**Technion**

*Electrical Engineering Department*

High Speed Digital System Lab

**INTERNAL LOGIC ANALYZER**

**Project Documentation**

**Students: Moran Katz**

**Zvika Pery**

**Mentor: Moshe Porian**

**TABLE OF CONTENT**

**INTRO**

GRNERAL DESCRIPTION

Building Internal Logic Analyzer to an FPGA, independent of the manufacturer. The hardware includes a system written in VHDL. It allows recording specific chosen signals by configuration and sending the recorded data back to the user. The software includes a GUI, which allow configuration selection and presentation of the recorded information to the user. Also, building a testability system.

PROJECT REQUIRMENTS

The ILA core will allow:

* Option to choose the parameters :

-trigger type

-trigger position

-number and names of signals

-recording duration

-save and load settings

* Save the recorded information and present it using waveform
* Using resources(logic and memories) independent of the FPGA manufacturer
* Defining smart triggers
* Internal communication is through Wishbone protocol
* External communication is through UART protocol
* Building a system which allows injection of data packages, and compare the recorded data with the requested.
* Designated board is an [Altera DE2 board](http://university.altera.com/materials/boards/de2/) that features an [Altera Cyclone® II 2C35 FPGA](http://www.altera.com/products/devkits/altera/kit-nios-2c35.html).
* System clock will be 100MHz. will be generated by a PLL from a 50MHZ oscillator onboard the DE2 card.
* The FPGA will communicate with its PC host via UART protocol with baud rate of 115,200 bits/sec.
* UART protocol:
  1. Line not active = '1'
  2. 8 bits will be wrapped by *start bit*, represented by '0', and *stop bit*, represented by '1'.
  3. Parity bit will be used can be added after the data bits frame, include 3 options:
     1. Odd - a bit will be added so the total '1' bits will be odd.
     2. Even - a bit will be added so the total '1' bits will be even.
     3. Inhibited.
* Message Pack Structure transferred on UART lines:
  1. SOF - Start of Frame: “00111100” (0x3C)– one byte.
  2. Type – Indicates which client is being accessed - one byte.
  3. Address – Address of the register in a client – one byte.
  4. Length - Number of Bytes of Data - two bytes.
  5. Data – Data written or read from registers in clients or from the FLASH - [length] bytes.
  6. CRC - A check if a successful data transfer was made. CRC will be calculated on the TYPE, ADDRESS, LENGTH and DATA blocks, in that order – one byte.
  7. EOF - End of Frame: “10100101” (0xA5)– one byte.
* Internal Communication using Wishbone protocol.

-Bus width: 8 bits

-Units with wishbone master interfaces: RX path, TX path, CCB

-Units with wishbone slave interfaces: TX path, Wait, LED, VIDEO, LCD, Flash control

-The transactions used are:

* Read single
* Write single
* Read burst
* Write burst
* The reset of the system is filtered, and then sampled in the clock rate.

PROJECT OVERVIEW

The FPGA manufacturers provide Debugging tool, Logic Analyzer, allowing recording of inter data in the FPGA and presenting it to the user. It consist of software and hardware.

The hardware part is in the FPGA code and includes RAM's for storage of the recorded data, configuration changing logic(trigger type etc..), identification of the chosen trigger, and sending the recorded data to the software.

The software part includes GUI, which is allowing the user to choose type of trigger, location of trigger, presentation of the signal's names and of the recorded data.

The Logic Analyzer of the FPGA manufacturer, Altera, is SignalTap. The XILINX tool is calles ChipScope.

