4) using a circular buffer.

The `One-Shot` option enables the **Enable One-Shot rearming control via input port** option.

`Enable One-Shot rearming control via input port` — Re-arms a One-Shot capture sequence
`off` (default) | `on`

When this parameter is selected, a One-Shot capture sequence is re-armed as follows:

1 Mod4 counter is reset to zero.
2 Mod4 counter is unfrozen.
3 Capture register loading is enabled.

Dependencies

This parameter appears only when you select `One-Shot` for **Select mode control**.

`Stop value after` — Number of capture events after which capture stops
`Capture Event 1` (default) | `Capture Event 2` | `Capture Event 3` | `Capture Event 4`

The number of capture events after which you want to stop the capture sequence.

`Enable reset counter after capture event # time-stamp` — Resets counter after capture event
`off` (default) | `on`

The eCAP process resets the counter after receiving a capture event timestamp. In this case, # represents the number of the capture event set in the **Stop value after** parameter.

`Select capture event # polarity` — Start capture event on rising edge or falling edge
`Rising Edge` (default) | `Falling Edge`

The option that starts a capture event. In this case, # represents the number of the capture event set in the **Stop value after** parameter.

`Time-Stamp counter data type` — Data type of counter
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the timestamp counter.

`Enable capture event status flag output` — Output capture event status
`off` (default) | `on`

Outputs the capture event status flag at the output port **CF**. The block outputs `0` until the event is captured. After the event, the flag value is `1`.

`Capture flag data type` — Data type of output port CF
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the output port **CF**.

Dependencies

This parameter appears only when you select **Enable capture event status flag output**.

`Enable overflow status flag output` — Output status of elements of FIFO buffer
off (default) | on

Outputs the status of the elements of the FIFO buffer at the output port **OF**.

`Overflow flag data type` — Data type of output port OF
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the output port **OF**.

Dependencies

This parameter appears only when you select **Enable overflow status flag output**.

APWM

To enable configuration parameters on the APWM tab, set **Operating mode** to `APWM` in the **General** tab.

`Waveform period units` — Units for measuring waveform period
`Seconds` (default) | `Clock cycles`

`Clock cycles` uses the high-speed peripheral clock cycles of the processor.

`Waveform period source` — Source from which waveform period value is obtained
`Specify via dialog` (default) | `Input port`

Select `Specify via dialog` to enter the value in **Waveform period**, or select `Input port` to use a value from the **T** input port.

`Waveform period` — Period of PWM waveform
`0.001` (default)

Period of the PWM waveform measured in clock cycles or seconds, as specified in **Waveform period units**.

Note The term clock cycles refers to the high-speed peripheral clock on the F2812 chip. This high-speed peripheral clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

Dependencies

This parameter appears only when **Waveform period source** is set to `Specify via dialog`.

`Duty cycle units` — Units for measuring duty cycle

`Percentages` (default) | `Clock cycles`

The units used for measuring the duty cycle.

`Duty cycle source` — Source from which duty cycle for PWM waveform is obtained

`Specify via dialog` (default) | `Input port`

Select `Specify via dialog` to enter the value in **Duty cycle**, or select `Input port` to use a value from the **W** input port.

`Duty cycle` — Ratio of PWM waveform pulse duration to PWM waveform period

50 (default)

The ratio of PWM waveform pulse duration to PWM waveform period. This ratio is expressed in **Duty cycle units**.

`Output polarity select` — Set active level for output

`Active High` (default) | `Active Low`

When you select `Active High`, the compare value (duty cycle) defines the high time. Selecting `Active Low` directs the compare value to define the low time.

Interrupt

`Post interrupt on capture event #` — Set interrupt source to capture event

`off` (default) | `on`

You can use the C28x Hardware Interrupt block to respond to this interrupt. In this case, # represents the number of the capture event set in the **Stop value after** parameter.

Dependencies

This parameter appears only when you set **Operating mode** to `eCAP` in the **General** tab.

`Post interrupt on counter overflow` — Trigger interrupt on counter overflow
`off` (default) | `on`

Triggers an interrupt when the counter overflows.

Dependencies

This parameter appears only when you set **Operating mode** to `eCAP` in the **General** tab.

`Post interrupt on counter equal period match` — Post interrupt when counter equals period register
`off` (default) | `on`

Posts interrupt when the value of counter is same as the value of the period register (CTR = PRD).

Dependencies

This parameter appears only when you set **Operating mode** to `APWM` in the **General** tab.

`Post interrupt on counter equal compare match` — Post interrupt when counter equals compare register
`off` (default) | `on`

Posts interrupt when the value of the counter is same as the value of the compare register (CTR = CMP).

Dependencies

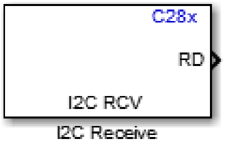
This parameter appears only when you set **Operating mode** to `APWM` in the **General** tab.

See Also

C28x Hardware Interrupt

C28x I2C Receive

Configure inter-integrated circuit (I2C) module to receive data from I2C bus



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The I2C Receive block configures the inter-integrated circuit (I2C) module to receive data from the two-wire I2C serial bus. The I2C Receive block supports I2C bus communication between the processor and external peripherals or other controllers. The block can run in either slave or master mode.

When the I2C module is configured as master, the module receives data from a slave. When the I2C module is configured as a slave, the module receives data from the master. Configure the I2C module by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources**.

To read data from a slave, send the address of the register to be read using the I2C Transmit block to the slave. Ensure that the data is sent from the Tx FIFO to the slave before the data is read from the slave using the I2C Receive block. For more information, see “Using the I2C Bus to Access Sensors”.

Input/Output Ports

Input

`SAR` — Slave address register value
scalar

The slave address register value.

Dependencies

This port appears only when **Slave address source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Output

`RD` — Received data from I2C bus
scalar | vector

The data read from the I2C bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

`status` — I2C communication status
scalar

Status values from the I2C status register (I2CSTR).

Dependencies

This port appears only when **Output receiving status** is selected.

Data Types: `uint16`

Parameters

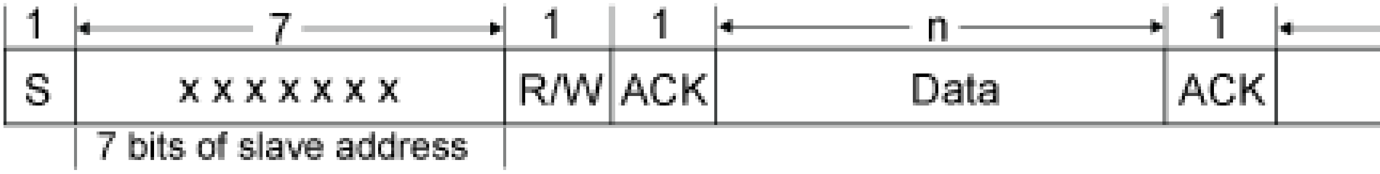
`Module` — Module for communication
`I2C_A` (default) | `I2C_B`

The I2C module to be used for communication. The number of I2C modules supported varies across different C2000 processors.

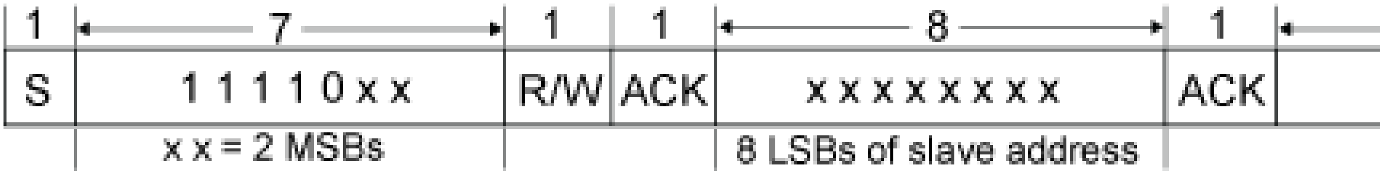
`Addressing format` — Address format for communication
`7–Bit addressing` (default) | `10–Bit addressing` | `Free data format`

The address format for communication. The diagram shows the format for each option. The I2C Receive block sets the R/W bit to 0.

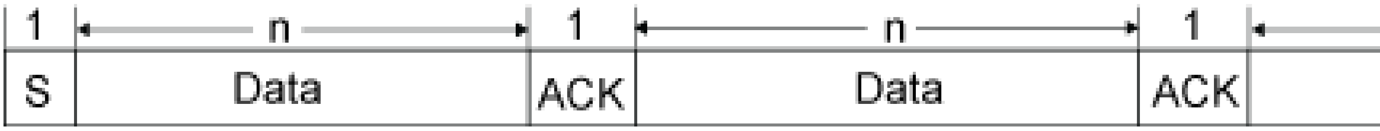
I2C Module 7-bit Addressing Format



I2C Module 10-bit Addressing Format



I2C Module Free Data Format



S — Start bit

R/W — Read/Write

ACK — Acknowledge

P — Stop bit

MSB — Most significant bit

LSB — Least significant bit

`Slave address source` — Slave address source of I2C slave
`Specify via dialog` (default) | `Input port`

The method for setting the slave address register of the I2C slave.

`Slave address register` — Slave address register value
`80` (default) | scalar

Enter a 7- or 10-bit slave address according to the addressing format selected.

Dependencies

This parameter appears only when **Slave address source** is set to `Specify via dialog`.

`Bit count` — Bit count for communication
8 (default) | integer in [1 8]

The number of bits in the data byte received by the I2C module.

`Read data length` — Length of received data
`1` (default) | scalar

The number of **Data type** the block receives (not bytes). If this parameter is set to more than `1`, the output will be a vector.

`Initial output` — Value of I2C node output to model

`0` (default) | scalar | vector

The value the I2C node outputs to the model before it has received data. By default, the block outputs `0` if the I2C value is not received.

`Set NACK bit` — NACK bit during I2C acknowledge cycle

`off` (default) | `on`

Generates a no-acknowledge bit (NACK) during the I2C acknowledge cycle and ignores new bits from the transmitting I2C node.

`Enable stop condition` — Stop message to I2C Transmit block

`off` (default) | `on`

Enables the I2C Receive block (master) to send a stop message to the I2C Transmit block (slave).

`Output receiving status` — Indicates when I2C Receive block receives message

`off` (default) | `on`

Enables the status output port, which indicates when the I2C Receive block receives a message.

`Sample time` — Frequency at which data is read from I2C device

`0.001` (default) | –1 | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

`Data type` — Type of data in data vector

`int8` (default) | `uint8` | `int16` | `uint16` | `int32` | `uint32`

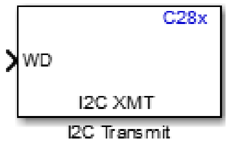
Sets the data type of the data received. If the size of the received data is less than 8 bits, then the data is right-justified.

See Also

C28x I2C Transmit

C28x I2C Transmit

Configure inter-integrated circuit (I2C) module to transmit data to I2C bus



Library:	Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The I2C Transmit block configures the inter-integrated circuit (I2C) module to transmit data to the two-wire I2C serial bus. The I2C Transmit block supports I2C bus communication between the processor and external peripherals or other controllers. The block can run in either slave or master mode.

When the I2C module is configured as master, the module receives data from a slave. When the I2C module is configured as a slave, the module receives data from the master. Configure the I2C module by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources**.

To read data from a slave, send the address of the register to be read using the I2C Transmit block to the slave. Ensure that the data is sent from the Tx FIFO to the slave before the data is read from the slave using the I2C Receive block. For more information, see “Using the I2C Bus to Access Sensors”.

Input/Output Ports

Input

`SAR` — Slave address register value

scalar

The slave address register value.

Dependencies

This port appears only when **Slave address source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`WD` — Data written to I2C bus

scalar | vector

The data written to the I2C bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Output

`status` — I2C communication status

scalar

Status values from the I2C status register (I2CSTR).

Dependencies

This port appears only when **Output transmitting status** is selected.

Data Types: `uint16`

Parameters

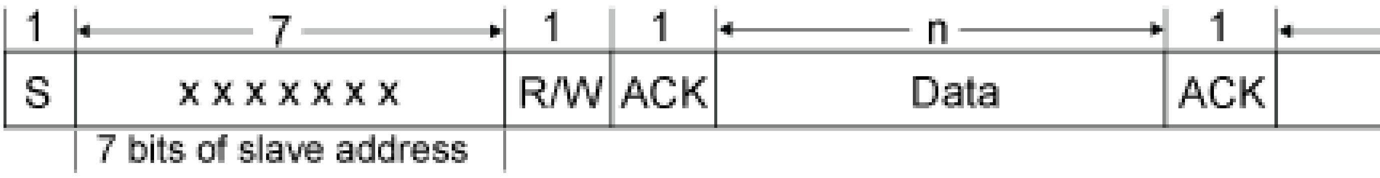
`Module` — Module for communication
`I2C_A` (default) | `I2C_B`

The I2C module to be used for communication. The number of I2C modules supported varies across different C2000 processors.

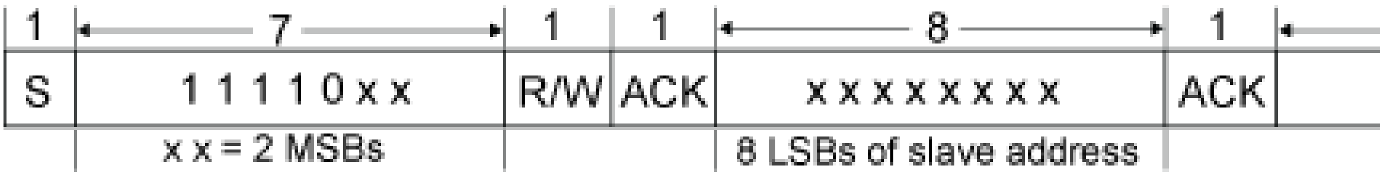
`Addressing format` — Address format for communication
`7–Bit addressing` (default) | `10–Bit addressing` | `Free data format`

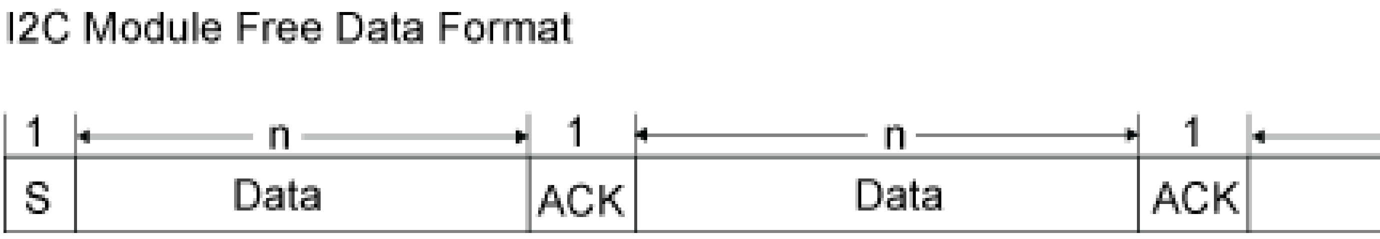
The address format for communication. The diagram shows the format for each option. The I2C Transmit block sets the R/W bit to 1.

I2C Module 7-bit Addressing Format



I2C Module 10-bit Addressing Format





S — Start bit

R/W — Read/Write

ACK — Acknowledge

P — Stop bit

MSB — Most significant bit

LSB — Least significant bit

`Slave address source` — Slave address source of I2C slave
`Specify via dialog` (default) | `Input port`

The method for setting the slave address register of the I2C slave.

