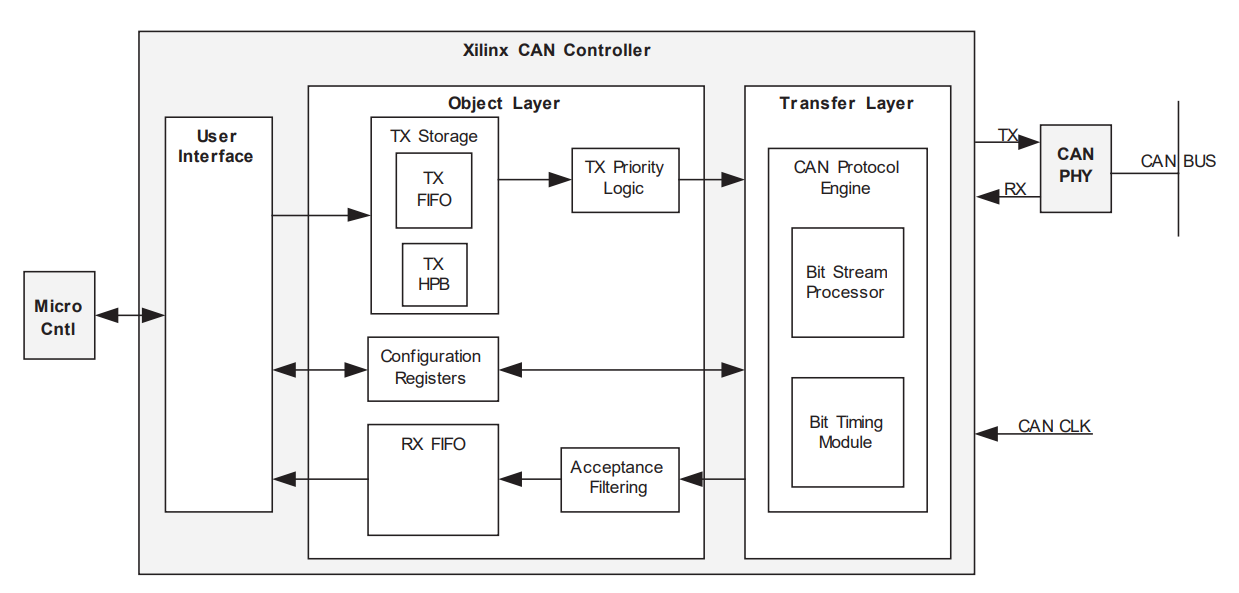
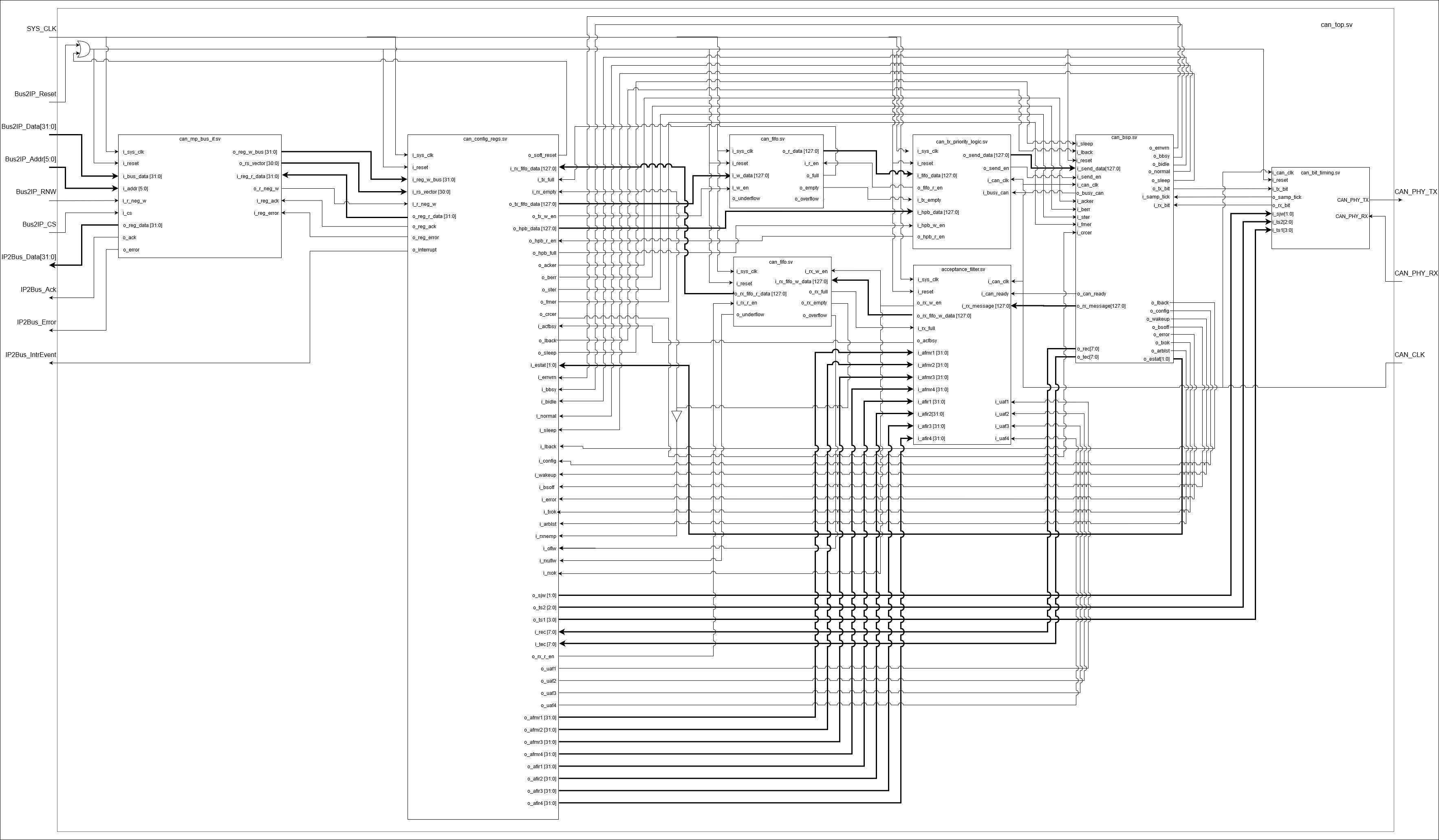
**System Specification v1**