## rx path

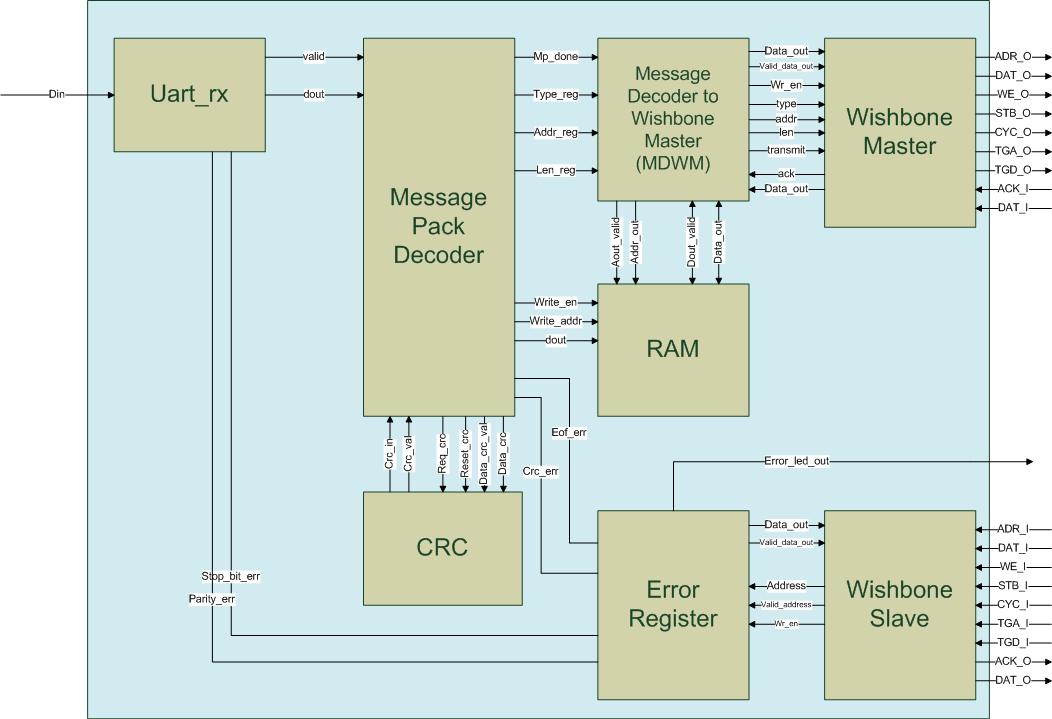


Figure - rx path

General Description

The RX Path processes data received from the host. It unwraps the data before it is transferred to the CCB or other clients of the bus.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| sys\_clk | input | 1 | system clock |
| sys\_reset | input | 1 | system reset |
| rx\_din | input | 1 | input of UART data |
| error\_led\_out | output | 1 | ‘1' when one of the error bits in the register is high |
| ADR\_O | output | addr\_d\_g \* data\_width\_g | contains the address word |
| DAT\_O | output | data\_width\_g | contains the data\_in word |
| WE\_O | output | 1 | '1' for write, '0' for read |
| STB\_O | output | 1 | ‘1' for active bus operation, '0' for no bus operation |
| CYC\_O | output | 1 | '1' for bus transmition request, '0' for no bus transmition request |
| TGA\_O | output | type\_d\_g \* data\_width\_g | contains the type word |
| TGD\_O | output | len\_d\_g \* data\_width\_g | contains the length word |
| ACK\_I | input | 1 | '1' when valid data is received from WS or for successful write operation in WS |
| DAT\_I | input | data\_width\_g | data received from WS |
| ADR\_I | input | addr\_d\_g \* data\_width\_g | contains the address word |
| WE\_I | input | 1 | '1' for write, '0' for read |
| STB\_I | input | 1 | ‘1' for active bus operation, '0' for no bus operation |
| CYC\_I | input | 1 | '1' for bus transmition request, '0' for no bus transmition request |
| TGA\_I | input | type\_d\_g \* data\_width\_g | contains the type word |
| TGD\_I | input | len\_d\_g \* data\_width\_g | contains the length word |
| ACK\_O | output | 1 | '1' when valid data is transmitted to MW or for successful write operation |
| DAT\_O | output | data\_width\_g | data transmit to MW |

Table - RX path interface

The generics of the rx\_path are all its sub-units generics.

The RX path contains the following units:

**UART RX -** This unit receives data via UART protocol. It converts the data received on the UART serial line to an 8 bit vector [dout] and sends it to the mp\_dec (message pack decoder unit). When the data is sent a valid signal is asserted. The uart\_rx also detects two types of errors:

* + - 1. Stop\_bit\_error – if the stop bit is different then ‘1’.
      2. Parity\_bit\_error – if the parity bit is different from the parity result calculated in the unit.

The errors are sent to the error register in the rx\_path and could be read later. An error assertion won’t interrupt the continuation of the data transfer.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | system clock |
| reset | input | 1 | block reset |
| din | input | 1 | UART serial input |
| dout | output | 8 | Parallel data out |
| valid | output | 1 | Parallel data valid |
| parity\_err | output | 1 | parity error |
| stop\_bit\_err | output | 1 | Stop bit error |

Table - Uart rx interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| parity\_en\_g | natural | 0 | 1 to Enable parity bit, 0 to disable parity bit |
| parity\_odd\_g | boolean | False | TRUE = odd, FALSE = even |
| uart\_idle\_g | std\_logic | ‘1’ | IDLE\_ST line value |
| baudrate\_g | positive | 115200 | UART baudrate [Hz] |
| clkrate\_g | positive | 100000000 | Sys. clock [Hz] |
| databits\_g | natural | 8 | Number of databits |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |

Table - Uart rx generics

**Message Pack Decoder (mp\_dec)-** The mp\_dec receives the [dout] vector transferred from the uart\_rx unit and prepares the data before transfer. It detects the start of the transmition (3C\_0x) then it checks in the TYPE, ADDRESS, LENGTH data to registers. The data that will be transferred is saved on the RAM. Finally it requests a CRC check. After receiving the EOF byte (A5\_0x) it asserts the mp\_done signal that informs the mdwm (message decoder to wishbone master) to start the data transfer.