`Slave address register` — Slave address register value
`80` (default) | scalar

Enter a 7- or 10-bit slave address according to the addressing format selected.

Dependencies

This parameter appears only when **Slave address source** is set to `Specify via dialog`.

`Bit count` — Bit count for communication
8 (default) | integer in [1 8]

The number of bits in the data byte received by the I2C module.

`Enable stop condition` — Send stop bit to indicate that data transmission is complete
`off` (default) | `on`

When the I2C module is configured as master, the I2C module sends out a stop bit to the I2C bus to indicate that the data transmission is complete. The I2C bus is free for any other I2C module to initiate a read/write operation.

`Enable repeat mode` — Retransmit data until stop or start condition is detected
`off` (default) | `on`

When you enable repeat mode, the I2C module transmits data continuously until it detects a stop or start condition. If you use this mode, also consider selecting **Enable stop condition** to ensure that data transmit stops after the stop condition.

If you disable repeat mode, the I2C module operates in standard mode, sending a specific number of data values once.

`Output transmitting status` — Indicates when I2C transmit block transmits message
`off` (default) | `on`

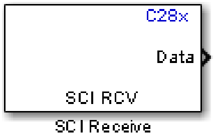
Enables the status output port, which indicates when the I2C transmit block transmits a message.

See Also

C28x I2C Receive

C28x SCI Receive

Receive data on target via serial communication interface (SCI) from host



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2838x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The SCI Receive block supports asynchronous serial digital communication between the processor and other asynchronous peripherals. This block receives scalar or vector data using the specified SCI hardware module.

A model can only contain one SCI Receive block for each module. The C28x processor has three SCI modules — A, B, and C. You can configure the SCI modules by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Input/Output Ports

Output

`Data` — Data received from serial bus
scalar | vector

The data received from the serial communication bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single`

`status` — Status of serial communication
scalar

Indicates the status of the received serial data:

- `0` — No errors.
- `1` — A time-out occurred while the block was waiting to receive data.
- `2` — The received data contains an error (checksum error).
- `3` — SCI parity error flag: occurs when a character is received with a mismatch.
- `4` — SCI framing error flag: occurs when an expected stop bit is not found.
- `5` — SCI overrun error flag: occurs when a character is transferred to the receive registers before reading the previous character.
- `6` — SCI break-detect flag: occurs when SCI receiver data line (SCIRXD) remains continuously low for at least ten bits.

Dependencies

This port appears only when you select **Output receiving status**.

Data Types: `uint16`

Parameters

`SCI module` — SCI module for communication

A (default) | B | C | D

The SCI module used for communication. The number of SCI modules supported varies across different C2000 processors.

`Additional package header` — Indicates start of data

`'S'` (default) | string | char | number from 0 to 255

The data located at the front of the received data package, which is not part of the data being received, and indicates the start of data. The additional package header must be represented using ASCII characters. You can use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not received or included in the total byte count. To specify a null value (no package header), enter two single quotes only.

Note Match additional package headers or terminators with those specified in the host SCI Transmit block.

`Additional package terminator` — Indicates end of data

`'E'` (default) | string | char | number from 0 to 255

The data located at the end of the received data package, which is not part of the data being received, and indicates the end of data. The additional package terminator must be represented using ASCII characters. Use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not received or included in the total byte count. To specify a null value (no package terminator), enter two single quotes only.

`Data type` — Data type of output data

`uint8` (default) | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32`

The data type of the output data.

`Data length` — Number of data types the block receives
`1` (default) | positive integer, finite

The number of **Data type** the block receives (not bytes). If this parameter is set to more than `1`, the output will be a vector. Ensure that the data length specified is same as that of the SCI Transmit block from which data is received.

`Initial output` — Default value output from block
`0` (default)

The default value output from the SCI Receive block. This value is output, for example,when the **Action taken when connection timeout** parameter is set to `Output the last received value` and a connection time-out occurs before data is received.

`Action taken when connection times out` — Output when connection time out occurs
`Output the last received value` (default) | `Output custom value`

Specifies what to output when a connection time out occurs. If `Output the last received value` is selected, the block outputs the last received value. If a value has not been received, the block outputs the **Initial output** value.

If you select `Output custom value`, use the **Output value when connection times out** parameter to set the custom value.

`Output value when connection times out` — Set custom time out value
`0` (default) | scalar | vector

Set the custom time out value.

`Sample time` — Frequency at which data is read from SCI device
`0.1` (default) | `-1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

`Output receiving status` — Status of serial communication
`off` (default) | `on`

Creates a **Status** block output that provides the status of serial communication.

`Enable receive FIFO interrupt` — Posts interrupt when FIFO is full
`off` (default) | `on`

If this option is selected, an interrupt is posted when the FIFO is full, allowing the subsystem to perform any action. For example, you can use the C28x Hardware Interrupt block for triggering the SCI Receive block to read the data as soon as it is received.

If the option is not selected, the SCI Receive block is in polling mode and checks the FIFO for data. If data is present, the block reads and outputs the data. If data is not present, in blocking mode, the block waits until data is available. In non-blocking mode, the block continues with the execution of the algorithm without waiting for data.

`Receive FIFO interrupt level (maximum 4 for Piccolo devices)` — Level for triggering interrupt
`1` (default) | `integer in the range of [1 16]`

The receive FIFO generates an interrupt when the number of data bytes in the receive FIFO is greater than or equal to the value selected for this parameter.

Dependencies

This parameter appears only when you select the **Enable receive FIFO interrupt** option.

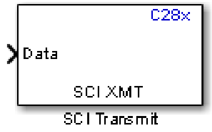
See Also

C28x SCI Transmit | C28x Hardware Interrupt

C28x SCI Transmit

Transmit data from target via serial communication interface (SCI) to host

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2838x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x



Description

The SCI Transmit block transmits scalar or vector data using the specified SCI hardware module. The sampling rate and data type are inherited from the input port.

A model can only contain one SCI Receive block for each module. The C28x processor has three SCI modules — A, B, and C. You can configure the SCI modules by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Note Fixed-point inputs are not supported by this block, but you can use a Data Type Conversion block to convert the fixed-point format to native data type. In the Data Type Conversion block, set the **Input and output to have equal** parameter to `Stored Integer (SI)`.

Input/Output Ports

Input

`Data` — Data written to serial bus
scalar | vector

Input data written to the serial communication bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single`

Parameters

`SCI module` — SCI module for communication
`A` (default) | `B` | `C` | `D`

The SCI module used for communication.

`Additional package header` — Indicates start of data
`'S'` (default) | string | char | number from 0 to 255

The data located at the beginning of the sent data package, which is not part of the data being transmitted, and indicates the start of data. The additional package header must be represented using ASCII characters. Use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not sent or included in the total byte count. To specify a null value (no package header), enter two single quotes only.

Note Match additional package headers or terminators with those specified in the host SCI Receive block.

`Additional package terminator` — Indicates end of data
`'E'` (default) | string | char | number from 0 to 255

The data located at the end of the sent data package, which is not part of the data being transmitted, and indicates the end of data. The additional package terminator must be represented using ASCII characters. Use a string or a number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent or included in the total byte count. To specify a null value (no package terminator), enter two single quotes only.

`Enable transmit FIFO interrupt` — Posts interrupt when FIFO is full
`off` (default) | `on`

If this option is selected, an interrupt is posted when the FIFO is full, allowing the subsystem to perform any action. For example, you can use the C28x Hardware Interrupt block for triggering the SCI Transmit block to write data as soon as the FIFO is available for transmitting data.

If the option is not selected, the SCI Transmit block is in polling mode and checks if the FIFO is available. If the FIFO is not full, the block writes data into the FIFO. If the FIFO is full, in blocking mode, the block waits until the FIFO is available. In non-blocking mode, the block continues with the execution of the algorithm without waiting for the availabitiy of the FIFO.

`Receive FIFO interrupt level (maximum 4 for Piccolo devices)` — Level for triggering interrupt
`1` (default) | `integer in the range [1 16]`

The receive FIFO generates an interrupt when the number of data bytes in the receive FIFO is less than or equal to the value selected for this parameter.

Dependencies

This parameter appears only when the **Enable transmit FIFO interrupt** option is selected.

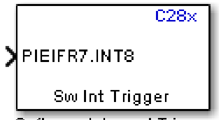
See Also

C28x SCI Receive | C28x Hardware Interrupt

C28x Software Interrupt Trigger

Generate software-triggered nonmaskable interrupt

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2838x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x



Software Interrupt Trigger

Description

When you add the Software Interrupt Trigger block to a model, the block polls the values on the input port. When the input value is greater than the value in the **Trigger software interrupt when input value is greater than** parameter, the block posts the interrupt corresponding to the selected CPU and Peripheral Interrupt Expansion (PIE) numbers to the Hardware Interrupt block in the model.

To use this block, add a Hardware Interrupt block to your model. The Hardware Interrupt block processes the software-triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set in the Software Interrupt Trigger block.

The CPU and PIE interrupt numbers together specify a single interrupt for a single peripheral module. For information about the mapping of CPU and PIE interrupt numbers to these peripheral interrupts, see C28x Hardware Interrupt.

Note Fixed-point inputs are not supported by the Software Interrupt Trigger block.

Input/Output Ports

Input

`PIEIFRx.INTy` — Triggers software interrupt
scalar

The Software Interrupt Trigger block triggers the software interrupt based on the **CUP interrupt number** and **PIE interrupt number** parameters when the input value is greater than the value in the **Trigger software interrupt when input value is greater than** parameter.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

`CPU interrupt number` — CPU interrupt number corresponding to hardware interrupt
7 (default) | scalar

Enter an integer value to set the CPU interrupt number corresponding to the hardware interrupt. For information about CPU numbers of C2000 processors, see C28x Hardware Interrupt.

`PIE interrupt number` — PIE interrupt number corresponding to hardware interrupt
8 (default) | scalar

Enter an integer value to set the PIE number corresponding to the hardware interrupt. For information about PIE numbers of C2000 processors, see C28x Hardware Interrupt.

`Trigger software interrupt when input value is greater than` — Sets value above which block posts an interrupt
0 (default) | scalar

Enter the value for the level that indicates that the interrupt is asserted by a requesting routine.

See Also

C28x Hardware Interrupt

C28x SPI Master Transfer

Write data to and read data from SPI slave device

Library:	Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

c28x
Tx
SPI
Master Transfer
Rx
Slave select: SPISTE

Description

The C28x SPI Master Transfer block writes data to and reads data from a slave device over the Serial Peripheral Interface (SPI). The block runs in master mode. The block outputs an array of the same size and data type as the input values. You can use this block with the Byte Pack and Byte Unpack blocks for heterogeneous data type transfers.

Configure the SPI modules for the specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Using this block, you can access an SPI device to measure quantities such as temperature and pressure.

Ports

Input

Tx — Data written to registers of SPI slave device (MOSI)
vector

The data written by the block to the registers of a slave device over the SPI interface.

Data Types: `uint16`

Output

Rx — Data read from registers of SPI slave device (MISO)
vector

The data read by the block from the registers of a slave device over the SPI interface.

Data Types: `uint16`

Parameters

Main

`SPI module` — SPI module to write and read data
`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI peripheral module to which the SPI slave device is connected. Each processor has a different number of modules.

`Clock polarity` — SPI clock polarity
`Rising_edge` (default) | `Falling_edge`

The clock polarity (CPOL) for SPI communication mode.

`Clock phase` — SPI clock phase
`No_delay` (default) | `Delay_half_cycle`

The clock phase (CPHA) for SPI communication mode.

`Enable register address` — Enables SPI register address
`on` (default) | `off`

Enables the **Register address** parameter.

`Register address` — SPI register address
`0` (default) | postive integer scalar | postive integer vector

The slave register address from which the block reads data.

Dependencies

This parameter appears only when you select **Enable register address**.

Advanced

`Data bits` — Number of bits in SPI transfer
8 (default) | integer in the range [1 16]

Length in bits of each transmitted or received character, specified as an integer in [1 16]. For example, if you select 8, the maximum value that can be transmitted using SPI is 2^8 – 1. If you send data values greater than this value, the buffer overflows.

`Slave select calling method` — Method to select SPI slave device

`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI master uses these methods to select SPI slave devices.

- `Provided by the SPI peripheral` — The SPI master uses the **STE pin assignment** parameter in **Hardware Implementation** > **Target Hardware Resources** > **SPI** to select the slave device. Slave select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI master uses the general purpose input/output pins explicitly to select/deselect SPI slave devices. The SPI Master Transfer block selects the slave before data is transmitted and deselects the slave after data is received using GPIO pins.

`Slave select pin polarity` — SPI slave select pin polarity

`Active low` (default) | `Active high`

The logic levels supported by the slave select pin to select the SPI slave device.

- `Active low` — The device is enabled on logic low. The SPI slave device is enabled when its slave select pin is set to low.
- `Active high` — The device is enabled on logic high. The SPI slave device is enabled when its slave select pin is set to high.

Dependencies

This parameter appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

`Slave select pin` — SPI slave select pin

`1` (default) | postive integer scalar

The general purpose input/output pin that serves as slave select for SPI.

Dependencies

This parameter appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

See Also

C28x SPI Receive | C28x SPI Transmit | C28x Hardware Interrupt

Introduced in R2017b

C28x SPI Receive

Receive data through Serial Peripheral Interface (SPI) on target

c28x
Rx
SPI
Receive
Status
Slave select: SPISTE

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The SPI Receive block supports synchronous, serial peripheral input/output port communications between the processor and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data, and the SPISOMI pin receives the data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum frequency for the clock is one quarter of the processor clock frequency.

The SPI device receives data and places the data in the receive buffer. The SPI Receive block reads the data from the receive buffer. In master mode, the C28x SPI Transmit block initiates SPI transmission by writing data to the transmit buffer. Then, the data received in the receive buffer is read by the SPI Receive block. In slave mode, the SPI Receive block is used to read the data in the receive buffer, which is received from the master. Then, the data is written into the transmit buffer using the SPI Transmit block. From the transmit buffer, the data is sent to the master.

Configure the SPI modules for a specific hardware board by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Output

Rx — SPI receive data
vector

The data read from the device over the SPI interface.

Data Types: `uint16`

`Status` — SPI receive status
`0` | `1` | `2`

Status of receipt of data. Error status values indicate:

- 0 — No errors.
- 1 — Data loss occurred because of overflow.
- 2 — Data not ready. A time out occurred while the block was waiting to receive data.

Dependencies

This port appears only when **Enable blocking mode** is not selected.

Data Types: `uint16`

Parameters

Main

`SPI module` — SPI module to read data
`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI module to which the SPI slave device is connected. Each processor has a different number of modules.

`Clock polarity` — SPI clock polarity
`Rising_edge` (default) | `Falling_edge`

The clock polarity used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

`Clock phase` — SPI clock phase
`No_delay` (default) | `Delay_half_cycle`

The clock phase used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

`Output data length` — SPI output data length
`1` (default) | positive integer

The received data is a vector of type `uint16` and the data length is as specified in this parameter (not bytes).

`Enable blocking mode` — Enable SPI blocking mode
`off` (default) | `on`

When this option is selected, the algorithm waits until data is received before continuing processing.

`Sample time` — SPI sample time selection

`0.1` (default) | `–1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Advanced

`Data bits` — Number of bits in SPI transfer

`8` (default) | integer in [1 16]

Length in bits of each transmitted or received character. For example, if you select `8`, the maximum value that can be transmitted using SPI is $2^8–1$. If you send data greater than this value, the buffer overflows. This parameter must be the same for both transmit and receive blocks.