**Design Overview:**





CAN Controller as per ISO 11898-2 based on Xilinx Logicore IP CAN v3.2 datasheet.

Inputs:

* From Microcontroller:
  + Bus2IP\_Reset
  + Bus2IP\_Data [31:0]
  + Bus2IP\_Addr[5:0]
  + Bus2IP\_RNW
  + Bus2IP\_CS
  + SYS\_CLK
* From CAN Transceiver:
  + CAN\_PHY\_RX

Outputs:

* To Microcontroller:
  + IP2Bus\_Data[31:0]
  + IP2Bus\_Ack
  + IP2Bus\_IntrEvent
  + IP2Bus\_Error
* To CAN Transceiver:
  + CAN\_PHY\_TX

**Microcontroller Bus Interface:**

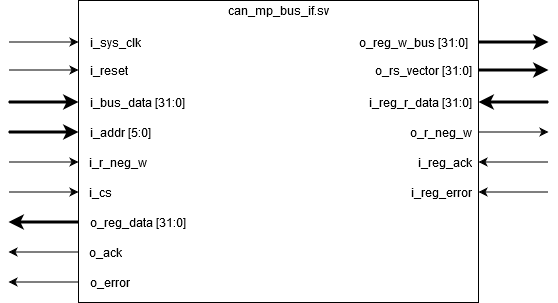
Decodes input address from Microcontroller and pulses the chip select bit corresponding to the register being addressed. On a low to high transition of ‘i\_cs’, if ‘i\_readnegwrite’ is high then this signals the start of a read operation, otherwise it starts a write operation.

Read operation:

On a low to high transition of ‘i\_cs’, ‘i\_addr’ gets decoded and the register select bit corresponding to the addressed register in ‘o\_rsvector’ gets pulsed for one clock cycle and the microcontroller bus interface waits for an acknowledge signal from the register module. Whenever the input ‘i\_regack’ is pulsed while the ‘i\_cs’ is still held high, the data from the register module ‘i\_regdata’ gets sent to ‘o\_regdata’ and ‘o\_ack’ gets pulsed for one clock cycle.

Write operation:

On a low to high transition of ‘i\_cs’, ‘i\_busdata’ gets latched, and ‘i\_addr’ gets decoded and the register select bit corresponding to the addressed register in ‘o\_rsvector’ gets pulsed for one clock cycle and the microcontroller bus interface waits for an acknowledge signal from the register module. Whenever the input ‘i\_regack’ is pulsed while the ‘i\_cs’ is still held high, ‘o\_ack’ gets pulsed for one clock cycle to signal end of write operation.



Inputs:

* From Microcontroller:
  + i\_reset
  + i\_bus\_data [31:0]
  + i\_addr[5:0]
  + i\_r\_neg\_w
  + i\_cs
  + i\_sys\_clk
* From Configuration Registers:
  + i\_reg\_r\_data[31:0]
  + i\_reg\_ack
  + i\_reg\_error

Outputs:

* To Microcontroller:
  + o\_reg\_data[31:0]
  + o\_ack
  + o\_error
* To Configuration Registers:
  + o\_reg\_w\_bus [31:0]
  + o\_r\_neg\_w
  + o\_rs\_vector[30:0] - Array of Register Selects (RS)
    - Bit [0]: o\_srr\_rs
    - Bit [1]: o\_msr\_rs
    - Bit [2]: o\_brpr\_rs
    - Bit [3]: o\_btr\_rs
    - Bit [4]: o\_ecr\_rs
    - Bit [5]: o\_esr\_rs
    - Bit [6]: o\_sr\_rs
    - Bit [7]: o\_isr\_rs
    - Bit [8]: o\_ier\_rs
    - Bit [9]: o\_icr\_rs
    - Bit [10]: o\_txfifoid\_rs
    - Bit [11]: o\_txfifodlc\_rs
    - Bit [12]: o\_txfifodw1\_rs
    - Bit [13]: o\_txfifodw2\_rs
    - Bit [14]: o\_txhpbid\_rs
    - Bit [15]: o\_txhpbdlc\_rs
    - Bit [16]: o\_txhpbdw1\_rs
    - Bit [17]: o\_txhpbdw2\_rs
    - Bit [18]: o\_rxfifoid\_rs
    - Bit [19]: o\_rxfifodlc\_rs
    - Bit [20]: o\_rxfifodw1\_rs
    - Bit [21]: o\_rxfifodw2\_rs
    - Bit [22]: o\_afr\_rs
    - Bit [23]: o\_afir1\_rs
    - Bit [24]: o\_afir2\_rs
    - Bit [25]: o\_afir3\_rs
    - Bit [26]: o\_afir4\_rs
    - Bit [27]: o\_afmr1\_rs
    - Bit [28]: o\_afmr2\_rs
    - Bit [29]: o\_afmr3\_rs
    - Bit [30]: o\_afmr4\_rs

**Tx Priority Logic:**

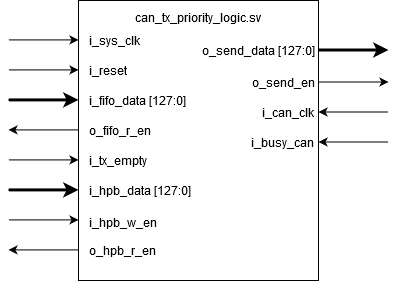
Serves two functions:

1) Priority multiplexer with higher priority for input coming from TX HPB and lower priority for input coming from TX FIFO.

2) Double synchronizer for signals crossing from user side clock to CAN bus clock.

Reading from buffers and writing to the stream processor can only occur when ‘i\_busy\_can’ is set to 0. When ‘i\_busy\_can’ is set to 0, data is fetched from TXHPB and ‘o\_hpb\_r\_en’ is pulsed if ‘i\_hpbfull’ is 1. If ‘i\_hpbfull’ is 0 and ‘i\_tx\_empty’ is 0, data is fetched from TXFIFO and ‘o\_fifo\_r\_en’ is pulsed. If both TXHPB and TXFIFO are empty, then no data is sent to the stream processor. ‘o\_send\_en’ signal is set to 1 when there is data to be sent to the stream processor and the stream processor is not busy.

The module operates normally when ‘i\_cen’ is set to 1. when ‘i\_cen’ is set to 0, it sets ‘o\_send\_en’ to 0 and act as if ‘i\_busy\_can’ is set to 1.



* Inputs:
  + From Wrapper:
    - i\_sys\_clk
    - i\_can\_clk
    - i\_reset
  + From Configuration Registers:
    - i\_ hpbfull
    - i\_hpb\_data[127:0]
    - i\_cen
  + From Tx FIFO:
    - i\_tx\_empty
    - i\_fifo\_data[127:0]
  + From Stream Processor (TX):
    - i\_busy\_can
* Outputs:
  + To Stream Processor (TX):
    - o\_send\_data[127:0]
    - o\_send\_en
  + To Tx FIFO/HPB:
    - o\_hpb\_r\_en
    - o\_fifo\_r\_en

**FIFO:**

**Tx FIFO:**

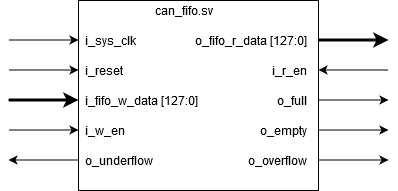
Synchronous FIFO (uses one clock) of parametrizable depth (2,4,8 … 64) and a width of 16 bytes that are filled out as follows:

* + Bytes 15-12: Identifier/Address
  + Bytes 11-8: DLC Field (Bits [31:4] of this field are zeroes)
  + Bytes 7-0: Data field (8 bytes, written MSB First; transmitter will stop after the DLCR[3:0] number of bytes written (maximum of 8 Bytes))
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Configuration Registers:
      * i\_tx\_fifo\_w\_data[127:0]
      * i\_tx\_w\_en
    - From Priority Logic:
      * i\_tx\_r\_en
  + Outputs:
    - To Configuration Registers:
      * o\_tx\_full
    - To Priority Logic:
      * o\_tx\_empty
    - To Priority Logic:
      * o\_tx\_fifo\_r\_data[127:0]

**Rx FIFO:**

Synchronous FIFO (uses one clock) of parametrizable depth (2,4,8 … 64) and a width of 16 bytes that are filled out as follows:

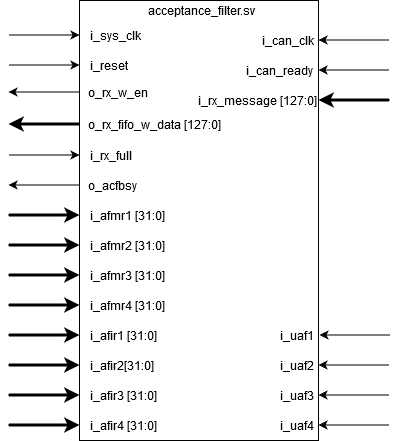
* + Bytes 15-12: Identifier/Address
  + Bytes 11-8: DLC Field (Bits [31:4] of this field are zeroes)
  + Bytes 7-0: Data field (8 bytes, written MSB First; transmitter will stop after the DLCR[3:0] number of bytes written (maximum of 8 Bytes))



* + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Configuration Registers:
      * i\_rx\_r\_en
    - From Acceptance Filter:
      * i\_rx\_w\_en
      * i\_rx\_fifo\_w\_data[127:0]
  + Outputs:
    - To Configuration Registers:
      * o\_rx\_empty
      * o\_rx\_fifo\_r\_data[127:0]
      * o\_rxoflw
      * o\_rxuflw
      * o\_rxok
    - To Configuration Registers and Acceptance Filter:
      * o\_rx\_full

**Acceptance Filter:**

Has two clock inputs: the system clock and the quantum clock. The module does the required synchronization between signals crossing the clock domains. Data and message ready signal come in from the bit stream processor. When a new data frame is available from the processor, ‘i\_can\_ready’ gets pulsed for one quantum clock cycle. If the number of acceptance filters parameter is zero, input data frames get sent to the rx fifand ‘o\_rx\_w\_en’ gets pulsed. If the number of acceptance filters parameter is larger than zero, then depending on UAF bits in the configuration register module, input data ID and AFIR bits get masked using the AFMR mask bits and then get compared. If they are equal, then the data frame gets sent to the rx fifo and ‘o\_rx\_w\_en’ gets pulsed.



* Inputs:
  + From Wrapper:
    - i\_sys\_clk
    - i\_can\_clk
    - i\_reset
  + From Bit Stream Processor:
    - i\_can\_ready
    - i\_rx\_message[127:0]
  + From Rx FIFO:
    - I\_rx\_full
  + From Configuration Registers:
    - i\_uaf1
    - i\_uaf2
    - i\_uaf3
    - i\_uaf4
    - i\_afir1[31:0]
    - i\_afir2[31:0]
    - i\_afir3[31:0]
    - i\_afir4[31:0]
    - i\_afmr1[31:0]
    - i\_afmr2[31:0]
    - i\_afmr3[31:0]
    - i\_afmr4[31:0]
* Outputs:
  + To Rx FIFO:
    - o\_rx\_fifo\_w\_data[127:0]
    - o\_rx\_w\_en
  + To Configuration Registers:
    - o\_acfbsy

**Configuration Register Module:**

Each addressable register is 32 bits wide.



* **Software Reset Register (SRR) (0x000):**

When the bit 31 (SRST) is set to 1, ‘o\_soft\_reset’ is set to 1. The system reset signal is the result of the or operation of ‘o\_soft\_reset’ and the external reset input. When the register is reset, both bit 31 (SRST) and bit 30 (CEN) are set to 0. Reading bit (SRST) always returns 0.

When bit 30 (CEN) is set to 0, the CAN controller transitions into the configuration mode, otherwise the CAN controller goes to one of the other modes depending on the register contents of (MSR).

* + Access: Supports Read and Write.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[0]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Wrapper:
      * o\_soft\_reset
    - To Bit Stream Processor:
      * o\_cen
    - To Priority Logic:
      * o\_cen
* **Mode Select Register (MSR) (0x004):**

This register can set the mode of the CAN controller. There are three modes that are available on this CAN controller: normal mode, sleep mode, and loopback mode. If the 31st bit is set to 1, the mode of the CAN controller is set to loopback mode. If the 32nd bit of this register is set to 1, the mode of the CAN controller is set to sleep mode. If the 31st bit and 32nd bit are both set to 1, then the mode defaults to loopback mode. If the 31st and 32nd bits are not set, then the mode defaults to normal mode.

* + Access: Supports Read and Write. Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[1]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Bit Timing Module:
      * o\_lback
    - To Bit Timing Module:
      * o\_sleep
* **Baud Rate Pre-scaler Register (BRPR) (0x008):**
  + Always outputs 0x0003 to represent derivation of 25MHz clock.
  + Access: Supports Read only.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_rs\_vector[2]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
* **Bit Timing Register (BTR) (0x00C):**

The bit timing register deals with the configuration of the propagation segment, phase segment 1, phase segment 2, and synchronization jump width of the bit time. Bits 23-24 represent the synchronization jump width, bits 25-27 represent phase segment 1, and bits 28-31 represent phase segment 2. The actual value of those fields is one more than the value written to its field in the register. Bits 0-22 are reserved and set to 0.

* + Access: Supports Read and Write.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[3]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Bit Timing Module:
      * o\_sjw[1:0]
      * o\_ts2[2:0]
      * o\_ts1[3:0]
* **Error Counter Register (ECR) (0x010):**

This register stores the error count for both tx and rx. The errors to be counted are as follows: ack error, bit error, stuff error, form error, and crc error. Receive error count is represented by bits 16-23. Transmit error count is represented by bits 24-31. Bits 0-15 are reserved for future purposes. This register is cleared when one of the following occurs: the cen bit in the srr is set to 0, the srst bit in srr is set to 1, when can controller enters into bus off state, or when the CAN controller enters in error active state for 128 occurances of 11 recessive bits during CAN off bus recovery.