The mp\_dec detects two types of errors:

1. Eof\_error – if the EOF byte is different from A5\_0x
2. Crc\_error – if the CRC error received is different from the one calculated by the CRC block.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | clock |
| rst | input | 1 | reset |
| din | input | width\_g | Input data |
| valid | input | 1 | Data valid |
| mp\_done | output | 1 | Message Pack has been recieved |
| eof\_err | output | 1 | EOF has not found |
| crc\_err | output | 1 | CRC error |
| type\_reg | output | width\_g \* type\_d\_g | type register |
| addr\_reg | output | width\_g \* addr\_d\_g | address register |
| len\_reg | output | width\_g \* len\_d\_g | length register |
| data\_crc\_val: | output | 1 | '1' when new data for CRC is valid, '0' otherwise |
| data\_crc | output | width\_g | Data to be calculated by CRC |
| reset\_crc | output | 1 | '1' to reset CRC value |
| req\_crc | output | 1 | '1' to request for current caluclated CRC |
| crc\_in | input | width\_g \* crc\_d\_g | CRC value |
| crc\_in\_val | input | 1 | '1' when CRC is valid |
| write\_en | output | 1 | 1' = Data is available (width\_g length) |
| write\_addr | output | width\_g \* len\_d\_g | RAM Address |
| dout | output | width\_g | Data to RAM |

Table - mp\_dec interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| clkrate\_g | positive | 100000000 | Sys. clock [Hz] |
| len\_dec1\_g | boolean | True | TRUE - Recieved length is decreased by 1 ,to save 1 bit --  FALSE - Recieved length is the actual length |
| sof\_d\_g | positive | 1 | SOF Depth |
| type\_d\_g | positive | 1 | Type Depth |
| addr\_d\_g | positive | 3 | Address Depth |
| len\_d\_g | positive | 1 | Length Depth |
| crc\_d\_g | positive | 1 | CRC Depth |
| eof\_d\_g | positive | 1 | EOF Depth |
| sof\_val\_g | natural | 60 | (3Ch) SOF block value. Upper block is MSB |
| eof\_val\_g | natural | 165 | (A5h) EOF block value. Upper block is MSB |

Table - mp\_dec generics

**RAM -** A 256 byte RAM.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | clock |
| rst | input | 1 | reset |
| addr\_in | input | addr\_bits\_g | Input address |
| addr\_out | input | addr\_bits\_g | Output address |
| aout\_valid | input | 1 | Output address is valid |
| data\_in | input | width\_in\_g | Input data |
| din\_valid | input | 1 | Input data valid |
| data\_out | output | width\_in\_g | Output data |
| dout\_valid | output | 1 | Output data valid |

Table - RAM interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| clkrate\_g | positive | 100000000 | Sys. clock [Hz] |
| addr\_bits\_g | positive | 8 | Depth of data (2^10 = 1024 addresses) |

Table - RAM generics

**CRC -** A block that calculates the CRC value of the data transferred to it. The mp\_dec uses this block for comparing the CRC value received with the one calculated. The system uses an x^8 polynomial calculation.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | clock |
| rst | input | 1 | reset |
| soc | input | 1 | start of calculation |
| data | input | 8 | data in |
| data\_valid | input | 1 | data in valid |
| eoc | input | 1 | end of calculation |
| crc | output | 8 | crc value |
| crc\_valid | output | 1 | crc value validity |

Table - CRC interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| clkrate\_g | positive | 100000000 | Sys. clock [Hz] |

Table - CRC generics

**Message Decoder to Wishbone Master (mdwm) -** The MDWM operates the rx\_path’s wishbone master in order to send the data received to the intended client. It is the wishbone’s masters director, telling it when to start a transition, and when to end one and what data would be transferred. The mdwm is awakened when the mp\_done signal is asserted by the mp\_dec unit. Then the mdwm reads the TYPE, ADDRESS, LENGTH values and directs it to the wishbone master. After this the unit reads data from the RAM according to the LEN value it has. Every wishbone transaction is ended when ACK\_I signal is asserted. The units operation is according to the following FSM:

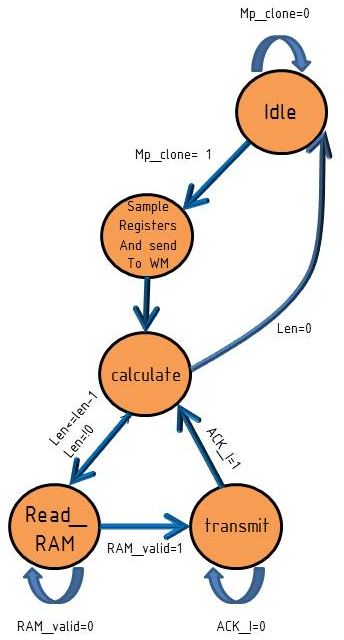


Figure - MDWM FSM

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | clock |
| rst | input | 1 | reset |
| mp\_done | input | 1 | Message Pack Decoder is done when '1' is recieved |
| type\_in | input | type\_d\_g \* data\_width\_g | type is the client which the data is directed to |
| addr\_in |  | addr\_d\_g \* data\_width\_g | the address in the client that the information will be written to |
| len\_in | input | len\_d\_g \* data\_width\_g | length of the data (in words) |
| valid\_RAM\_data | input | 1 | '1' is recieved when the data from RAM is valid |
| RAM\_data | input | data\_width\_g | data recieved from RAM |
| valid\_RAM\_address | input | 1 | '1' is when mdwm requests to read from RAM |
| RAM\_address | input | data\_width\_g | address in RAM to be read |
| data\_out | output | data\_width\_g | data word that will be written to Wishbone client |
| valid\_data\_out | output | 1 | data word is valid |
| wr\_en | output | 1 | '1' for write request, '0' for read request |
| transmit | output | 1 | '1' for wishbone bus transmition request |
| type\_out | output | type\_d\_g \* data\_width\_g | type is the client which the data is directed to |
| addr\_out | output | addr\_d\_g \* data\_width\_g | the address in the client that the information will be written to |
| len\_out | output | len\_d\_g \* data\_width\_g | length of the data (in words) |
| ack | input | 1 | ack is recieved from wishbone master when a successfull transaction ends |

Table - MDWM interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| clkrate\_g | positive | 100000000 | Sys. clock [Hz] |
| type\_d\_g | positive | 1 | Type Depth |
| addr\_d\_g | positive | 3 | Address Depth |
| len\_d\_g | positive | 1 | Length Depth |

Table - MDWM generics

**Wishbone Master -** see wishbone units.

**Wishbone Slave -** see wishbone units.

**Error Register -** Samples an 8 bit vector of error bits. The error vector is sampled every cycle and saved in a register. When the rx\_path’s requests the value of the register it is transferred to it and being set to zero on the following cycle. The unit also asserts the error\_led\_out signal – it is an OR operation on the error register saved bits. If this signal is ‘1’ a led would be lightened, meaning that at least one error has occurred. On the current configuration:

Error\_in[0] – stop\_bit\_error

Error\_in[1] - parity\_error

Error\_in[2] – eof\_error

Error\_in[3] – crc\_error

Error\_in[4..7] – not in use, set to ‘0’

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | clock |
| rst | input | 1 | reset |
| error\_in | input | data\_width\_g | error vector |
| error\_led\_out | output | 1 | '1' when one of the error bits in the register is high |
| data\_out | output | data\_width\_g | data sent to WS |
| valid\_data\_out | output | 1 | validity of data directed to WS |
| address\_in | input | address\_width\_g | address line |
| valid\_in | input | 1 | validity of the address directed from WS |
| wr\_en | input | 1 | enables reading the error register |

Table - Error Register interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| address\_width\_g | natural | 8 | defines the width of the address lines of the system |
| led\_active\_polarity\_g | std\_logic | 1 | defines the active state of the error signal input: '0' active low, '1' |
| error\_register\_address\_g | natural | 0 | defines the address that should be sent on access to the unit |
| error\_active\_polarity\_g | std\_logic | 1 | defines the polarity which the error signal is active in |

Table - Error register generics

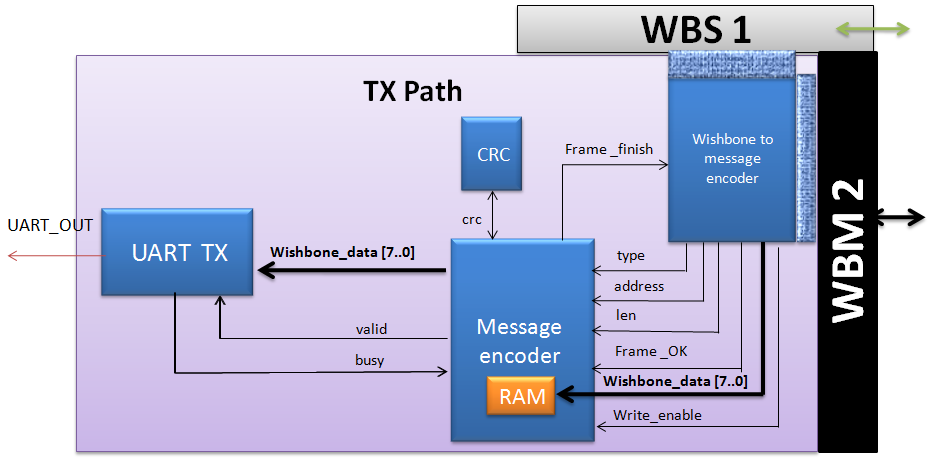
TX path 

Figure - TX path

General Description

The TX is activated when a read transaction is occurring.