`Slave select calling method` — Method to select SPI slave device

`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI master uses these methods to select SPI slave devices:

- `Provided by the SPI peripheral` — The SPI master uses the STE pin assignment provided in **Hardware Implementation** > **Target hardware resources** > **SPI** to select the slave device. Slave select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI master uses the general purpose input/output pins instead of the STE pin of the SPI peripheral to select/deselect SPI slave devices. The SPI Receive block deselects the slave using GPIO pins after receiving data. To select the slave, the C28x SPI Transmit block must be used along with the SPI Receive block. Use this option only in master mode. Select the **Enable blocking mode** option to ensure that the SPI transmission is complete before the slave is deselected.

`Slave select pin polarity` — SPI slave select pin polarity

`Active low` (default) | `Active high`

The logic levels supported by the slave select pin to select the SPI slave device.

- `Active low` — The device is enabled on logic low. The SPI slave device is enabled when its slave select pin is set to low.

- `Active high` — The device is enabled on logic high. The SPI slave device is enabled when its slave select pin is set to high.

Dependencies

This option appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

`Slave select pin` — SPI slave select pin
`0` (default) | postive integer scalar

The general purpose input/output pin that serves as the slave select for SPI.

Dependencies

This option appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

See Also

C28x SPI Transmit | C28x SPI Master Transfer | C28x Hardware Interrupt

Introduced in R2017b

C28x SPI Transmit

Transmit data through Serial Peripheral Interface (SPI) on target

c28x
Tx
SPI
Transmit
Status
Slave select: SPISTE

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The SPI Transmit block supports synchronous, serial peripheral input/output port communications between the processor and external peripherals or other controllers. The block can run in either slave or master mode. In master mode, the SPISIMO pin transmits data, and the SPISOMI pin receives the data. When master mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both master and slave to send and receive data simultaneously. The maximum frequency for the clock is one quarter of the processor clock frequency.

The SPI Transmit block writes data to the transmit buffer, and the data is transmitted to the SPI device. In master mode, the SPI Transmit block initiates SPI transmission by writing the data to the SPI transmit buffer. The C28x SPI Receive block must be used along with the SPI Transmit block to read the data received in the receive buffer. In slave mode, the SPI Receive block is used to read the data in the receive buffer, which is received from the master. Then, the data is written into the transmit buffer using the SPI Transmit block. From the transmit buffer, the data is sent to the master.

The sampling rate is inherited from the input port. The supported data type is `uint16`.

When the SPI transmit interrupt is configured, the transmit FIFO interrupt flags are cleared in the step function instead of the interrupt service routine. After the data is placed in the transmit buffer, the transmit FIFO interrupt is set and the previous transmit interrupt FIFO flags are cleared. Configure the SPI modules for a specific hardware board by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Input

Tx — SPI Transmit data

vector

The data written to the device over the SPI interface.

Data Types: `uint16`

Output

`Status` — SPI transmit status

0 | 1 | 2

Status of SPI data transmission. Error status values indicate:

- 0 — No errors.
- 1 — A time-out occurred while the block was transmitting data.
- 2 — The transmitted data contains an error.

Dependencies

This port appears only when **Enable blocking mode** is not selected.

Data Types: `uint16`

Parameters

Main

`SPI module` — SPI module to write data

`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI module to which the SPI slave device is connected. Each processor has a different number of modules.

`Clock polarity` — SPI clock polarity

`Rising_edge` (default) | `Falling_edge`

The clock polarity used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

`Clock phase` — SPI clock phase

`No_delay` (default) | `Delay_half_cycle`

The clock phase used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

`Enable blocking mode` — SPI blocking mode enable
`off` (default) | `on`

When this option is selected, the algorithm waits until data is sent before continuing processing.

Advanced

`Data bits` — Number of bits in SPI transfer
8 (default) | integer in [1 16]

Length in bits of each transmitted or received character. For example, if you select `8`, the maximum value that can be transmitted using SPI is 2^8–1. If you send data greater than this value, the buffer overflows. This parameter must be the same for both transmit and receive blocks.

`Slave select calling method` — Method to select SPI slave device
`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI master uses these methods to select SPI slave devices:

- `Provided by the SPI peripheral` — The SPI master uses the STE pin assignment provided in **Hardware Implementation** > **Target hardware resources** > **SPI** to select the slave device. Slave select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI master uses the general purpose input/output pins instead of the STE pin of the SPI peripheral to select/deselect SPI slave devices. The SPI Transmit block selects the slave using GPIO pins before transmitting data. To deselect the slave, you must use the C28x SPI Receive block along with the SPI Transmit block. Use this option only in master mode. Select the **Enable blocking mode** option to ensure that the SPI transmission is complete before the slave is deselected.

`Slave select pin polarity` — SPI slave select pin polarity
`Active low` (default) | `Active high`

The logic levels supported by slave select pin to select the SPI slave device.

- `Active low` — The device is enabled on logic low. The SPI slave device is enabled when its slave select pin is set to low.

- `Active high` — The device is enabled on logic high. The SPI slave device is enabled when its slave select pin is set to high.

Dependencies

This option appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

`Slave select pin` — SPI slave select pin

`0` (default) | postive integer scalar

The general purpose input/output pin that serves as the slave select for SPI.

Dependencies

This option appears only when **Slave select calling method** is set to `Explicit GPIO calls`.

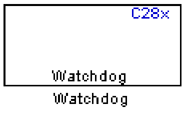
See Also

C28x SPI Receive | C28x SPI Master Transfer | C28x Hardware Interrupt

Introduced in R2017b

C28x Watchdog

Configure counter reset source of processor watchdog module



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2802x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x

Description

This block configures the counter reset source of the watchdog module on the processor. The watchdog module, after configuration, resets the system if not serviced periodically.

Ports

Input

`Input` — Resets watchdog counter
scalar

When the input signal is `1`, the counter is reset.

Dependencies

This parameter appears only when **Watchdog counter reset source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

`Watchdog counter reset source` — Watchdog counter reset source
`Specify via dialog` (default) | `Input port`

The watchdog counter reset source.

- `Input` — Create an input port on the watchdog block. When the input signal is `1`, the counter is reset.
- `Specify via dialog` — The watchdog timer is reset based on the **Sample time** value.

`Sample time` — Frequency at which processor resets Watchdog timer
–1 (default) | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Dependencies

This parameter appears only when **Watchdog counter reset source** is set to `Specify via dialog`.

See Also

C28x Hardware Interrupt

C28x CAN Calibration Protocol

Implement CAN Calibration Protocol (CCP)

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x

Description

The CAN Calibration Protocol block provides an implementation of a subset of the Controller Area Network (CAN) calibration protocol (CCP) version 2.1. CCP is a protocol for communicating between the target processor and the host machine over a CAN. A calibration tool (see “Compatibility with Calibration Packages” on page 2-214) running on the host can communicate with the target, allowing remote signal monitoring and parameter tuning.

This block processes a command receive object (CRO) and outputs the resulting data transmission object (DTO) and data acquisition (DAQ) messages.

For more information about CCP, refer to *ASAM Standards: ASAM MCD: MCD 1a* at the Association for Standardization of Automation and Measuring Systems (ASAM) website: `https://www.asam.net`.

Using the DAQ Output

Note The CCP DAQ list mode of operation is only supported with Embedded Coder. If Embedded Coder is not available, then custom storage classes `canlib.signal` are ignored during code generation, which means that the CCP DAQ list mode of operation cannot be used.

You can use the CCP polling mode of operation with or without Embedded Coder.

The DAQ output is the output for CCP DAQ lists that have been set up. You can use the ASAP2 file generation feature of the real-time target to:

- Set up signals to be transmitted using CCP DAQ lists.
- Assign signals in your model to a CCP event channel automatically (see “Generate an ASAP2 File” (Simulink Coder)).

After these signals are set up, event channels periodically fire events that trigger the transmission of DAQ data to the host. When this occurs, CAN messages with the CCP/DAQ data appear in the DAQ output, along with an associated function call trigger.

The calibration tool (see “Compatibility with Calibration Packages” on page 2-214) must use CCP commands to assign an event channel and data to the available DAQ lists. Based on the assignment, the calibration tool interprets the synchronous response.

Using DAQ lists for signal monitoring has these advantages over the polling method:

- The host does not need to poll for the data. Network traffic is halved.
- The data is transmitted at the update rate that matches the signal, reducing network traffic.
- Data is consistent. The transmission takes place after the signals have been updated, reducing interruptions while sampling the signal.

Parameters

`CCP station address (16-bit integer)` — Station address of target
`hex2dec('1')` (default) | 16-bit integer

The station address is interpreted as a `uint16` data type. It is used to distinguish between different targets. By assigning unique station addresses to targets sharing the same CAN bus, a single host can communicate with multiple targets.

`CAN module` — CAN module the block configures
`eCAN_A` (default) | `eCAN_B`

If your processor has more than one module, select the module this block configures.

`CAN message identifier (CRO)` — CAN message identifier of command receive object
`hex2dec('6FA')` (default) | 11-bit integer | 29-bit integer

The CAN message identifier of the command receive object (CRO) message you want to process.

`CAN message type (CRO)` — Incoming message type of command receive object
`Extended(29-bit identifier)` (default) | `Standard(11-bit identifier)`

The incoming message type of the command receive object.

`CAN message identifier (DTO/DAQ)` — CAN message ID used for DTO and DAQ
`hex2dec('6FB')` (default) | 11-bit integer | 29-bit integer

The CAN message ID used for DTO and DAQ message outputs.

`CAN message type (DTO/DAQ)` — Message type to be transmitted by DTO and DAQ
`Extended(29-bit identifier)` (default) | `Standard(11-bit identifier)`

The message type to be transmitted by DTO and DAQ outputs.

`Total Number of Object Descriptor Tables (ODTs)` — Number of ODTs based on number of signals logged
8 (default) | integer in the range [0, 254]

ODTs are shared equally between all available DAQ lists. You can choose a value in the range of 0 to 254, depending on the number of signals you log simultaneously. You must allocate at least one ODT per DAQ list, or your build may fail. The calibration tool gives an error message if there are too few ODTs for the number of signals you specify for monitoring. However, too many ODTs can make the sample time overrun. If you choose more than the maximum number of ODTs (254), the build fails.

A single ODT uses 7 bytes of memory. Using all 254 ODTs requires 1778 bytes or more of memory, a large proportion of the available memory on the target. To conserve memory on the target, the default number is low, allowing DAQ list signal monitoring with reduced memory overhead and processing power.

As an example, if you have five different rates in a model, and you are using three rates for DAQ, this creates three DAQ lists. You must ensure that you have at least three ODTs. ODTs are shared equally among DAQ lists, and therefore, you end up with one ODT per DAQ list. With less than three ODTs for three DAQ lists, the behavior is undefined.

Taking this example further, say you have three DAQ lists with one ODT each, and you are monitoring signals in a calibration tool. If you try to assign too many signals to a particular DAQ list, thereby requiring more space than 7 bytes, the calibration tool reports an error.

`CRO sample time` — Sample time for CRO messages
`0.1` (default) | `-1` | scalar

The sample time for CRO messages. To execute this block asynchronously, set this parameter to `-1`.

More About

Supported CCP Commands

These CCP commands are supported by the CAN Calibration Protocol block:

- `CONNECT`
- `DISCONNECT`
- `DNLOAD`
- `DNLOAD_6`
- `EXCHANGE_ID`
- `GET_CCP_VERSION`
- `GET_DAQ_SIZE`
- `GET_S_STATUS`
- `SET_DAQ_PTR`

- `SET_MTA`
- `SET_S_STATUS`
- `SHORT_UP`
- `START_STOP`
- `START_STOP_all`
- `TEST`
- `UPLOAD`
- `WRITE_DAQ`

Compatibility with Calibration Packages

The supported CCP commands are compatible with:

- Synchronous signal monitoring via calibration packages that use DAQ lists
- Asynchronous signal monitoring via calibration packages that poll the target
- Asynchronous parameter tuning via CCP memory programming

The CCP implementation has been tested with Vector Informatik CANape calibration package running in both DAQ list and polling mode.

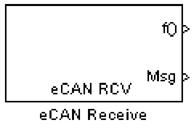
See Also

"Parameter Tuning and Signal Logging over Serial Communication" | "Set Up CAN Communication with Target Hardware"

C28x eCAN Receive

Enhanced Controller Area Network receive mailbox

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x



Description

The eCAN Receive block generates source code for receiving enhanced Controller Area Network (eCAN) messages through an eCAN mailbox. eCAN modules on the processor provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The block supports eCAN data frames in standard or extended format.

To use the eCAN Receive block with the eCAN Pack block in the `canmsglib` library, set **Data type** to `CAN_MESSAGE_TYPE`.

Configure the eCAN modules for a specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Output

`f()` — Function call port
scalar

Connect a function call subsystem to this port. When a new message is received, the subsystem is executed.

`Msg` — Message data port
vector

The received data is output in the form of a vector of elements of the selected data type. The length of the vector is 8 bytes. When the block is used in polling mode, and a new message is not created between consecutive executions of the block, the existing message is repeated.

To use the eCAN Receive block with the CAN Unpack block in the `canmsglib` library, set **Data type** to `CAN_MESSAGE_TYPE`.

Data Types: `uint8` | `uint16` | `uint32` | `CAN_MESSAGE_TYPE`

`len` — Length of output message
scalar

The length of output message received.

Dependencies

This port appears only if the **Output message length** parameter is selected.

Data Types: `uint16`

Parameters

`Module` — eCAN module the block configures
`eCAN_A` (default) | `eCAN_B`

Determines the eCAN module configured by the eCAN Receive block.

`Mailbox number(0–31)` — Sets value of mailbox number register (MBNR)
`0` (default) | integer in the range [0 31]

For standard CAN controller (SCC) mode, enter a unique number from `0` to `15`. For high-end CAN controller (HECC) mode, enter a unique number from `0` to `31` . In SCC mode, transmissions from the mailbox with the highest number have the highest priority. In HECC mode, the mailbox number only determines priority if the transmit priority level (TPL) of two mailboxes is equal.

`Message identifier` — Sets value of message identifier register (MID)
bin2dec('111000111') (default) | numeric identifier of length 11 or 29 bits

The message identifier is 11 bits long for the standard frame size or 29 bits long for the extended frame size in decimal, binary, or hex format. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry.

`Message type` — Message identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

The message identifier type.

`Sample time` — Frequency at which mailbox is polled for new message
`1` (default) | `-1` | scalar

A new message causes a function call to be sent from the mailbox. If you want to update the message output only when a new message arrives, the block needs to be executed asynchronously. To execute the block asynchronously, set this parameter to `-1`, and select the **Post interrupt when message is received** option.

Note For information about setting the timing parameters of the CAN module, see “Configuring Timing Parameters for CAN Blocks”.

Data type — Output message data type

uint16 (default) | uint8 | uint32 | CAN_MESSAGE_TYPE

The options available are:

- `uint8`: Vector length = 8 elements
- `uint16`: Vector length = 4 elements
- `uint32`: Vector length = 2 elements
- `CAN_MESSAGE_TYPE`: Outputs data as a structure. Use the CAN Unpack block to extract the data from the structure.

The length of the vector for the received message is at most 8 bytes. If the message is less than 8 bytes, the data buffer bytes are right-aligned in the output. The data are unpacked as follows using the data buffer, which is 8 bytes.