* + Access: Supports Read only.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_rs\_vector[4]
    - From Bit Stream Processor:
      * i\_rec[7:0]
      * i\_tec[7:0]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
* **Error Status Register (ESR) (0x014):**

Specifies what error had occurred by latching the low to high transition of the input error from the bit stream processor until the bit is cleared by a reset or by writing a 1 to it. If bit is set to 1 in a field, that error relating to that specific field has occurred. Bit 27 represents an ack error, bit 28 represents a bit error, bit 29 represents a stuff error, bit 30 represents a form error, and bit 31 represents a crc error. If any bit in this register is set, writing a 1 to the set bit clears it.

* + Access: Supports Read and Write-to-Clear.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[5]
    - From Bit Stream Processor:
      * i\_acker
      * i\_berr
      * i\_ster
      * i\_fmer
      * i\_crcer
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
* **Status Register (SR) (0x018):**

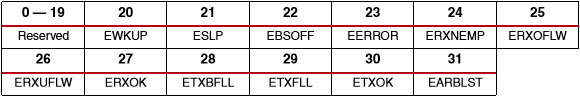
Contain the status of various operations within the CAN controller. The fields are specified in the datasheet, pages 20-21

* + Access: Supports Read only.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_rs\_vector[6]
    - From Acceptance Filter:
      * i\_acfbsy
    - From Tx FIFO:
      * i\_tx\_full
    - From Bit Stream Processor:
      * i\_rec[7:0]
      * i\_tec[7:0]
      * i\_estat[1:0]
      * i\_bbsy
      * i\_bidle
      * i\_normal
      * i\_sleep
      * i\_lback
      * i\_config
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
* **Interrupt Status Register (ISR) (0x01C):**

Sets interrupt bits to 1 on low to high transitions and clears each bit when the corresponding bit in the interrupt clear register is set to 1 or if the bit has a special clear condition as stated in the datasheet.

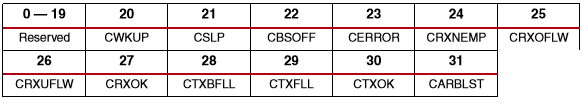
* + Access: Supports Read only.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_rs\_vector[7]
    - From Acceptance Filter:
      * i\_acfbsy
    - From Tx FIFO:
      * i\_tx\_full
    - From Bit Stream Processor:
      * i\_wakeup
      * i\_sleep
      * i\_bsoff
      * i\_error
      * i\_txok
      * i\_arblst
    - From Rx FIFO:
      * i\_rxnemp
      * i\_oflw
      * i\_rxuflw
      * i\_rxok
    - From Tx HPB:
      * i\_hpb\_full
    - From Tx FIFO:
      * i\_tx\_full
    - From Interrupt Clear Register:
      * i\_cwkup
      * i\_cslp
      * i\_cbsoff
      * i\_cerror
      * i\_crxnemp
      * i\_crxoflw
      * i\_crxuflw
      * i\_crxok
      * i\_ctxbfll
      * i\_ctxfll
      * i\_ctxok
      * i\_carblst
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Interrupt Enable Register:
      * o\_wkup
      * o\_slp
      * o\_bsoff
      * o\_error
      * o\_rxnemp
      * o\_rxoflw
      * o\_rxuflw
      * o\_rxok
      * o\_txbfll
      * o\_txfll
      * o\_txok
      * o\_arblst
* **Interrupt Enable Register (IER) (0x020):**

The interrupt enable register enables interrupt generation. Each interrupt input from the status register is AND’d with its enable bit. If at least one interrupt is enabled and asserted, then the ‘IP2BUS\_IntrEvent’ bit is asserted until the interrupt is cleared or disabled. If ‘IP2BUS\_IntrEvent’ signal is high because of more than one interrupt being active, then the signal goes low for one clock cycle and then back to high if one of the interrupts gets cleared or disabled. Below are the bit positions of the enables:



* + Access: Supports Read and write.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Interrupt Status Register:
      * i\_wkup
      * i\_slp
      * i\_bsoff
      * i\_error
      * i\_rxnemp
      * i\_rxoflw
      * i\_rxuflw
      * i\_rxok
      * i\_txbfll
      * i\_txfll
      * i\_txok
      * i\_arblst
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[8]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Wrapper:
      * o\_interrupt
* **Interrupt Clear Register (ICR) (0x024):**

The interrupt clear register clears the bits of the interrupt status register when a 1 is written to one of the corresponding bits.