The TX (via wishbone slave) is informed that there is a read request, the WTME receive and save the data request and the slave upraise acknowledge . Then the master of the TX is activated and asks to read the required data from the FLASH or from one CLIENT. When the TX master receives the data he wraps it in the form of the message pack structure according to the type, address and length than Transferred over to him.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| Sys\_clk | In | 1 | clock |
| Sys\_reset | In | 1 | Reset. Reset polarity will be set according to the generic parameter 'reset\_polarity\_g' |
| DAT\_I\_S | In | data\_width\_g | The **data** input receiving to the TX slave from the RX master |
| ADR\_I\_S\_TX | In | (addr\_d\_g)\*(data\_width\_g) | The **address**  input receiving to the TX slave from the RX master |
| TGA\_I\_S\_TX | In | (type\_d\_g)\*(data\_width\_g) | The **type** input receiving to the TX slave from the RX master |
| TGD\_I\_S\_TX | In | (len\_d\_g)\*(data\_width\_g) | The **length** input receiving to the TX slave from the RX master |
| WE\_I\_S\_TX | In | write\_en\_polarity\_g | The wishbone Write enable signal transmitted to the TX slave, this signal will be here at ‘0’ (the RX make a write transaction to the TX ) |
| STB\_I\_S\_TX | In | 1 | The wishbone **STB** signal transmitted to the TX slave |
| CYC\_I\_TX | In | 1 | The wishbone **CYC** signal transmitted to the TX slave |
| ACK\_O\_TO\_M | out | 1 | The wishbone **acknowledge**  signal transmitted to the RX master |
| DAT\_O\_TO\_M | Out | data\_width\_g | The data output transferring to the RX master - NOT USED |
| DAT\_I\_CLIENT | in | data\_width\_g | The data input that was read from the CLIENT / FLASH |
| ACK\_I\_CLIENT | in | 1 | The wishbone acknowledge signal from the CLIENT / FLASH |
| ADR\_O\_CLIENT | Out | (addr\_d\_g)\*(data\_width\_g) | The address to begin read in the CLIENT / FLASH |

Table - TX Path interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Generic Parameter** | **type** | **Default value** | **Description** |
| reset\_polarity\_g | Std\_logic | ‘1’ | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| valid\_in\_polarity\_g | Std\_logic | ‘1’ | ‘1’  Value is valid, ‘0’ value isn’t valid |
| write\_en\_polarity\_g | Std\_logic | ‘1’ | defines the polarity which the write\_en is active on: '1' is active high |
| addr\_d\_g | positive | 3 | Address Depth |
| len\_d\_g | positive | 1 | Length Depth |
| type\_d\_g | positive | 1 | Type Depth |
| fifo\_d\_g | positive | 9 | Maximum elements in FIFO |
| addr\_bits\_g | positive | 8 | Depth of data (2^10 = 1024 addresses) |

Table - TX Path generics

**The TX Path contains the following blocks:**

**Wishbone To Message Encoder (WTME):**

The WTME is activated through the RX master in request to read data from the FLASH or from one CLIENT. The RX master transfer the type, address and length of the request the WTME save them in register, upraise acknowledge and activate his own master.

The TX master turn to the FLASH / CLIENT slave and request to read the data. The data is transmitted to the WTME byte by byte (total of LENGTH byte). After all the data was transmitted ‘frame\_ok’ is raise to inform the message encoder to begin to read the data (from the RAM).

WTME FSM:

Figure - WTME FSM

**Message encoder:**

Message Pack Decoder Encoder transmits data from the Type and Address registers, and from the RAM, in a Message Pack format, wraps it and transfer the date to the UART.

This block also produces ‘frame finish’ that signal to the WTME while the present transaction is still running. While frame finish is ‘0’ the WTME won’t try to write new message to the Message Encoder, but only after this signal will turn to ‘1’.