For `uint8` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 8-bit data to eight elements (using the lower part of the 16-bit memory).

```
Output[0] = data_buffer[0];
Output[1] = data_buffer[1];
Output[2] = data_buffer[2];
Output[3] = data_buffer[3];
Output[4] = data_buffer[4];
Output[5] = data_buffer[5];
Output[6] = data_buffer[6];
Output[7] = data_buffer[7];
```

For `uint16` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 16-bit data to four elements.

```
Output[0] = data_buffer[1..0];
Output[1] = data_buffer[3..2];
Output[2] = data_buffer[5..4];
Output[3] = data_buffer[7..6];
```

For `uint32` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 32-bit data to two elements.

```
Output[0] = data_buffer[3..0];
Output[1] = data_buffer[7..4];
```

For example, if the received message has two bytes:

```
data_buffer[0] = 0x21
data_buffer[1] = 0x43
```

The `uint16` output is:

```
Output[0] = 0x4321
Output[1] = 0x0000
Output[2] = 0x0000
Output[3] = 0x0000
```

When you select `CAN_MESSAGE_TYPE`, the block outputs the following struct data (defined in `can_message.h`):

```
struct {

  /* Is Extended frame */
  uint8_T Extended;

  /* Length */
  uint8_T Length;

  /* RTR */
  uint8_T Remote;

  /* Error */
  uint8_T Error;

  /* CAN ID */
  uint32_T ID;

  /*
  TIMESTAMP_NOT_REQUIRED is a macro that will be defined by Target teams
  PIL, xPC if they do not require the timestamp field during code
  generation. By default, timestamp is defined. If the targets do not require
  the timestamp field, they should define the macro TIMESTAMP_NOT_REQUIRED before
  including this header file for code generation.
  */
  #ifndef TIMESTAMP_NOT_REQUIRED
    /* Timestamp */
    double Timestamp;
  #endif

  /* Data field */
  uint8_T Data[8];

};
```

`Initial output` — Sets output before receiving data

`0` (default) | integer

The value of the output before receiving data.

`Output message length` — Output message length in bytes

`off` (default) | `on`

The message length in bytes, sent to the **len** port. If not selected, the **len** port is not visible.

`Post interrupt when message is received` — Posts asynchronous interrupt when message is received

`off` (default) | `on`

When selected, the block posts an asynchronous interrupt when a message is received.

`Interrupt line` — Interrupt line of asynchronous interrupt

`0` (default) | `1`

The interrupt line the asynchronous interrupt uses. The value of this parameter sets bit 2 (GIL) in the global interrupt mask register (CANGIM):

- `1` maps the global interrupts to the ECAN1INT line.
- `0` maps the global interrupts to the ECAN0INT line.

Dependencies

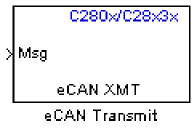
This parameter appears only when you select **Post interrupt when message is received**.

See Also

C28x eCAN Transmit | C28x Hardware Interrupt

C28x eCAN Transmit

Enhanced Controller Area Network transmit mailbox



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C281x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x

Description

The eCAN Transmit block generates source code for transmitting enhanced Controller Area Network (eCAN) messages through an eCAN mailbox. eCAN modules on the processor provide serial communication capability and have 32 mailboxes configurable for receive or transmit. This block supports eCAN data frames in standard or extended format.

Note Fixed-point inputs are not supported by this block.

Configure the eCAN modules for a specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Input

`Msg` — Message data
vector

Input message data.

Data Types: `uint8` | `uint16` | `uint32` | `CAN_MESSAGE_TYPE`

Parameters

`Module` — eCAN module the block configures
`eCAN_A` (default) | `eCAN_B`

Determines the eCAN module configured by this instance of the eCAN Transmit block.

`Mailbox number(0–31)` — Sets value of mailbox number register (MBNR)
`1` (default) | integer in the range [0 31]

A unique number from 0 to 15 for standard or from 0 to 31 for enhanced CAN mode. The number refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

`Message identifier` — Value of message identifier register (MID)
bin2dec('111000111') (default) | numeric identifier of length 11 or 29 bits

The message identifier is 11 bits long for the standard frame size or 29 bits long for the extended frame size in decimal, binary, or hex. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

Note CAN messages use the value of the message identifier parameter in the C28x CAN Transmit block for transmission even when you use the CAN Pack block to create the CAN message.

`Message type` — Message identifier type

`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

The message identifier type.

`Enable blocking mode` — Sets blocking mode

`off` (default) | `on`

If selected, the CAN block waits indefinitely for a transmit (XMT) acknowledgment. If not selected, the CAN block does not wait for a transmit (XMT) acknowledgment, which is useful if the hardware fails to acknowledge transmissions.

`Post interrupt when message is transmitted` — Posts asynchronous interrupt when message is transmitted

`off` (default) | `on`

When selected, this block posts an asynchronous interrupt when data is transmitted.

`Interrupt Line` — Interrupt line of asynchronous interrupt

`0` (default) | `1`

The interrupt line the asynchronous interrupt uses. The value of this parameter sets bit 2 (GIL) in the global interrupt mask register (CANGIM):

- `1` maps the global interrupts to the ECAN1INT line.
- `0` maps the global interrupts to the ECAN0INT line.

Note For information about setting the timing parameters of the CAN module, see "Configuring Timing Parameters for CAN Blocks".

Dependencies

This parameter appears only when **Post interrupt when message is transmitted** is selected.

More About

Data Vectors

The length of the vector for each transmitted mailbox message is 8 bytes. Input data are right-aligned in the message data buffer. The `uint8` (vector length = 8 elements), `uint16` (vector length = 4 elements), and `uint32` (vector length = 2 elements) data types are accepted. While using the eCAN Transmit block with the CAN Pack block in the `canmsglib` library, `CAN_MESSAGE_TYPE` is also accepted.

The following examples show how different types of input data are aligned in the data buffer:

For input of data type `uint32`,

```
inputdata [0] = 0x12345678
```

the data buffer is:

```
data buffer[0] = 0x78
data buffer[1] = 0x56
data buffer[2] = 0x34
data buffer[3] = 0x12
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

For input of data type `uint16`,

```
inputdata [0] = 0x1234
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x00
data buffer[3] = 0x00
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