* + Access: Supports Write only.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_rs\_vector[9]
  + Outputs:
    - To Interrupt Status Register:
      * o\_cwkup
      * o\_cslp
      * o\_cbsoff
      * o\_cerror
      * o\_crxnemp
      * o\_crxoflw
      * o\_crxuflw
      * o\_crxok
      * o\_ctxbfll
      * o\_ctxfll
      * o\_ctxok
      * o\_carblst
* **Transmitter FIFO Input Register (TX FIFO) Group:**

A combination of four addressable 4-byte registers that send a data write pulse to the FIFO when a write operation occurs into the DW2 register. Four registers are laid out as follows:

* + - **TX FIFO ID register (0x030)**

This register contains the identifier field of the can message of the tx. The register fields are defined in the datasheet, pages 28-29.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[10]

* + - * Outputs:
        + To Tx FIFO:

o\_tx\_fifo\_data[127:96]

* + - **TX FIFO DLC register (0x034)**

This register contains the data length code of the tx CAN message. Bits 0-3 specify the data length of the CAN frame. Bits 4-31 are reserved for future purposes.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[11]

* + - * Outputs:
        + To Tx FIFO:

o\_tx\_fifo\_data [95:64]

* + - **TX FIFO DW1 register (0x038)**

This register contains the data word 1 bytes of the tx CAN message. Data word byte 0 (DWB0) is contained in bits 0-7. DWB1 is contained in bits 8-15, DWB2 is contained in bits 16-23, and DWB3 is contained in bits 24-31.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[12]

* + - * Outputs:
        + To Tx FIFO:

o\_tx\_fifo\_data[63:32]

* + - **TX FIFO DW2 register (0x03C)**

This register contains the data word 2 bytes of the tx CAN message. Data word byte 0 (DWB0) is contained in bits 0-7. DWB1 is contained in bits 8-15, DWB2 is contained in bits 16-23, and DWB3 is contained in bits 24-31.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[13]

* + - * Outputs:
        + To Tx FIFO:

o\_tx\_w\_en

o\_tx\_fifo\_data[31:0]

* **Transmitter High Priority Buffer Input Register (TX HPB) Group:**
  + Synchronous register that includes four addressable 4-byte fields that are filled out as follows:
    - Bytes 15-12: Identifier/Address
    - Bytes 11-8: DLC Field (Bits [31:4] of this field are zeroes)
    - Bytes 7-0: Data field (8 bytes, written MSB First; transmitter will stop after the DLCR[3:0] number of bytes written (maximum of 8 Bytes))
  + From this, we derive a register structure similar to the TX FIFO Input register:
    - **TX HPB ID register (0x040)**

This register contains the ID of the message to be written to the TX HPB, the first step in the process of writing to the TX HPB.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[14]

* + - * Outputs:
        + To Priority Logic:

o\_hpb\_data[127:96]

* + - **TX HPB DLC register (0x044)**

This register contains the DLC (Data Length Code) of the message to the TX HPB, the second step in the process of writing to the TX HPB.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_rs\_vector[15]

* + - * Outputs:
        + To Priority Logic:

o\_hpb\_data[95:64]

* + - **TX HPB DW1 register (0x048)**

This register contains the DW1 (Data Word 1) of the message to the TX HPB, the third step in the process of writing to the TX HPB.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[16]

* + - * Outputs:
        + To Priority Logic:

o\_hpb\_data[63:32]

* + - **TX HPB DW2 register (0x04C)**

In addition to containing the second data word, this register implements the full flag for TXHBP, and the input read request from the priority logic. When a write operation is performed on this register, it is assumed that all 128 bits of TXHPB have been written to and ‘o\_hpbfull’ flag is asserted. The full flag is de-asserted only when the system is reset or when ‘i\_hpbreaden’ is pulsed. This is the final step in writing a message to the TX HPB.

* + - * Access: Supports Write only.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus [31:0]

i\_rs\_vector[17]

* + - * + From Priority Logic:

i\_ hpb\_w\_en

* + - * Outputs:
        + To Priority Logic:

o\_hpb\_full

o\_hpb\_data[31:0]

* **Receiver FIFO Output Register (RX FIFO):**

A combination of four addressable 4-byte registers that store a data frame from rx fifo to be read. Initially, it checks rx fifo empty flag, and whenever it becomes 0 it reads one frame and pulses the read enable signal. Then, on every transition from low to high of ‘i\_rs\_vector[21]’, the read enable signal gets pulsed and a new data frame is grabbed from the fifo if it is not empty. Four registers are laid out as follows:

* + - **RX FIFO ID register (0x050)**

This register contains ID of the message to be read from RX FIFO.

* + - * Access: Supports Read.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_rs\_vector[18]

* + - * + From Rx FIFO:

i\_rx\_fifo\_data[127:96]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_data[31:0]

* + - **RX FIFO DLC register (0x054)**

This register contains DLC (Data Length Code) of the message to be read from RX FIFO.

* + - * Access: Supports Read.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_rs\_vector[19]

* + - * + From Rx FIFO:

i\_rx\_fifo\_data[95:64]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_data[31:0]

* + - **RX FIFO DW1 register (0x058)**

This register contains DW1 (Data Word 1) of the message to be read from RX FIFO.

* + - * Access: Supports Read.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_rs\_vector[20]

* + - * + From Rx FIFO:

i\_rx\_fifo\_data[63:32]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_data[31:0]

* + - **RX FIFO DW2 register (0x05C)**

This register contains DW2 (Data Word 2) of the message to be read from RX FIFO.