This unit also receives write enable, to allow writing the data to the RAM.

| Pin Name | Direction | Description |
| --- | --- | --- |
| Clk | In | Clock |
| Rst | In | Reset. Reset polarity will be set according to the generic parameter 'reset\_polarity\_g' |
| Fifo\_full | In | FIFO is full, and cannot receive more data from MP Encoder |
| Reg\_ready | In | Input Type, Address and Data Length registers values are ready |
| Type\_reg [width\_g \* type\_d\_g-1..0] | In | Input Type value. Will be valid together with the *reg\_ready* signal |
| Addr\_reg [width\_g \* addr\_d\_g-1..0] | In | Input Address value. Will be valid together with the *reg\_ready* signal |
| Len\_reg [width\_g \* len\_d\_g-1..0] | In | Input Data Length value. Will be valid together with the *reg\_ready* signal |
| Crc\_in [width\_g \* crc\_d\_g-1..0] | In | Calculated CRC value from Checksum block |
| Crc\_in\_val | In | Calculated CRC value from Checksum block is valid |
| Din [width\_g-1..0] | In | Input data (payload), from RAM |
| Din\_valid | In | Input data (payload), from RAM is valid |
| Mp\_done | Out | Message Pack has been successfully transmitted. This flag will be raised together with the EOF output data |
| Dout [width\_g-1..0] | Out | Output data, to the FIFO |
| Dout\_valid | Out | Output data, to the FIFO, is valid |
| Data\_crc\_val | Out | Data to the CRC block is valid |
| Data\_crc [width\_g-1..0] | Out | Data to the CRC, for CRC calculation |
| Reset\_crc | Out | Reset the CRC value to its default value |
| Req\_crc | Out | Request for calculated CRC value |
| Read\_addr\_en | Out | Address to RAM is valid |
| Read\_addr [width\_g \* len\_d\_g-1..0] | Out | Address to RAM |

Table 16 - message encoder signals

| Generic Parameter | type | Default Value | Description |
| --- | --- | --- | --- |
| Reset\_polartiy\_g | Std\_logic | '0' | Reset active in this polarity |
| Len\_dec1\_g | boolean | true | TRUE to receive decreased length by 1. For example: in case actual length is 6, 5 will be received. |
| Sof\_d\_g | positive | 1 | SOF block depth |
| Type\_d\_g | positive | 1 | Type block depth |
| Addr\_d\_g | positive | 3 | Address block depth |
| Len\_d\_g | positive | 2 | Length block depth |
| Crc\_d\_g | positive | 1 | CRC block depth |
| Eof\_d\_g | positive | 1 | EOF block depth |
| Sof\_val\_g | natural | 100 | Initial SOF value (decimal = 64hex) |
| Eof\_val\_g | natural | 200 | Initial EOF value (decimal = C8hex) |
| Width\_g | positive | 8 | Data width (number of bits) |

Table 17 - message encoder generics

**RAM:** a 256 byte RAM. See signal and generics list at the RX Path.

**CRC:** The CRC receive the data from the Message encoder, calculate it’s CRC value, return it to the Message Encoder which insert the CRC value into the UART package. See signal and generics list at the TX Path.

**FIFO**: This block is a general FIFO, it receiving from the message encoder the data and arrange it in a queue before it arrive to the UART.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | system clock |
| reset | input | 1 | block reset |
| din | input | width\_g | Input Data |
| rd\_en | input | 1 | Read Enable (request for data) |
| flush | input | 1 | Flush data |
| dout | output | width\_g | Output Data |
| Dout\_valid | output | 1 | Output data is valid |
| afull | output | 1 | FIFO is almost full |
| full | output | 1 | FIFO is full |
| aempty | output | 1 | FIFO is almost empty |
| empty | output | 1 | FIFO is empty |
| used | output | log\_depth\_g | Current number of elements is FIFO. Note the range. In case depth\_g is 2^x, then the extra bit will be used |

| Generic Parameter | type | Default Value | Description |
| --- | --- | --- | --- |
| Reset\_polartiy\_g | Std\_logic | '0' | Reset active in this polarity |
| width\_g | positive | 8 | Width of data |
| depth\_g | positive | 9 | Maximum elements in FIFO |
| log\_depth\_g | natural | 4 | Logarithm of depth\_g (Number of bits to represent depth\_g. 2^4=16 > 9) |
| almost\_full\_g | positive | 8 | Rise almost full flag at this number of elements in FIFO |
| almost\_empty\_g | positive | 1 | Rise almost empty flag at this number of elements in FIFO |

Table - FIFO signals

Table - FIFO Generics

**UART TX**: receiving the data wrapped from the MESSAGE ENCODER and transmits it out to the HOST.

The UART require rate of 115,200 Kbit/sec.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| clk | input | 1 | system clock |
| reset | input | 1 | block reset |
| din | input | databits\_g | Parallel data in |
| fifo\_din\_valid | input |  | FIFO Ready to transmitte new data to tx |
| fifo\_empty | input | 1 | FIFO is not empty |
| fifo\_rd\_en | output | 1 | Controls FIFO rd\_en |
| dout | output | 1 | Serial data out |

Table - UART\_TX signals

See generics list at the RX Path.