For input of data type `uint16[2]`, which is a two-element vector,

```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x78
data buffer[3] = 0x56
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

See Also

C28x eCAN Receive | C28x Hardware Interrupt

C28x eQEP

Quadrature encoder pulse block used to derive position, direction, and speed

C28x
qposcnt
eQEP
eQEP

Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / C2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C280x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2833x
Embedded Coder Support Package for Texas Instruments C2000 Processors / C2834x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M35x / C28x
Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors / F28M36x / C28x

Description

The enhanced quadrature encoder pulse (eQEP) block is used along with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine.

The eQEP peripheral module inputs include QEPA, QEPB, QEPI (index), and QEPS (strobe).

To configure your device to work with the block, navigate to **Model Configuration Parameters** > **Hardware Implementation**, select your device at **Hardware board**, and expand **Target hardware resources**.

Input/Output Ports

Input

`swi` — Dynamically update initialization value for position counter

scalar

If the input is `true`, the position counter is initialized to the value in the **Initialization value (0~4294967295)** on page 2-0 parameter.

Dependencies

This port appears only when, in the **Position counter** tab, you select **Enable software initialization** and set **Software initialization source** to `Input port`.

Data Types: `Boolean`

`cmp` — Value for comparing position

scalar

Input value that generates the position compare sync signal.

Dependencies

This port appears only when, in the **Compare output** tab, you select **Enable position-compare sync signal output** and set **Compare value source** to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`iel` — Software index marker

scalar

Software index event marker for latching the position counter.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.
- You set **Index event latch of position counter** to `Software index marker via input port`.

Note In **Compare output** tab, if **Sync output pin selection** is set to `Index pin is used for sync output`, then the **Index event latch of position counter** parameter cannot be set to `Software index marker via input port`.

Data Types: `Boolean`

Output

`qposcnt` — Position counter signal

scalar

Position counter signal (PCSOUT) received from the position counter and control unit (PCCU).

Dependencies

This port appears only when, in the **Position counter** tab, you select **Output position counter**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`pcef` — Position counter error flag on error

scalar

Outputs the position counter error flag on an error.

- `0` — No error occurred during the last index transition.
- `1` — Position counter error.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on an index event`.
- You select **Output position counter error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qdf` — Direction flag of quadrature module
scalar

Direction flag of the quadrature module.

- `0` — counterclockwise rotation (or reverse movement).
- `1` — clockwise rotation (or forward movement).

Dependencies

This port appears only when, in the **General** tab, you select **Quadrature direction flag output port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qctmr` — Capture timer signal
scalar

Outputs the eQEP capture timer signal.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qcprd` — Capture period signal
scalar

Outputs the capture period signal, which holds the period count value between the last successive eQEP position events.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture period timer**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`coef` — eQEP overflow error flag
scalar

Outputs overflow error flag (COEF flag) in the event of capture timer overflow between unit position events.

- `0` — Overflow has not occurred.
- `1` — Overflow occurred in the eQEP capture timer register (QEPCTMR).

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Enable and output overflow error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`cdef` — Direction change error flag
scalar

Outputs the direction change error flag.

- `0` — Capture direction error has not occurred.
- `1` — Direction change occurred between two capture position events.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Enable and output direction change error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qctmrlat` — Capture timer latched value
scalar

Outputs the capture timer latched value from the QCTMRLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qcprdlat` — Capture timer period latched value
scalar

Outputs the capture timer period latched value from the QCPRDLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer period latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qposlat` — Position counter latched value
scalar

Outputs position counter latched value from the QPOSLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output position counter latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qposilat` — Latches position counter on index event
scalar

eQEP index input can be configured to latch the position counter register (QPOSCNT) as output on the occurrence of a definite event on this pin.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

`qposslat` — Latches position counter on strobe event
scalar

eQEP strobe input can be configured to latch the position counter register (QPOSCNT) as output on the occurrence of a definite event on this pin.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on strobe event**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

General

`Module` — eQEP module to obtain position, direction, and speed
`eQEP1` (default) | `eQEP2` | `eQEP3`

The eQEP peripheral module used to obtain position, direction, and speed information. The number of eQEP modules supported varies for different C2000 processors.

`Position counter mode` — Mode that matches how input to eQEP peripheral is encoded
`Quadrature-count` (default) | `Direction-count` | `Up-count` | `Down-count`

The eQEP peripheral inputs are QEPA, QEPB, QEPI, and QEPS. Configure the GPIO pins for these inputs by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources > eQEP**.

Input signals QEPA and QEPB to the eQEP peripheral are processed by the quadrature decoder unit (QDU) in the eQEP peripheral to produce clock (QCLK) and direction (QDIR) signals. Choose the position counter mode that matches the way the input to the eQEP module is encoded:

- `Quadrature-count` — Two square signals (A and B) 90 degrees out of phase are sent to the eQEP peripheral. The QDU uses the phase relationship of these two inputs to generate quadrature clock and direction signals.
- `Direction-count` — Direction and clock signals are directly sent to the eQEP peripheral. The QEPA pin provides the clock input, and the QEPB pin provides the direction input.
- `Up-count` — The position counter is used to measure the frequency of the signal in the QEPA pin. The direction is hard-wired for up count in this mode.
- `Down-count` — The position counter is used to measure the frequency of the signal in the QEPA pin. The direction is hard-wired for down count in this mode.

`Positive rotation` — Direction of rotation

`Clockwise` (default) | `Counterclockwise`

Set the direction of rotation.

If `Clockwise` is selected, the quadrature count operation is performed without swapping the quadrature clock inputs fed to the QDU.

If `Counterclockwise` is selected, reverse counting is performed by swapping the quadrature clock inputs fed to the QDU. The quadrature decoder reverses the counting direction.

Dependencies

This parameter appears only when you set **Position counter mode** to `Quadrature-count` on the **General** tab.

`External clock rate` — Measurement resolution

`2x resolution: Count the rising/falling edge` (default) | `1x resolution: Count the rising edge only`

Select the resolution of the clock generator that the position counter uses as input so that the counting occurs on both rising and falling edges of the QEPA input or on the rising edge only. Choosing the former increases the measurement resolution by a factor of 2.

Dependencies

This parameter appears only when you set **Position counter mode** to `Direction-count`, `Up-count`, or `Down-count`.

`Quadrature direction flag output port` — Creates port for direction flag
`off` (default) | `on`

Creates a port (qdf) on the block that gives the direction flag of the quadrature module.

Dependencies

This parameter appears only when, in the **General** tab, **Position counter mode** is set to `Quadrature-count`.

`Invert input QEPxX polarity` — Inverts polarity of eQEP input
`off` (default) | `on`

Inverts the polarity of the eQEP peripheral inputs. The **Invert input QEPxA polarity** checkbox corresponds to QEPA, **Invert input QEPxB polairty** corresponds to QEPB, and so on.

`Index pulse gating option` — Enables gating of index pulse with strobe input
`off` (default) | `on`

Enables the gating of the peripheral input index signal with the peripheral input strobe signal. In this case, the internal index signal is high only when both the peripheral input signals eQEPxI and eQEPxS are high.

`Sample time` — Frequency at which block reads position counter
`0.0001` (default) | `-1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Position counter

`Output position counter` — Outputs position counter signal
on (default) | `off`

Outputs the position counter signal PCSOUT from the position counter and control unit (PCCU). The position counter register counts up or down on every eQEP pulse based on the direction of the input. The count value is proportional to the position from a given reference point.

`Maximum position counter value (0~4294967295)` — Specifies maximum position counter value
`4294967295` (default) | integer in [0, 4,294,967,295]

Enter a maximum value (QPOSMAX) for the position counter. If the position counter reaches QPOSMAX, the position counter is set to `0` on the next increment of the counter. If the position counter is `0`, the position counter is set to QPOSMAX on the next decrement of the counter.

`Enable set to init value on index event` — Enables option to set initialization value for position counter
`off` (default) | `on`

Enables option to set the position counter to its initialization value on an index event.

`Set to init value on index event` — Initialization value for position counter
`Rising edge` (default) | `Falling edge`

Sets the position counter to its initialization value on the rising edge or falling edge of the index event.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable set to init value on index event**.

`Enable set to init value on strobe event` — Enables option to set initialization value for position counter
`off` (default) | `on`

Enables option to set the position counter to its initialization value on a strobe event.

`Set to init value on strobe event` — Sets initialization value for position counter
`Rising edge` (default) | `Depending on direction`

The `Rising edge` option sets the position counter to its initialization value on the rising edge of the strobe input. The `Depending on direction` option sets the position counter to its initialization value on the:

- rising edge of the strobe input, in the forward direction.
- falling edge of the strobe input, in the reverse direction.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable set to init value on strobe event**.

`Enable software initialization` — Enables option to set initialization value for position counter

`off` (default) | `on`

Allows the position counter to be set to its initialization value using the software.

`Software initialization source` — Specifies initialization source of position counter

`Input port` (default) | `Set to init value at start up`

Choose the `Set to init value at start up` option to initialize the position counter to the value entered in **Initialization value** at the start of the execution of the program. Choose the `Input port` option to update the initialization value dynamically based on an input initialization signal (input port **swi**). If the input **swi** is `true`, the position counter is initialized to the **Initialization value**.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable software initialization**.

`Initialization value (0~4294967295)` — Initialization value for position counter

`2147483648` | integer in [0, 4,294,967,295]

Enter the initialization value for the position counter.

Dependencies

This parameter appears only when you select **Enable set to init value on index event**, **Enable set to init value on strobe event**, or **Enable software initialization**.

`Position counter reset mode` — Resets position counter
`Reset on an index event` (default) | `Reset on the maximum position` | `Reset on the first index event` | `Reset on a time unit event`

Position counter reset mode, depending on the nature of the system the eQEP module is working with.

- `Reset on an index event` — If the index event occurs during the forward direction, then the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse direction, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock.
- `Reset on the maximum position` — During the forward direction, when the position counter is equal to QPOSMAX, the position counter is reset to 0 on the next eQEP clock, and the position counter overflow flag is set. During the reverse direction, when the position counter is equal to 0, the position counter is reset to QPOSMAX on the next QEP clock, and the position counter underflow flag is set.
- `Reset on the first index event` — If the index event occurs during the forward direction, the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse direction, the position counter is reset to the value in the QPOSMAX register on the next eQEP clock. The position counter is reset using the `Reset on the first index event` option only on the first index event occurrence. After the first index event occurrence, the position counter is reset based on the maximum position.
- `Reset on a time unit event` — The QPOSCNT value is latched to the QPOSLAT register on a unit time event. The QPOSCNT register is then reset to 0 for the forward direction and QPOSMAX for the reverse direction. You can use this option for frequency measurement.

`Output position counter error flag` — Outputs position counter error flag on error
`off` (default) | `on`

Outputs the position counter error flag on error. When you select this option, the output port **pcef** is created.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on an index event`.

`Output latch position counter on index event` — Latches position counter on index event
`off` (default) | `on`

When this option is enabled, the position counter QPOSCNT latches into QPOSLAT on the occurrence of an event on the strobe pin.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.

`Index event latch of position counter` — Configures position counter to latch on an event
`Rising edge` (default) | `Falling edge` | `Software index marker via input port`

Configures the eQEP position counter to latch on the index event selected.

Dependencies