* + - * Access: Supports Read.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_rs\_vector[21]

* + - * + From Rx FIFO:

i\_rx\_empty

i\_rx\_fifo\_data[31:0]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_data[31:0]

* + - * + To Rx FIFO:

o\_rx\_r\_en

* **Acceptance Filter Register (AFR):** 
  + Defines which acceptance filters to use.
  + If number of acceptance filters is 1: UAF1 bit exists.
  + If number of acceptance filters is 2: UAF1 & UAF2 bits exist.
  + If number of acceptance filters is 3: UAF1, UAF2 & UAF3 bits exist.
  + If number of acceptance filters is 4: UAF1, UAF2, UAF3 & UAF4 bits exist.
  + Access: Supports Read and Write.
  + Inputs:
    - From Wrapper:
      * i\_sys\_clk
      * i\_reset
    - From Microcontroller bus Interface:
      * i\_reg\_w\_bus[31:0]
      * i\_r\_neg\_w
      * i\_rs\_vector[22]
  + Outputs:
    - To Microcontroller bus Interface:
      * o\_reg\_r\_data[31:0]
    - To Acceptance Filter Mask Register 1, Acceptance Filter Identifier Module 1, and Acceptance Filter:
      * o\_uaf1
    - To Acceptance Filter Mask Register 2, Acceptance Filter Identifier Module 2, and Acceptance Filter:
      * o\_uaf2
    - To Acceptance Filter Mask Register 3, Acceptance Filter Identifier Module 3, and Acceptance Filter:
      * o\_uaf3
    - To Acceptance Filter Mask Register 4, Acceptance Filter Identifier Module 4, and Acceptance Filter:
      * o\_uaf4

**Bit Positions:**



* **Acceptance Filter Identifier Register (AFIR) Group**
  + If number of acceptance filters is 1: AFIR1 is used for acceptance filtering.
  + If number of acceptance filters is 2: AFIR1 & AFIR2 are used for acceptance filtering.
  + If number of acceptance filters is 3: AFIR1, AFIR2 & AFIR3 are used.
  + If number of acceptance filters is 4: AFIR1, AFIR2, AFIR3 & AFIR4 are used.
  + This register doesn’t get cleared when a reset or a soft reset occur.
  + When **Extended Frames** are enabled, all bit fields need to be defined.
  + When **Standard Frames** are enabled, only AIID[28:18], AISRR and AIIDE need to be defined. AIID[17:0] and AIRTR should be written to ‘0’.
    - **Acceptance Filter Identifier Register 1 (AFIR1):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[23]

* + - * + From Acceptance Filter Register:

i\_uaf1

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afir1[31:0]

* + - **Acceptance Filter Identifier Register 2 (AFIR2):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf2

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[24]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afir2[31:0]

* + - **Acceptance Filter Identifier Register 3 (AFIR3):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf3

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[25]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afir3[31:0]

* + - **Acceptance Filter Identifier Register 4 (AFIR4):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf4

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[26]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afir4[31:0]

**Bit Positions:**



* **Acceptance Filter Mask Register (AFMR) Group**
  + If number of acceptance filters is 1: AFMR1 is used for acceptance filtering.
  + If number of acceptance filters is 2: AFMR1 & AFMR2 are used for acceptance filtering.
  + If number of acceptance filters is 3: AFMR1, AFMR2 & AFMR3 are used for filtering.
  + If number of acceptance filters is 4: AFMR1, AFMR2, AFMR3 & AFMR4 are used.
  + This register doesn’t get cleared when a reset or a soft reset occur.
  + When **Extended Frames** are enabled, all bit fields need to be defined
  + When **Standard Frames** are enabled, only AMID[28:18], AMSRR and AMIDE need to be defined. AMID[17:0] and AMRTR should be written to ‘0’.
    - **Acceptance Filter Mask Register 1 (AFMR1):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf1

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[27]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afmr1[31:0]

* + - **Acceptance Filter Mask Register 2 (AFMR2):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf2

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[28]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afmr2[28:0]

* + - **Acceptance Filter Mask Register 3 (AFMR3):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf3

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[29]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afmr3[31:0]

* + - **Acceptance Filter Mask Register 4 (AFMR4):**
      * Access: Supports Read and Write.
      * Inputs:
        + From Wrapper:

i\_sys\_clk

i\_reset

* + - * + From Acceptance Filter Register:

i\_uaf4

* + - * + From Microcontroller bus Interface:

i\_reg\_w\_bus[31:0]

i\_r\_neg\_w

i\_rs\_vector[30]

* + - * Outputs:
        + To Microcontroller bus Interface:

o\_reg\_r\_data[31:0]

* + - * + To Acceptance Filter:

o\_afmr4[31:0]

**Bit Positions:**



**Bit Stream Processor**

Normal mode:

Runs at the quantum clock frequency. It grabs the data from the priority logic and transmits the current message serially at the tick of the prescaled quantum clock. Builds a frame as per ISO 11898-2(CAN) standard. Can be built as either standard or extended frame. Sends the serial data to the bit timing module to be put on the Tx bus.

The module checks for errors and increments or decrements the internal rx error counter and tx error counter. Depending on the detected error type, the error bit connected to the error status register is set to 1. The internal counters are used to decide the error state if they exceed a specific value following the CAN standard.