This parameter appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.

`Output latch position counter on strobe event` — Latches position counter on strobe event
`off` (default) | `on`

The eQEP strobe input can be configured to latch the position counter (QPOSCNT) into QPOSSLAT on occurrence of a definite event on this pin. This option latches the position counter on each strobe event.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.

`Strobe event of latched position counter` — Configures position counter to latch on strobe
`Rising edge` (default) | `Depending on direction`

`Rising edge` latches on the rising edge of the strobe event input. `Depending on direction` latches on the rising edge in the forward direction and the falling edge in the reverse direction.

Dependencies

This parameter appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on strobe event**.

Speed calculation

To view the other parameters of this tab, select the **Enable QEP capture** option.

`Enable QEP capture` — Enables edge capture unit
`off` (default) | `on`

The eQEP peripheral includes an integrated edge capture unit to measure the elapsed time between the unit position events. This option enables the edge capture unit.

`Output capture timer` — Outputs capture timer
`off` (default) | `on`

Outputs the capture timer value from the QCTMR register.

`Output capture period timer` — Outputs capture period
`off` (default) | `on`

Outputs the period count value between the last successive eQEP position events from the QCPRD register.

`eQEP capture timer prescaler` — Prescales capture timer clock frequency
`128` (default) | `1` | `2` | `4` | `8` | `16` | `32` | `64`

The eQEP capture timer runs from prescaled SYSCLKOUT. The capture timer clock frequency is the frequency of SYSCLKOUT divided by the value you choose for this parameter.

`Unit position event prescaler` — Prescales quadrature clock
`128` (default) | `1` | `2` | `4` | `8` | `16` | `32` | `64` | `256` | `512` | `1024` | `2048`

The timing of the unit position event is determined by prescaling the quadrature clock (QCLK). QCLK is divided by the prescalar value you choose for this parameter.

`Enable and output overflow error flag` — Outputs overflow error flag when capture timer overflows
`off` (default) | `on`

Enables and outputs the eQEP overflow error flag (COEF) in the event of capture timer overflow between unit position events.

`Enable and output direction change error flag` — Outputs direction change error flag
`off` (default) | `on`

Enables and outputs the direction change error flag (CDEF) when direction change occurs between the unit position events.

`Capture timer and position` — QEP capture latch mode
`On position counter read` (default) | `On unit time-out event`

Event that triggers the latching of the capture timer and capture period register:

- `On position counter read` — Latch on position counter read by the processor. The capture timer and capture period values are latched into the QCTMRLAT and QCPRDLAT registers when the processor reads the QPOSCNT register.
- `On unit time-out event` — Latch on unit time-out. The position counter, capture timer, and capture period values are latched into the QPOSLAT, QCTMRLAT, and QCPRDLAT registers on unit time-out.

`Unit timer period (0~4294967295)` — Sets unit timer period
`100000000` (default) | value in the range [0, 4,294,967,295]

Set the unit timer period.

Dependencies

This parameter appears only when you set **Capture timer and position** to `On unit time-out event`.

`Output capture timer latched value` — Outputs capture timer latched value
`off` (default) | `on`

Outputs the capture timer latched value from the QCTMRLAT register at the output port **qctmrlat**.

`Output capture timer period latched value` — Outputs capture timer period latched value
`off` (default) | `on`

Outputs the capture timer period latched value from the QCPRDLAT register at the output port **qcprdlat**.

`Output position counter latched value` — Outputs position counter latched value
`off` (default) | `on`

Outputs the position counter latched value from the QPOSLAT register at the output port **qposlat**.

Compare output

To view the other parameters of this tab, select the **Enable position-compare sync signal output** option.

`Enable position-compare sync signal output` — Enables position compare sync signal output
`off` (default) | `on`

The eQEP peripheral includes a position compare unit that generates the position compare sync signal when the position counter register (QPOSCNT) and the position compare register (QPOSCMP) values match. This option enables the position compare sync signal output. The sync signal can be output using an index pin or strobe pin of the eQEP peripheral.

`Sync output pin selection` — GPIO pin used for sync signal
`Index pin is used for sync output` (default) | `Strobe pin is used for sync output`

The GPIO pin used for the sync signal output. Use the index pin or strobe pin of the eQEP peripheral to output the position compare sync signal.

`Compare value source` — Source of value for position comparison
`Specify via dialog` (default) | `Input port`

Source of the value to be used for the position compare register (QPOSCMP). When this parameter is set to `Input port` the input port **cmp** is created.

`Position compare shadow load mode` — Shadow mode for generating position compare sync signal
`Load on QPOSCNT=0` (default) | `Shadow disabled(load immediate)` | `Load on QPOSCNT=QPOSCMP`

This parameter lets you enable or disable shadow mode for updating the position compare (QPOSCMP) register. When shadow mode is enabled, you can also choose an event to trigger the loading of the shadow register value into the active register. When shadow mode is disabled, the processor directly loads the value into the active register.

`Load on QPOSCNT=0` loads on a position counter zero event, and `Load on QPOSCNT=QPOSCMP` loads when the QPOSCNT and QPOSCMP values match. When you select these options, shadow mode is enabled.

`Position compare value (0~4294967295)` — Value for comparing postiton
`4294967295` (default) | value in the range [0, 4,294,967,295]

This value is loaded into the position compare register (QPOSCMP).

Dependencies

This parameter appears only when you set **Compare value source** to `Specify via dialog`.

`Sync output pulse width (1~4096)` — Pulse width of position compare sync output signal
`1` (default) | value in the range [0, 4,096]

The pulse stretcher logic in the position compare unit generates a programmable position compare sync pulse output on the position compare match.

A value from `1` to `4096` that determines the pulse width of the position compare sync output signal. The width of the output pulse, measured in SYSCKOUT cycles, is four times the entered value.

`Polarity of sync output` — Polarity of sync output
`Active high` (default) | `Active low`

Select the polarity of the sync output signal generated.

Watchdog unit

`Watchdog timer enable` — Enables watchdog time-out flag
`off` (default) | `on`

The eQEP peripheral contains a watchdog timer that monitors the quadrature clock to indicate that the motion-control system is operating. The timer is reset on an edge transition of the quadrature clock. The watchdog unit generates an interrupt, which you can enable in the **Interrupt** tab.

`Watchdog timer` — Time-out value for watchdog timer
`65535` (default) | value in the range [0, 65,535]

The time period after which the watchdog unit generates an interrupt.

Dependencies

This parameter appears only when you select **Watchdog timer enable**.

Signal Data Types

When you select signals as output in the other tabs, the corresponding signals appear in this tab. For example, when you select the **Output position counter** option on the **Position counter** tab, the **Position counter value data type** option appears on this tab. Using this tab, you can select the data types of the signals.

The valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean`.

The following table summarizes the options for which you can set the data type in the **Signal data types** tab:

Pane	Option
General	Quadrature direction flag output port
Position counter	Output position counter (selected by default)
	Output position counter error flag
	Output latch position counter on index event
	Output latch position counter on strobe event
Speed calculation	Output capture timer
	Output capture period timer
	Enable and output overflow error flag
	Enable and output direction change error flag
	Output capture timer latched value
	Output capture timer period latched value
	Output position counter latched value

Interrupt

Interrupts corresponding to specific events are enabled or disabled based on the settings in this tab. The generated interrupts are used with the C28x Hardware Interrupt.

`Position counter error interrupt enable` — Enables position counter error interrupts

`off` (default) | `on`

Enables position counter error interrupts. The position counter interrupt is generated only in index event reset mode. When eQEP is configured in this mode, the position counter value is latched to the QPOSILAT register, and the direction information is recorded on every index event marker. If the latched value is not equal to 0 or QPOSMAX, the position counter error interrupt is generated.

`Quadrature phase error interrupt enable` — Enables quadrature phase error interrupts

`off` (default) | `on`

Enables quadrature phase error interrupts. In quadrature count mode, the quadrature inputs QEPA and QEPB are expected to be 90 degrees out of phase. The quadrature phase error interrupt is generated when edge transition is detected simultaneously on the QEPA and QEPB signals.

`Quadrature direction change interrupt enable` — Enables quadrature direction change interrupt
`off` (default) | `on`

When the direction of counting changes, the quadrature direction change interrupt is generated.

`Watchdog timeout interrupt enable` — Enables watchdog timeout interrupts
`off` (default) | `on`

The eQEP peripheral contains a watchdog timer that monitors the quadrature clock. If no quadrature clock event is detected until the watchdog timer matches the watchdog period, time-out occurs and the watchdog timeout interrupt is generated.

`Position counter underflow interrupt enable` — Enables position counter underflow interrupts
`off` (default) | `on`

Enables position counter underflow interrupts. In the reverse direction, if the position counter reaches 0, then the position counter is reset to QPOSMAX on the next QEP clock and the position counter underflow interrupt is generated.

`Position counter overflow interrupt enable` — Enables position counter overflow interrupts
`off` (default) | `on`

Enables position counter overflow interrupts. In the forward direction, if the position counter reaches QPOSMAX, the position counter is reset to 0 on the next QEP clock, and the position counter overflow interrupt is generated.

`Position-compare ready interrupt enable` — Enables position compare ready interrupts
`off` (default) | `on`

Enables position compare ready interrupts. When the position compare register is configured for shadow mode, the position compare ready interrupt is generated after the shadow register value is loaded into the active register.

`Position-compare match interrupt enable` — Enables position compare match interrupts
`off` (default) | `on`

Enables position compare match interrupts. The position compare match interrupt is generated when the position counter value QPOSCNT matches with the active position compare register QPOSCMP.

`Strobe event latch interrupt enable` — Enables strobe event latch interrupts
`off` (default) | `on`

Enables strobe event latch interrupts. The strobe event latch interrupt is generated when the position counter is latched to the QPOSSLAT register during a strobe event latch.

`Index event latch interrupt enable` — Enables index event latch interrupts
`off` (default) | `on`

Enables index event latch interrupts. The strobe event latch interrupt is generated when the position counter is latched to the QPOSILAT register during an index event latch.

`Unit timeout interrupt enable` — Enables unit timeout interrupts
`off` (default) | `on`

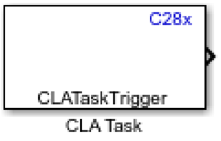
Enables unit timeout interrupts. The unit time-out interrupt is generated when the unit timer matches the unit period.

See Also

"Embedded Coder Support Package for Texas Instruments C2000 Processors"

C28x CLA Task

Create CLA task that executes downstream function-call subsystem on CLA core



Library: Embedded Coder Support Package for Texas Instruments C2000 Processors / F2803x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2805x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2806x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2807x
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xD
Embedded Coder Support Package for Texas Instruments C2000 Processors / F2837xS
Embedded Coder Support Package for Texas Instruments C2000 Processors / F28004x

Description

The CLA Task block creates a (Control Law Accelerator) CLA task that executes a downstream function-call subsystem on the CLA core. CLA is a coprocessor that allows parallel processing. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

You can specify the interrupt source to trigger the CLA task. You can create up to eight CLA tasks to execute on the CLA core.

For information about how to configure a CLA block to execute a downstream function-call subsystem, see “Using the Control Law Accelerator (CLA)”.

Limitations

- The CLA application code can only be triggered by a C28x event. CLA Task can be triggered by the C28x CPU via software or by different peripheral interrupts.
- All interfaces between the CLA and the CPU must be placed in specific memory locations. The CpuToCla1MsgRAM memory section is used to exchange data from

C28x to the CLA. The Cla1ToCpuMsgRAM memory section is used to exchange data from the CLA to C28x.

- The CLA application code does not have access to global variables.
- Early versions of the CLA C compiler support only two levels of function calls. CLA interrupt service routines may call leaf functions only. Leaf functions may not call other functions.
- Recursive function calls are not supported.
- Integer division, modulus, and integer unsigned comparison are not supported with the CLA C compiler.

For more details and a full list of limitations, see http://processors.wiki.ti.com/index.php/C2000_CLA_C_Compiler.

Ports

Output

`Port_1` — Function-call signal to a function-call subsystem or function-call model
scalar

The output triggers the CLA task that executes a downstream function-call subsystem on the CLA core.

Parameters

`CLA task number` — CLA task number executed on CLA core
`1` (default) | integer in [1, 8]

The CLA task number that you want to execute on the CLA core.

`CLA task trigger source` — Source of CLA task trigger
`Software` (default) | peripheral interrupt

The software or peripheral interrupt source that triggers the CLA task to execute on the CLA core.

`Sample time` — Frequency at which function-call subsystem is triggered
`0.2` (default) | `-1` | scalar

Set the frequency at which the function-call subsystem is triggered by the **CLA task trigger source**. To execute this block asynchronously, set this parameter to `-1`.

Dependencies

This parameter appears only when you select the `Software` option in the **CLA task trigger source** parameter.

See Also

"Embedded Coder Support Package for Texas Instruments C2000 Processors"

Introduced in R2016b

Byte Pack

Convert input signals to 8-, 16-, or 32-bit vector

Library: Simulink Support Package for Arduino Hardware/Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/Utilities
Embedded Coder Support Package for Texas Instruments C2000 Processors/Target Communication

Description

The Byte Pack block converts one or more signals of user-selectable data types to a single `uint8`, `uint16`, or `uint32` vector output. Using the parameters of this block, you specify the input data types and the alignment of the data in the output vector. The output of this block connects to an input port of a send block, such as SPI Transmit, SCI Transmit, or UDP Send. The send block then transmits signals across various communication networks, such as SPI, SCI, UDP, or I2C.

Input/Output Ports

Input

`Port_1` — First of *N* input ports
scalar | vector | matrix

The number of input ports and their types specified as a cell array in the **Input port data types (cell array)** parameter. The block can have from `1` to *N* input ports. *N* is the number of incoming data types specified in the cell array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

`Port_1` — Vector containing packed data
vector

Transmits a vector of packed data.

Data Types: `uint8` | `uint16` | `uint32`

Parameters

`Output port (packed) data type` — Data type of packed output signal
`uint8` (default) | `uint16` | `uint32`

The data type of the packed output signal at the output port.

`Input port data types (cell array)` — Data types of unpacked input signals
`{'double'}` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Data types of input signals (unpacked), specified as a cell array. The block creates input ports in the order of the incoming data types specified in the cell array. For example, the first data type in the cell array corresponds to the top port and the last data type corresponds to the bottom port.

For example, if the data types are `single`, `uint8`, and `uint8`, the block creates three input ports. The order of the input port data types is same as the data types specified in the cell array.

`Byte alignment` — Alignment of input signal data types after packing
1 (default) | 2 | 4 | 8

Each element in the input signal list starts at a multiple of the byte alignment value, specified from the start of the vector. If the byte alignment value is larger than the size of the data type in bytes, the input values are padded with zeros to fill the space allotted.

For example, if the byte alignment value is 4, a `uint32` receives no padding, a `uint16` receives 2 bytes of padding, and a `uint8` receives 3 bytes of padding.

Tip If the model accesses the data items frequently, consider selecting a byte alignment value equal to the largest data type that you want to access. If the model transfers the data items frequently as a group, consider selecting a byte alignment value of `1`, which packs the data into the smallest space possible.

Example

Suppose that you are packing four signals into a vector of data type `uint8` or `uint16`, and the signals have these attributes.

Dimension	Size	Type
Vector	3	`int8`
Vector	2	`int16`
Scalar	1	`uint8`
Scalar	1	`uint32`

To pack the signals:

1. Set **Output port (packed) data type**. This example compares `uint8` and `uint16`.
2. Set **Input port data types (cell array)** to:

   ```
   {'int8', ‘int16’, ‘uint8’, ‘uint32’}
   ```

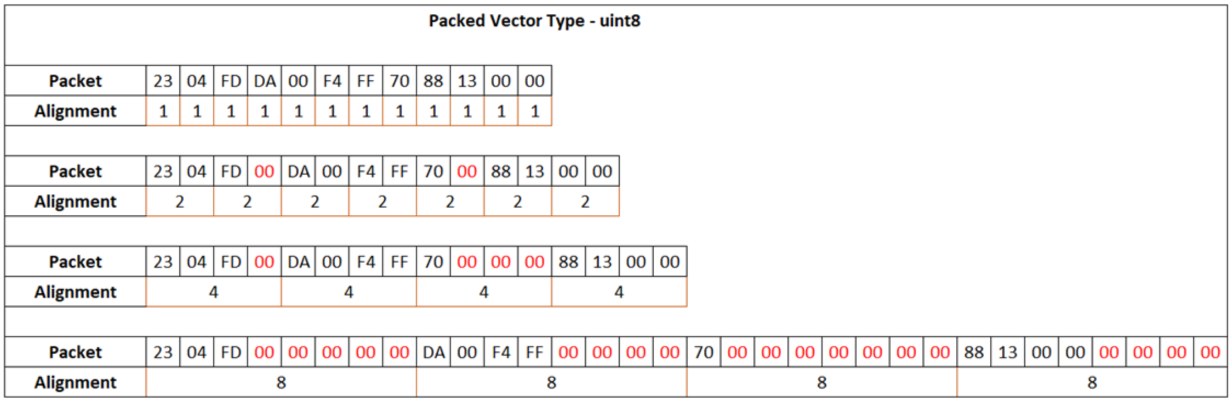
   The block creates four input ports that match the order of the incoming signal data types specified in the cell array.
3. Set the required byte alignment value. The byte alignment value specifies the number of bytes after which a new byte starts from the previous boundary.

   The size of the output is based on the packed vector size, the byte alignment value, and the smallest memory cell size of the processor. Depending on the byte alignment value, input values are padded with zeros before the next signal is packed. The smallest addressable memory cell indicates the number of bits occupied by the `char` or `uint8` data type for a processor and determines the structure of packets.
4. Connect incoming signals to the input port of the Byte Pack block.

For processors with a smallest addressable memory cell of 8 bits per char, consider these values for input signals.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	23
			4	04
			-3	FD
Vector	2	`int16`	218	00DA
			-12	FFF4
Scalar	1	`uint8`	112	70
Scalar	1	`uint32`	5000	00001388

The packed output vector data type `uint8` is:



Packed Vector Type - uint8

Packet	23	04	FD	DA	00	F4	FF	70	88	13	00	00
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	23	04	FD	00	DA	00	F4	FF	70	00	88	13	00	00
Alignment	2		2		2		2		2		2		2	

Packet	23	04	FD	00	DA	00	F4	FF	70	00	00	00	88	13	00	00
Alignment	4				4				4				4			

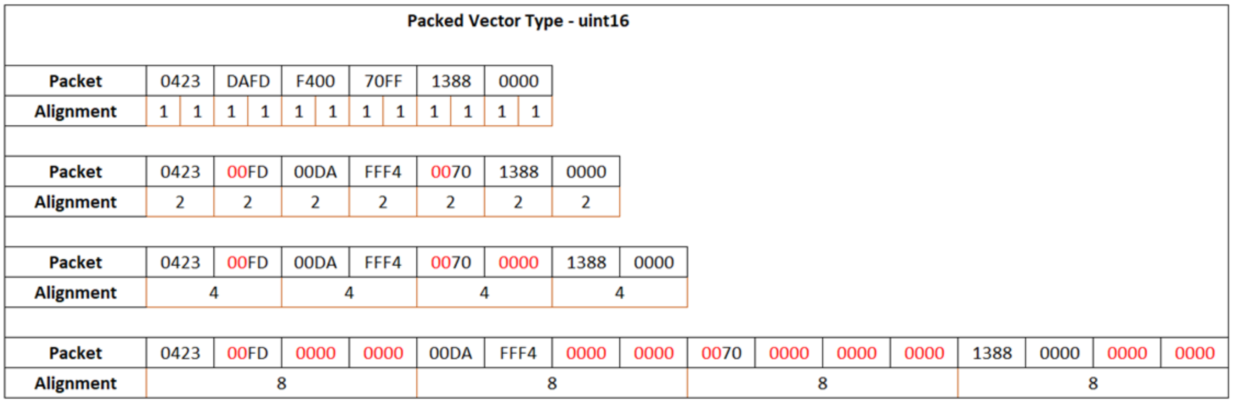
Packet	23	04	FD	00	00	00	00	00	DA	00	F4	FF	00	00	00	00	70	00	00	00	00	00	00	00	88	13	00	00	00	00	00	00
Alignment	8								8								8								8							

Red zeros represent padded empty memory cells.

For a packed output vector of data type `uint8` and byte alignment value 2, the `int8` data value (23 04 FD) occupies the first three memory cells, with each cell occupying 8 bits. Because three is not a multiple of the byte alignment value 2, the next input signal of `int16` data value (00DA FFF4) is allocated the next four cells (fifth through eighth), leaving the fourth cell empty. The block fills the empty cell with zero. The rest of the input signals are packed in a similar way.

After packing all input signals, the Byte Pack block calculates the total packets allocated and outputs a `uint8` vector of size 4 + 4 + 2 + 4 = 14. Here, the `int8` signal occupies the first 4 cells, the `int16` signal occupies the second 4 cells, the `uint16` signal occupies the third 2 cells, and the `uint32` signal occupies the fourth 4 cells.

The packed output vector of data type `uint16` is:



Packed Vector Type - uint16

Packet	0423	DAFD	F400	70FF	1388	0000
Alignment	1 1	1 1	1 1	1 1	1 1	1 1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

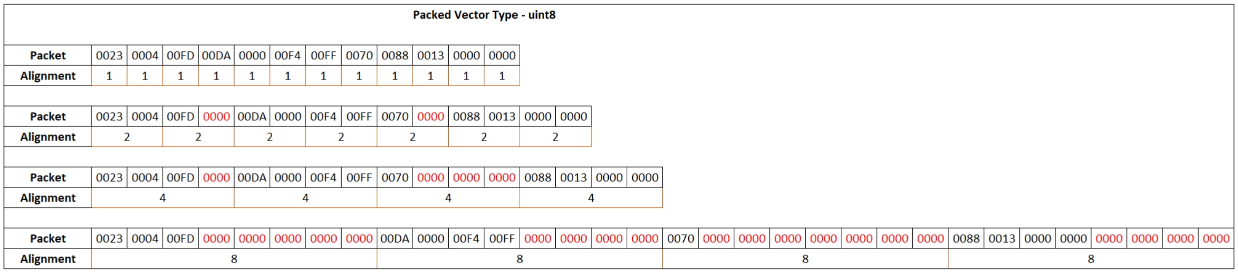
Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

For processors such as Texas Instruments C2000, with the smallest addressable memory cell of 16 bits per char, consider these values for input signals. The `int8` and `uint8` data values occupy 16 bits, as indicated by the hex value.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	0023
			4	0004
			-3	FFFD
Vector	2	`int16`	218	00DA
			-12	FFF4
Scalar	1	`uint8`	112	0070
Scalar	1	`uint32`	5000	00001388

For the packed output vector of data type `uint8`, the output packet occupies 16 bits, although the data value the packet represents is 8 bits. The byte alignment values are calculated with respect to the16-bit addressable memory.



Packed Vector Type - uint8

Packet	0023	0004	00FD	00DA	0000	00F4	00FF	0070	0088	0013	0000	0000
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

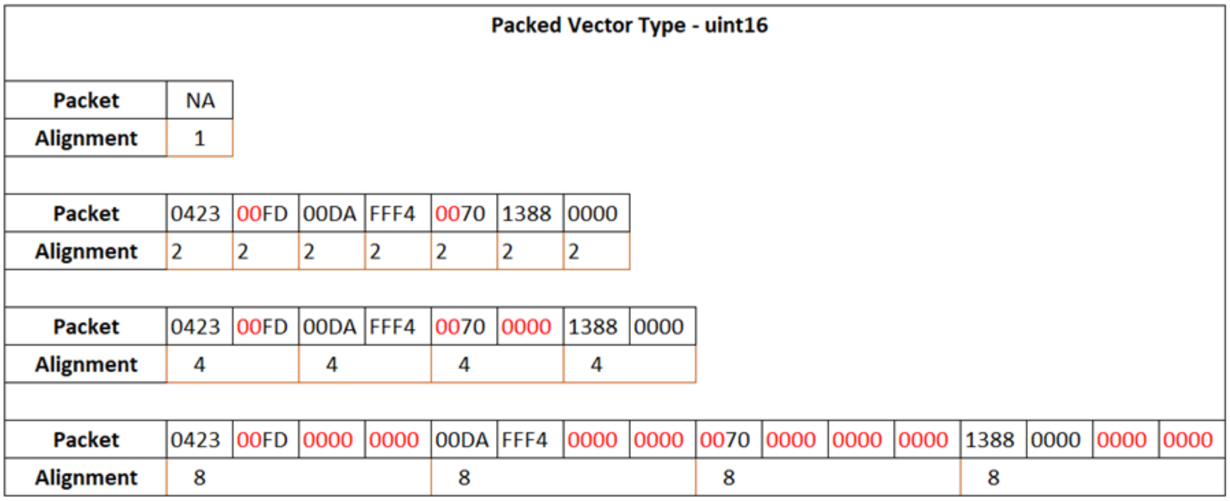
Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0088	0013	0000	0000
Alignment	2		2		2		2		2		2		2	

Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0000	0000	0088	0013	0000	0000
Alignment	4				4				4				4			

Packet	0023	0004	00FD	0000	0000	0000	0000	0000	00DA	0000	00F4	00FF	0000	0000	0000	0000	0070	0000	0000	0000	0000	0000	0000	0000	0088	0013	0000	0000	0000	0000	0000	0000
Alignment	8								8								8								8							

For a packed output vector of data type `uint8` and byte alignment value 2, the `int8` data value (0023 0004 00FD) occupies the first three memory cells, with each cell occupying 16 bits. Because three is not a multiple of byte alignment value 2, the next signal of data type `int16` (00DA 0000 00F4 00FF) is allocated the next four cells (fifth through eighth), leaving the fourth cell empty. The block fills the empty cell with zero. The rest of the input signals are packed in a similar way. After packing all input signals, the Byte Pack block calculates total packets allocated and outputs a `uint8` vector of size 4 + 4 + 2 + 4 = 14.

For the packed output vector of data type `uint16`, the output packet occupies 16 bits, and the data value the packet represents is also 16 bits. For a packet size of 16 and larger, the byte alignment is calculated with respect to the number of bytes the data values are packed into. Therefore, in this case, 1-byte alignment is not allowed.



Packed Vector Type - uint16

Packet	NA
Alignment	1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

For a packed output of data type `uint16` and byte alignment value 2, the three `int8` data values (0423 FD) are packed together as two words in the first two memory cells. The fourth byte in the second memory cell is empty and filled with zero. The `int16` data value (00DA FFF4) is allocated the next two memory cells (third and fourth). The rest of the input signals are packed in a similar way. After packing all signals, the Byte Pack block calculates total packets allocated and outputs a `uint16` vector of size 2 + 2 + 1 + 2 = 7.

See Also

Byte Unpack |

Introduced in R2016b

Byte Unpack

Unpack 8-, 16-, or 32-bit input vector to multiple output vectors

Library: Simulink Support Package for Arduino Hardware/Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/Utilities
Embedded Coder Support Package for Texas Instruments C2000 Processors/Target Communication

Description

The Byte Unpack block converts a vector of `uint8`, `uint16`, or `uint32` data type to one or more signals of user-selectable data types. This block is the inverse of the Byte Pack block. The input of this block connects to an output port of a receive block, such as SPI Receive, SCI Receive, or UDP Receive. The Receive block then transmits signals across various communication networks, such as SPI, SCI, UDP, or I2C.

Input/Output Ports

Input

`Port_1` — Packed data
scalar | vector | matrix

Receives a vector of packed data.

Data Types: `uint8` | `uint16` | `uint32`

Output

`Port_1` — First of *N* output ports
scalar | vector | matrix

The block can have from 1 to *N* output ports, as specified by elements of the cell array in the parameter **Output port data types (cell array)**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

`Output port dimensions (cell array)` — Dimensions of each output port (unpacked)
`{[1]}` (default) | `{[`*N*`], [`*M*`], ...}`

Output port dimensions specified as a cell array of vectors.

Specify the same dimensions that you set for the corresponding Byte Pack block in the model.

`Output port data types (cell array)` — Data types for unpacked output signals
`double` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Data types of the output ports (unpacked) specified for different output signals as a cell array. The number of elements in the cell array determines the number of output ports shown by this block instance.

Specify the same data types that you set in the **Input port data types (cell array)** parameter for the corresponding Byte Pack block in the model.

`Byte alignment` — Alignment of output signal data types before unpacking
1 (default) | 2 | 4 | 8

Each element in the input signals list starts at a multiple of the byte alignment value, specified from the start of the vector. If the byte alignment value is larger than the size of the data type in bytes, the output values are padded with zeros to fill the space allotted.

For example, if the byte alignment value is 4, a `uint32` receives no padding, a `uint16` receives 2 bytes of padding, and a `uint8` receives 3 bytes of padding.

Choose the same byte alignment value that you set in the **Byte alignment** parameter for the corresponding Byte Pack block in the model.

Example

Suppose that you are unpacking a vector of data type `uint8` or `uint16`, and the unpacked signals have these attributes.

Dimension	Size	Type
Vector	3	`int8`
Vector	2	`int16`
Scalar	1	`uint8`
Scalar	1	`uint32`

To unpack the signals:

1. Set **Output port dimensions (cell array)** to:

   ```
   {'3’, ‘2’, ‘1’, ‘1’}
   ```
2. Set **Output port data types (cell array)** to:

   ```
   {'int8’, ‘int16’, ‘uint8’, ‘uint32’}
   ```

   The block creates four output ports that match the order of the signal data types specified in the cell array.
3. Set the required byte alignment value. The byte alignment value specifies the number of bytes after which a new byte starts from the previous boundary.

   The size of the output is based on the packed vector size, the byte alignment value, and the smallest memory cell size of the processor. Depending on the byte alignment value, output values padded with zeros are discarded before the next signal is unpacked. The smallest addressable memory cell indicates the number of bits occupied by `char` or `uint8` data type for a processor, and determines the structure of packets.
4. Connect incoming signals to the input port of the Byte Unpack block.

For processors with a smallest addressable memory cell of 8 bits per char, consider the packed input vector data type `uint8`.

Packed Vector Type - uint8

Packet	23	04	FD	DA	00	F4	FF	70	88	13	00	00
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	23	04	FD	00	DA	00	F4	FF	70	00	88	13	00	00
Alignment	2		2		2		2		2		2		2	

Packet	23	04	FD	00	DA	00	F4	FF	70	00	00	00	88	13	00	00
Alignment	4				4				4				4			

Packet	23	04	FD	00	00	00	00	00	DA	00	F4	FF	00	00	00	00	70	00	00	00	00	00	00	00	88	13	00	00	00	00	00	00
Alignment	8								8								8								8							

Red zeros represent padded empty memory cells.

For a packed input vector of data type `uint8` and byte alignment value 2, the `int8` data value (23 04 FD) occupies three memory cells, with each cell occupying 8 bits. The next input signal of `int16` data value (00DA FFF4) occupies the next four cells (fifth through eighth), and the fourth cell is empty (padded). The Byte Unpack block considers the alignment and padding of cells while unpacking.

The packed input vector of data type `uint16` is:

Packed Vector Type - uint16

Packet	0423		DAFD		F400		70FF		1388		0000	
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

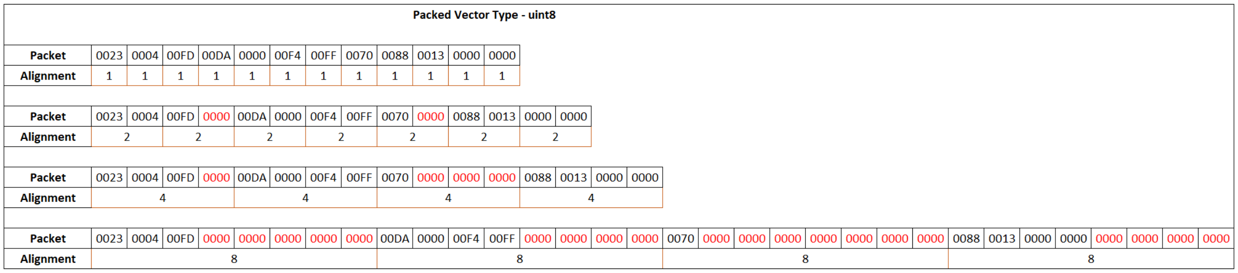
Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

The unpacked output signals are:

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	23
			4	04
			-3	FD
Vector	2	`int16`	218	00DA
			-12	FFF4
Scalar	1	`uint8`	112	70
Scalar	1	`uint32`	5000	00001388

For processors such as Texas Instruments C2000, with a smallest addressable memory cell of 16 bits per char, consider a packed input vector data type `uint8`. The output packet occupies 16 bits although the data value that the packet represents is 8 bits. The byte alignment values are calculated with respect to the 16-bit addressable memory.


Packed Vector Type - uint8

Packet	0023	0004	00FD	00DA	0000	00F4	00FF	0070	0088	0013	0000	0000
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

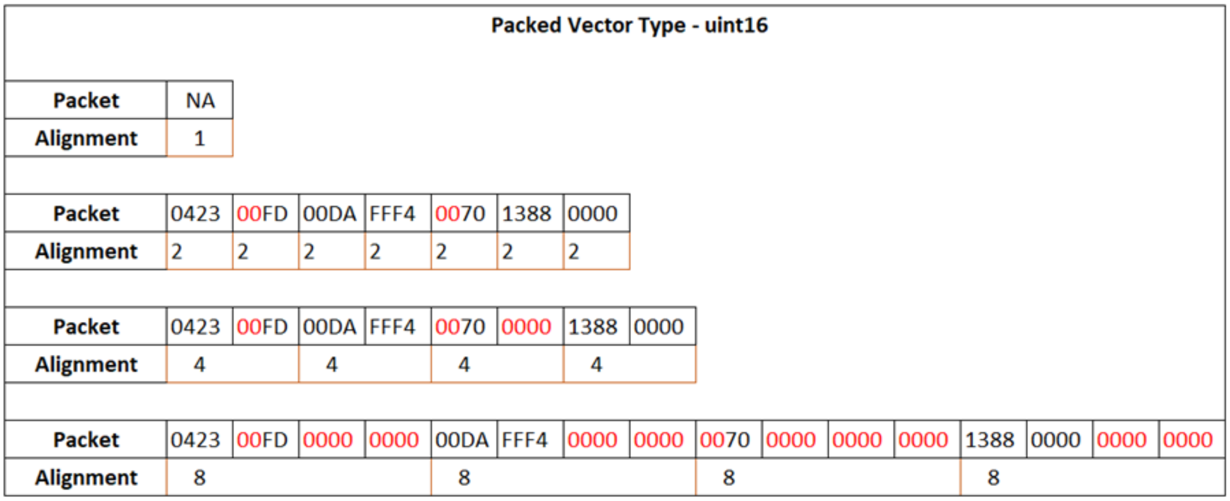
Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0088	0013	0000	0000
Alignment	2		2		2		2		2		2		2	

Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0000	0000	0088	0013	0000	0000
Alignment	4				4				4				4			

Packet	0023	0004	00FD	0000	0000	0000	0000	0000	00DA	0000	00F4	00FF	0000	0000	0000	0000	0070	0000	0000	0000	0000	0000	0000	0000	0088	0013	0000	0000	0000	0000	0000	0000
Alignment	8								8								8								8							