For Rx side, the module grabs data from the timing module at tick of the prescaled quantum clock and removes the stuffed bits, control bits and checks the CRC code. Then derived data length, identifier, and data words are passed onto the acceptance filter and ‘o\_can\_ready’ gets pulsed.

The module is responsible for implementing arbitration by comparing ‘i\_rx\_bit’ and ‘o\_tx\_bit’ according to the arbitration method mentioned in the protocol. If arbitration is lost, the module sends recessive bits and tries again in the next arbitration period.

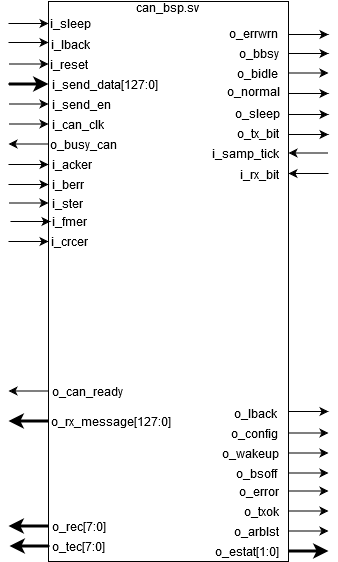
When the module is idle, ‘o\_busy\_can’ is set to 0 and is set to 1 when 1 is received on ‘i\_send\_en’. ‘o\_busy\_can’ stays asserted until arbitration is won and the module finishes sending the data element, at which point ‘o\_busy\_can’ is set to 0, waiting for the next data element from the priority logic.

Loopback mode:

The module runs just like in normal mode but when sending a frame, the acknowledge bit is set as dominant instead of recessive.

Configuration mode:

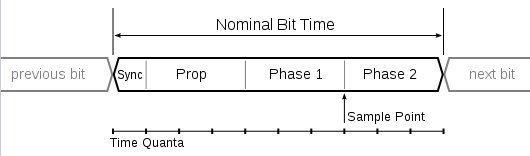
Aborts any sending operation in progress, resets all internal counters and sends recessive bits to the bit timing module.



* Inputs:
  + From Wrapper:
    - i\_can\_clk
    - i\_reset
  + From Configuration Registers:
    - i\_cen
    - i\_sleep
    - i\_lback
  + From Priority Logic:
    - i\_send\_data[127:0]
    - i\_send\_en
  + From Bit Timing Module:
    - i\_samp\_tick
    - i\_rx\_bit
* Outputs:
  + To Priority Logic:
    - o\_busy\_can
  + To Bit Timing Module:
    - o\_tx\_bit
  + To Configuration Register:
    - o\_rec[7:0]
    - o\_tec[7:0]
    - o\_acker
    - o\_berr
    - o\_ster
    - o\_fmer
    - o\_crcer
    - o\_estat[1:0]
    - o\_bbsy
    - o\_bidle
    - o\_normal
    - o\_config
    - o\_txok
    - o\_arblst
    - o\_error
    - o\_bsoff
    - o\_sleep
    - o\_lback
    - o\_wakeup
  + To Acceptance Filter:
    - o\_can\_ready
    - o\_rx\_message[127:0]

**Bit Timing Module:**

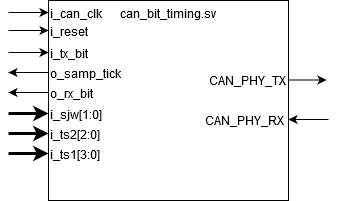
Prescales a CAN\_CLK, a quantum clock that has a period of one time quanta (tq) into a bit time period tick (sampling tick). Additionally, performs hard synchronization as synchronization for the CAN bus, validates writes and informs the stream processor if his write has lower priority.



Synchronization occurs at first transmission of a message bit on the CAN Bus, detected from CAN\_PHY\_RX. The quantum clock period of rising edge of that bit will be marked as the duration of the sync stage. Based off of propagation, as well as time stages 1 and 2, we define a bit window for the time, which subsequently defines the bit transmission frequency. Transmission onto the TX lane will occur at the transition into timing stage 2.

Resynchronization is based on possible drift from the syncronization stage on bit rising edge. If the edge occurs prior to or after the sync stage, we would need to shorten or extend the frame respectively by manipulating the preset propagation as well as timing phase durations.

Max size of send frame is 128 bits extended bit stuffed for worst case scenario = 154 bits.



Inputs:

* From Wrapper:
  + i\_can\_clk
  + CAN\_PHY\_RX
  + i\_reset
* From Configuration Registers:
  + i\_sjw[1:0]sjw[1:0]
  + i\_ts2[2:0]
  + i\_ts1[3:0]
* From Bit Timing Module:
  + i\_tx\_bit

Outputs:

* To Bit Stream Processor:
  + o\_rx\_bit
  + o\_samp\_tick
* To Wrapper:
  + CAN\_PHY\_TX

**DCM for Clock Division (CAN\_CLK generator):**

Takes 100 MHz clock input and outputs a 25 MHz clock to be used to generate the time quanta. Nominal bit time is 1/25 ns, containing ideally 25 time quanta.

Inputs:

* From Wrapper
  + i\_sys\_clk
  + i\_reset
* To Wrapper
  + o\_can\_clk