For a packed input vector of data type `uint8` and byte alignment value 2, the `int8` data value (0023 0004 00FD) occupies three memory cells, with each cell occupying 16 bits. The next signal of data type `int16` (00DA 0000 00F4 00FF) occupies the next four cells (fifth through eighth), and the fourth cell is empty (padded). The Byte Unpack block considers the alignment and padding of cells while unpacking.

For the packed input vector of data type `uint16`, the output packet occupies 16 bits, and the data value the packet represents is also 16 bits. For a packet size of 16 and larger, the byte alignment is calculated with respect to the number of bytes the data values have to be packed. Therefore, in this case, 1-byte alignment is not allowed.



Packed Vector Type - uint16

Packet	NA
Alignment	1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

For a packed input of data type `uint16` and byte alignment value 2, the three `int8` data values (0423 FD) occupy the first two memory cells. The fourth byte in the second memory cell is empty and padded with zero. The `int16` data value (00DA FFF4) occupies the next two memory cells (third and fourth). The Byte Unpack block considers the alignment and padding of cells while unpacking.

The table lists the unpacked output signals. The `int8` and `uint8` data values occupy 16 bits, as indicated by the hex value.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	0023
			4	0004
			-3	FFFD
Vector	2	`int16`	218	00DA
			-12	FFF4
Scalar	1	`uint8`	112	0070
Scalar	1	`uint32`	5000	00001388

See Also

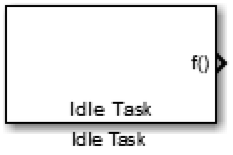
Byte Pack |

Introduced in R2016b

Idle Task

Create free-running task

Description



The Idle Task block, and the subsystem connected to it, specify one or more functions to execute as background tasks. The tasks executed through the Idle Task block are of the lowest priority, lower than that of the base rate task.

This block is not supported on targets running an operating system or RTOS.

Vectorized Output

The block output comprises a set of vectors—the task numbers vector and the preemption flag or flags vector. A preemption-flag vector must be the same length as the number of tasks vector unless the preemption flag vector has only one element. The value of the preemption flag determines whether a given interrupt (and task) is preemptable. Preemption overrides prioritization. A lower-priority nonpreemptable task can preempt a higher-priority preemptable task.

When the preemption flag vector has one element, that element value applies to the functions in the downstream subsystem as defined by the task numbers in the task number vector. If the preemption flag vector has the same number of elements as the task number vector, each task defined in the task number vector has a preemption status defined by the value of the corresponding element in the preemption flag vector.

Parameters

Task numbers

Identifies the created tasks by number. Enter as many tasks as you need by entering a vector of integers. The default values are `[1,2]` to indicate that the downstream subsystem has two functions.

The values you enter determine the execution order of the functions in the downstream subsystem, while the number of values you enter corresponds to the number of functions in the downstream subsystem.

Enter a vector containing the same number of elements as the number of functions in the downstream subsystem. This vector can contain up to 16 elements, and the values must be from 0 to 15 inclusive.

The value of the first element in the vector determines the order in which the first function in the subsystem is executed, the value of the second element determines the order in which the second function in the subsystem is executed, and so on.

For example, entering [2,3,1] in this field indicates that there are three functions to be executed, and that the third function is executed first, the first function is executed second, and the second function is executed third. After the functions are executed, the Idle Task block cycles back and repeats the execution of the functions in the same order.

Preemption flags

Higher-priority interrupts can preempt interrupts that have lower priority. To allow you to control preemption, use the preemption flags to specify whether an interrupt can be preempted.

Entering `1` indicates that the interrupt can be preempted. Entering `0` indicates the interrupt cannot be preempted. When **Task numbers** contains more than one task, you can assign different preemption flags to each task by entering a vector of flag values, corresponding to the order of the tasks in **Task numbers**. If **Task numbers** contains more than one task, and you enter only one flag value here, that status applies to the tasks.

In the default settings `[0 1]`, the task with priority `1` in **Task numbers** is not preemptable, and the priority `2` task can be preempted.

Enable simulation input

When you select this option, Simulink software adds an input port to the Idle Task block. This port is used in simulation only. Connect one or more simulated interrupt sources to the simulation input.

Note Select this check box to test asynchronous interrupt processing behavior in Simulink software.

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x GPIO Digital Input

Configure general-purpose input/output pins as digital input

GPIOx
GPIO DI
Digital Input

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2805x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C280x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2833x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2834x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2807x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xD

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xS

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2838x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F28004x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

This block configures the general-purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation (the default). When a pin is configured for digital input, it becomes unavailable for digital output or peripheral operation. You can configure the **Input qualification type** for individual digital input pins. To configure, go to **Configuration Parameters > Hardware Implementation > Target Hardware Resources** and select the appropriate GPIO group.

Each processor has a different number of available GPIO pins.

Note To avoid losing new settings, click **Apply** before changing the **GPIO Group** parameter.

Parameters

GPIO Group

Select the group of GPIO pins you want to view or configure. For a table of GPIO pins and peripherals, refer to the Texas Instruments documentation for your specific target.

Sample time

Specify the time interval between output samples. To inherit sample time from the upstream block, set this parameter to `-1`. For more information, refer to the section on "Specify Sample Time" (Simulink) in the Simulink documentation.

Data type

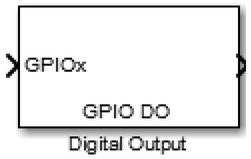
Specify the data type of the input. The input is read as 16-bit integer, and then cast to the selected data type. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `boolean`.

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F28004x GPIO Digital Output

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x GPIO Digital Output

Configure general-purpose input/output pins as digital output



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2805x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C280x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2833x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2834x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2807x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xD

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2837xS

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F2838x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ F28004x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Configure individual general-purpose input/output (GPIO) pins to operate as digital outputs. When a pin is configured for digital output, it cannot operate as a digital input or connect to peripheral I/O signals. When you select a pin for digital output, the user interface presents a **Toggle** option that inverts the output signal on the pin.

Each processor has a different number of available GPIO pins.

Note To avoid losing new settings, click **Apply** before changing the **GPIO Group** parameter.

Parameters

GPIO Group

Select the group of GPIO pins you want to view or configure.

GPIO pins for output

To configure a GPIO pin for digital output, select the checkbox next to it. Refer to the block for a table of all available peripherals for each pin.

A value of `True` at the input of the block drives the selected GPIO pin high. A value of `False` at the input of the block grounds the selected GPIO pin.

Toggle GPIO[bit#]

For each pin selected for output, you can elect to toggle the signal of that pin. In **Toggle** mode, a value of `True` at the input of the block switches the GPIO pin output level. Thus, if the GPIO pin was driven high, in **Toggle** mode, with the value of `True` at the input, the pin output level is driven low. If the GPIO pin was driven low, in **Toggle** mode, with the value of `True` at the input of the block, the same pin output level is driven high. If the input of the block is `False`, the GPIO pin output level is unaffected.

Note The outputs of this block can be vectorized.

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F28004x GPIO Digital Input

C2802x/C2803x/C2806x/F28M3x AnalogIO Input

Configure pin, sample time, and data type for analog input

AIOx
AIO DI
AnalogIO Input

Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Use this block to sample the voltage on Analog IO pins and output the results.

Parameters

Parameters (Input pins)

Select the input pins to sample.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

Data type

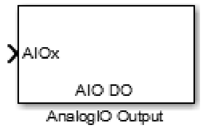
Select the data type of the digital output data. If you select `auto`, the block automatically selects the data type for your model. You can also manually select a data type. You can choose from the options `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

See Also

C2802x/C2803x/C2806x/F28M3x AnalogIO Output

C2802x/C2803x/C2806x/F28M3x AnalogIO Output

Configure Analog IO to output analog signals on specific pins



Library

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2802x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2803x

Embedded Coder Support Package for Texas Instruments C2000 Processors/ C2806x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M35x/ C28x

Embedded Coder Support Package for Texas Instruments C2000 F28M3x Concerto Processors/ F28M36x/ C28x

Description

Use this block to drive the output voltage of Analog IO pins.

In regular mode, a value of True at the input of the block pulls the Analog IO pin high while a value of False grounds the pin.

In toggle mode, a value of True at the input of the block switches the actual output level of the Analog IO pin while a value of False does not alter the output level of the Analog IO pin.

Parameters

Parameters (Output Pins)

Select the analog output pins. Selecting **Toggle** inverts the output voltage levels of the pins if the input of the block is True.

See Also

C2802x/C2803x/C2806x/F28M3x AnalogIO Input

3

Appendix

Support SPI Communication

In this section...

SPI, or Serial Peripheral Interface, is a synchronous, full duplex serial communication protocol between high-speed devices over short distances. The SPI protocol supports a single master with one or more slaves. The master can communicate to any slave on the bus, but each slave can communicate only with the master.

The SPI Master Transfer block in the support package library enables communication with other SPI devices. You can use this block only when you use your hardware as the master device.

With SPI, you can:

- Connect various sensors to boards to measure different quantities such as temperature, pressure.
- Connect various shields to boards to enhance capabilities such as WiFi shield.
- Access an SD card to store data or extend the available memory.

You can set SPI properties such as the SPI clock out frequency (in MHz), SPI mode, and the Bit order in the **Configuration Parameters > Hardware Implementation > SPI properties** section.

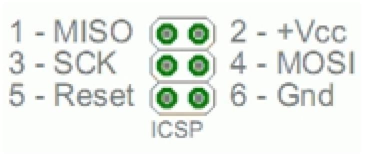
SPI Lines

SPI uses a four-wire serial bus for communication: MISO, MOSI, SCK, and SS. The MISO, MOSI, and the SCK lines are common to all devices. The SS line is specific to each slave.

- **MISO** (Master In Slave Out) – This line is the slave line for sending data to the SPI master.
- **MOSI** (Master Out Slave In) – This line is the master line for sending data to the SPI peripherals.
- **SCK** (Serial Clock) –The master generates the clock pulses that synchronize the data transmission.

- **SS** (Slave Select) – This is specific to the device. This is the pin on each device that the SPI master can use to enable and disable the device. This signal is an ‘active low’ signal which means a device becomes a slave when its SS pin is set to low.

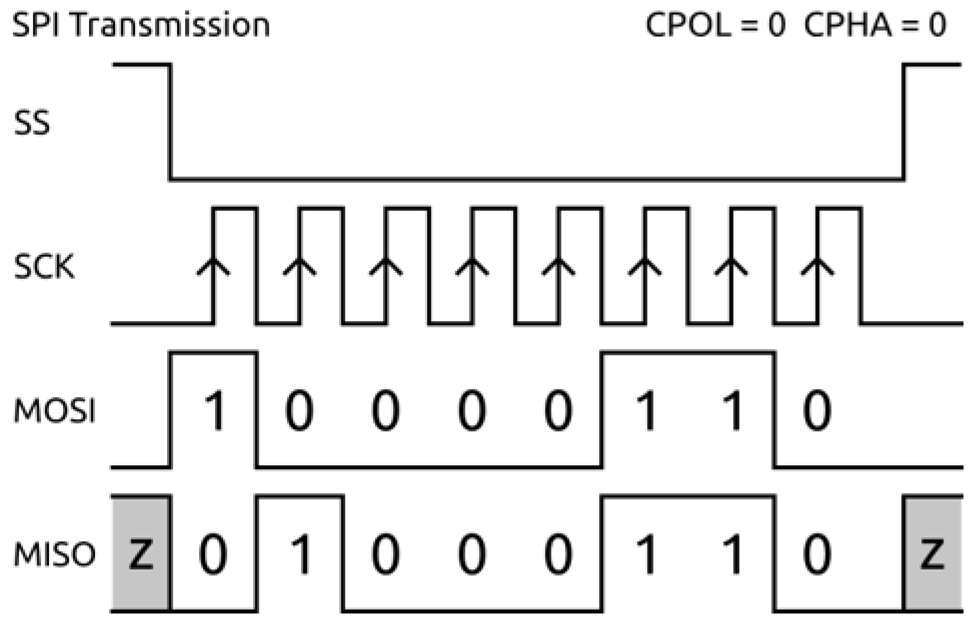
The SPI lines over In Circuit Serial Programming (ICSP) header are consistent across all the boards shown as follows.

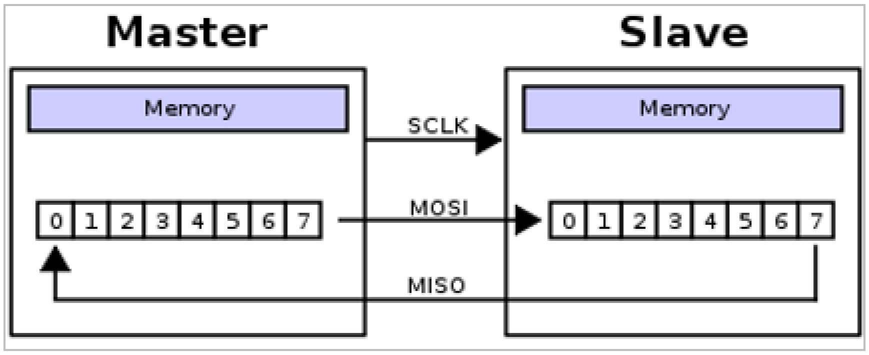


Data Transmission

- The SPI master sets the clock with a frequency supported by the SPI slave with which the master wants to communicate.
- The master selects the slave by setting the SS pin of slave to low (0). The master can select only one slave at a time.
- As each SPI transfer is a full duplex transmission, the master sends a bit on MOSI line and the slave reads it. The slave also sends a bit on the MISO line and the master reads it.

  When the master makes a data transfer, the slave cannot opt out of sending data. However, the slave device sends dummy bytes (usually all 1s or all 0s) when communication is one-way. Similarly, when the master reads data from a slave, the slave knows to ignore the data that the master sends.
- When the transfer is complete, the master stops toggling the SCK and mostly pulls up the SS to deselect the slave.
- During the data transmission, the other slaves on the SPI bus that have not been selected by the master ignore the SCK and the MOSI signals, and do not drive the MISO.





SPI Transfer Modes

An SPI master sets the clock polarity and the clock phase.

Clock Polarity (CPOL) – is the default value (HIGH/LOW) of SCK signal when the bus is idle.

CPOL = 0 means a default LOW value of SCK when bus is idle. CPOL = 1 means default HIGH value of SCK when bus is idle.

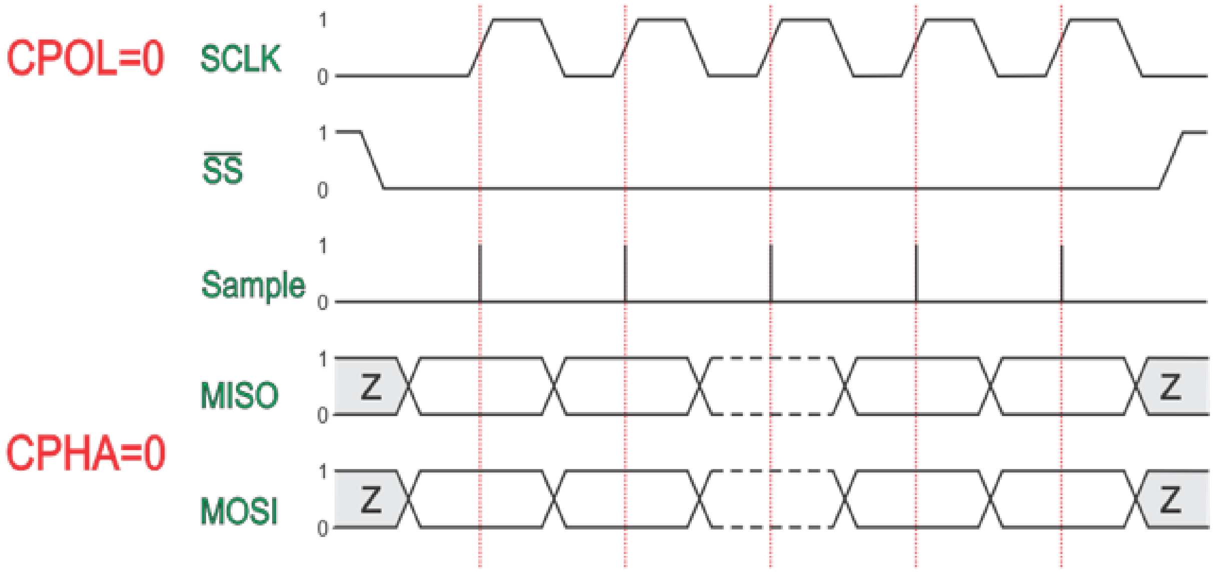
Clock Phase (CPHA) – indicates, if the clock data is sampled at LEADING (first) or TRAILING (second) edge of SCK.

CPHA = 0 means sample at LEADING edge of SCK and CPHA = 1 means sample at TRAILING edge of SCK, regardless of whether the clock edge is RISING or FALLING.

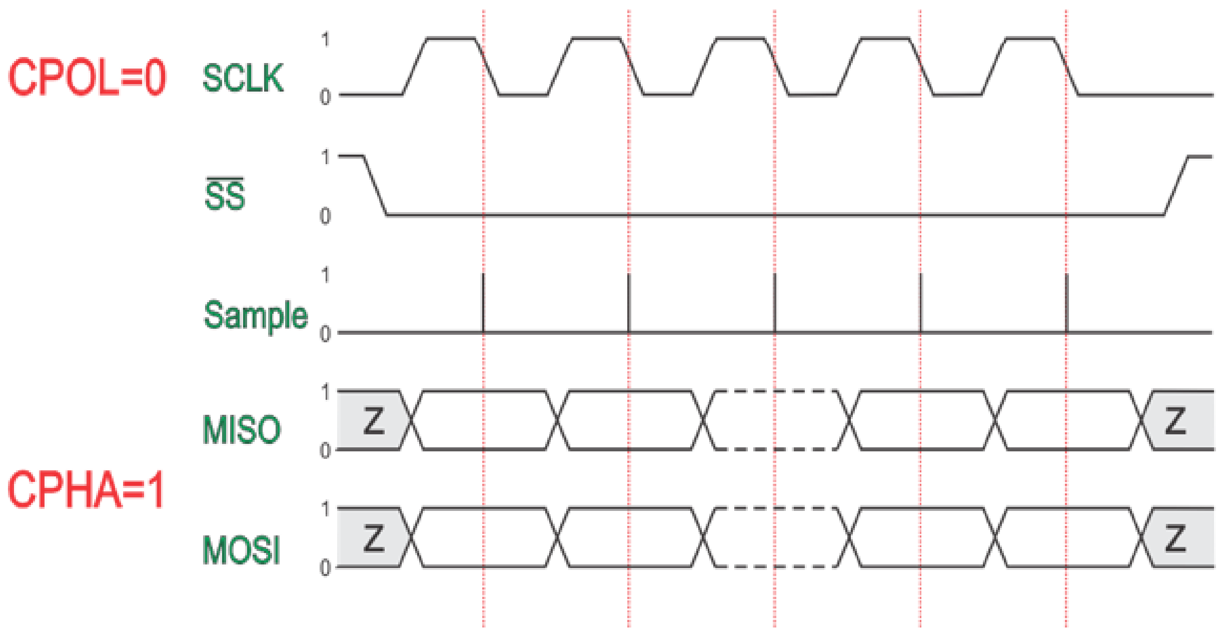
The combination of polarity and phase are referred to as SPI modes. The SPI modes 0–3 are shown in the table.

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)
0	0	1
1	0	0
2	1	1
3	1	0

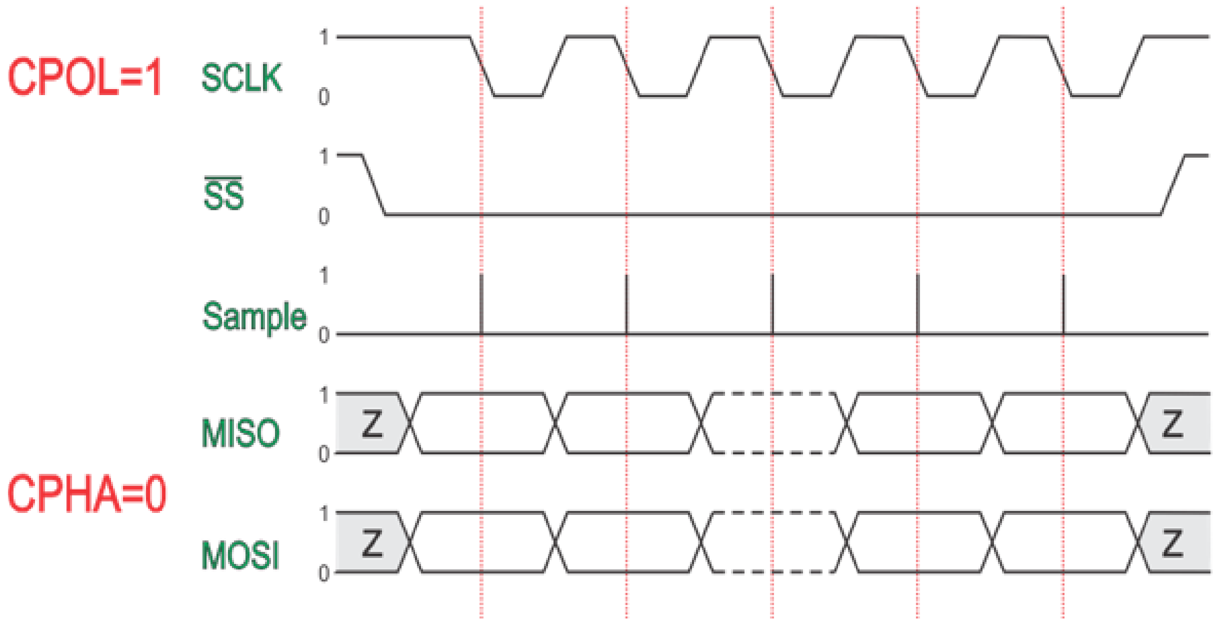
Mode 0



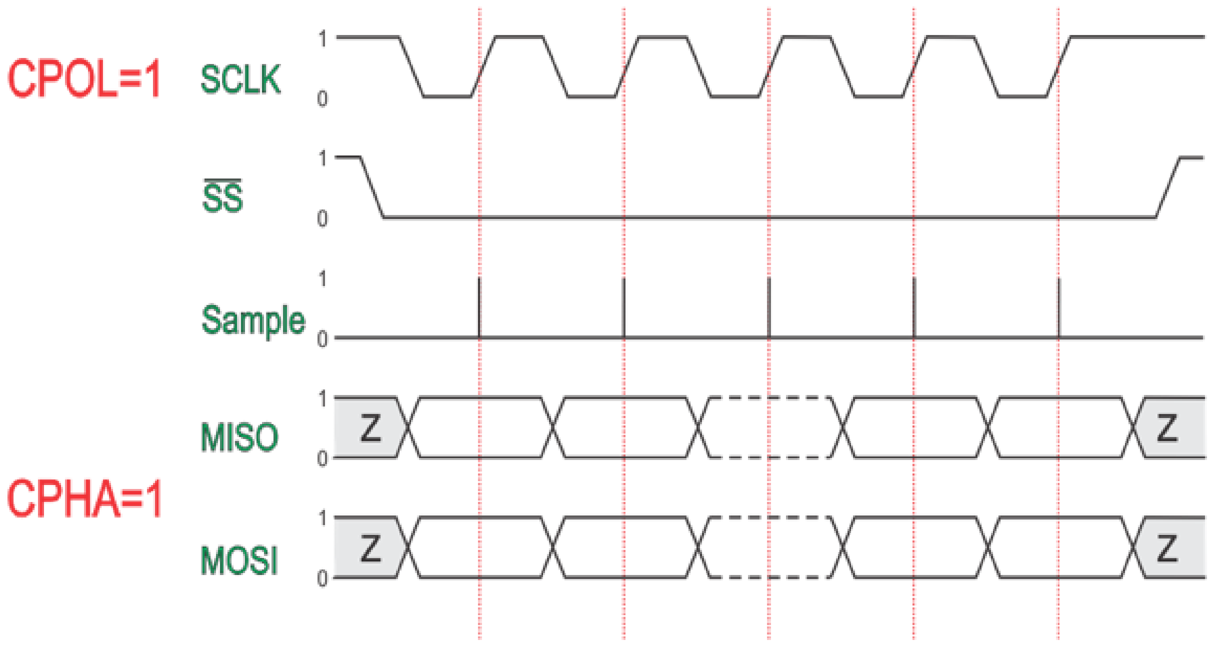
Mode 1



Mode 2



Mode 3


CPOL=1
CPHA=1
SCLK
SS
Sample
MISO
MOSI
1
0
Z
Z