**Wishbone Master -** see wishbone units.

**Wishbone Slave -** see wishbone units.

## Wishbone Blocks and protocol

### Wishbone Protocol

The Internal communication system uses the international Wishbone protocol. The Wishbone Bus is an open source hardware computer bus intended to let the parts of an integrated circuit communicate with each other. The aim is to allow the connection of differing cores to each other inside of a chip.

There are two types of clients on the wishbone bus: master and slave.

1 – wishbone master (WM) – active unit. Initiates bus cycles and ends them.

2 – wishbone slave (WS) - passive unit. Writes or reads data according to master request.

In our system we use the following signals in order to activate the wishbone bus.

CYC – '1' for bus transmition request, '0' for no bus transmition request

STB – '1' for active bus operation, '0' for no bus operation

DAT(sent from WM to WS) – data sent on the bus

ADR – initial address for transaction

TGA – type of client being accessed

TGD – length of the data ( length[data] -1)

WE – write enable

DAT – data sent on the bus

ACK(sent from WS to WM) - successful read or write operation ended by WS.

Wishbone cycle example: In the following waveform a master is writing 5 bytes (TGD=04\_0x) to the Led client which is client number 4 (TGA=04\_0x) . The initial address is ADR=000000\_0x. Notice the assertion of CYC in the beginning of the process and the assertion of STB every new data. The slave asserts ACK=’1’ for one cycle for each data byte.

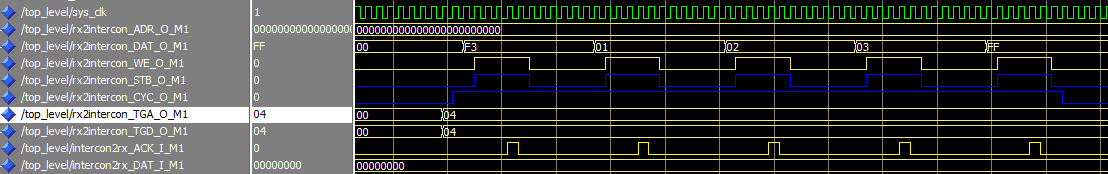


Figure - Wishbone cycle waveform example

Numbering: Each master and slave has a unique number that characterizes it. For the wishbone slaves this number is the TGA signal. (CHANGE NUMBERS)

Wishbone master 1 – RX path

Wishbone master 2 – TX path

Wishbone master 3 – Config Control Block (CCB)

Wishbone slave 1 – RX path

Wishbone slave 2 – TX path

Wishbone slave 3 – Wait client

Wishbone slave 4 – Led client

Wishbone slave 5 – Display client

Wishbone slave 6 – Flash control

Wishbone slave 7 – LCD client (optional)

As seen above there are three masters in the systems. Only one master can make a transaction on the bus. Therefore there is a routing policy implemented in the **wishbone\_intercon** unit.

### Wishbone Intercon

The Wishbone Intercon consists of a router and an arbiter. The router directs the wishbone signals from the operating master to the chosen slave threw a series of muxes. The arbiter is a FSM which enables only one master to use the wishbone bus. Every master that wants to make a transaction asserts CYC. Only the master that is enabled by the arbiter FSM gets its signals passed to the intended slave and can get back an ACK sig.

The arbiter FSM operates as follow. If no master is using the bus than the master that asserts CYC firsts can use the bus. If more than one master requests the bus at the same time, the priority is:

WM 1 -> WM 2 -> WM3

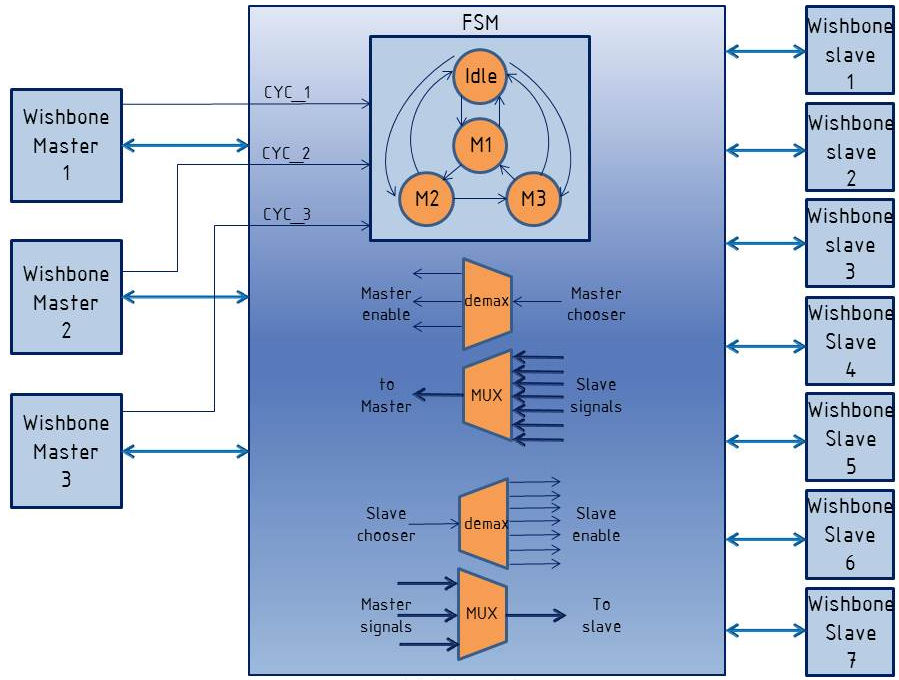


Figure - Wishbone Intercon

### Wishbone Master

The Wishbone Master is the unit that initiates and manages a transaction on the wishbone bus. Although it seems like an active unit, it actually needs to be operated by its host block. For example , the MDWM in the RX path operates WM1. The MDWM reads the data from the RAM and transfers the data to WM1 in order to be transmitted to a chosen slave. The principle is the Wishbone Master is a simple unit that does only its role – transfer data on the wishbone bus. No reading data from RAM or any other operations except the transition.

The Wishbone master operates according to its FSM. There are 2 main branches on this FSM:

Read – for a reading transaction

Write – for a writing transaction

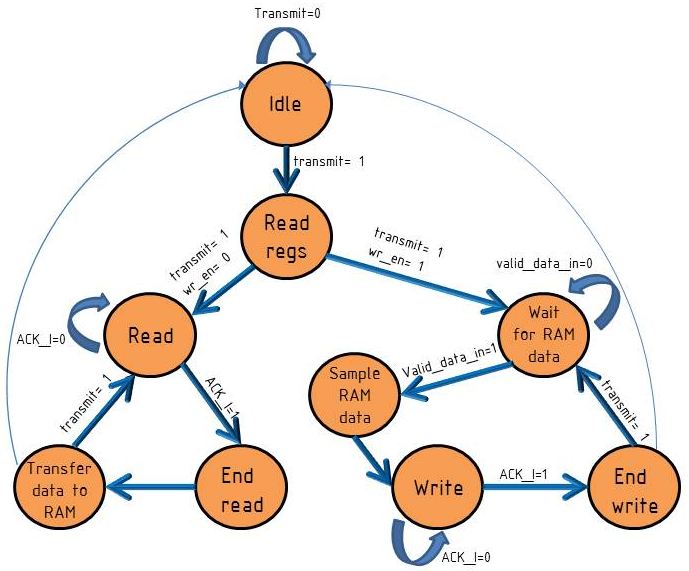


Figure - Wishbone Master FSM

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| sys\_clk | input | 1 | system clock |
| sys\_reset | input | 1 | system reset |
| ADR\_O | output | addr\_d\_g \* data\_width\_g | contains the address word |
| DAT\_O | output | data\_width\_g | contains the data\_in word |
| WE\_O | output | 1 | '1' for write, '0' for read |
| STB\_O | output | 1 | 1' for active bus operation, '0' for no bus operation |
| CYC\_O | output | 1 | '1' for bus transmition request, '0' for no bus transmition request |
| TGA\_O | output | type\_d\_g \* data\_width\_g | contains the type word |
| TGD\_O | output | len\_d\_g \* data\_width\_g | contains the length word |
| ACK\_I | input | 1 | '1' when valid data is recieved from WS or for successfull write operation in WS |
| DAT\_I | input | data\_width\_g | data recieved from WS |
| data\_in | input | data\_width\_g | data to be transmited via bus |
| valid\_data\_in | input | 1 | data validity |
| wr\_en | input | 1 | determines if the WM will make a read('0') or write('1') request |
| transmit | input | 1 | when '1' the units request permission to use the bus by asserting CYC\_O |
| type\_in | input | type\_d\_g \* data\_width\_g | type is the client which the data is directed to |
| len\_in | input | len\_d\_g \* data\_width\_g | length of the data (in words) |
| addr\_in | input | addr\_d\_g \* data\_width\_g | the address in the client that the information will be written to |
| ack | output | 1 | informs the user side that ACK\_I has been asserted |
| data\_out | output | data\_width\_g | data recieved from WS , valid when ack='1' |

Table - Wishbone Master interface

|  |  |  |  |
| --- | --- | --- | --- |
| **generic name** | **type** | **Default value** | **description** |
| reset\_polarity\_g | std\_logic | 1 | '0' = Active Low, '1' = Active High |
| data\_width\_g | natural | 8 | defines the width of the data lines of the system |
| type\_d\_g | positive | 1 | Type Depth |
| addr\_d\_g | positive | 3 | Address Depth |
| len\_d\_g | positive | 1 | Length Depth |

Table - Wishbone Master generics

### Wishbone Slave

Wishbone Slave is the client’s interface to the system. The unit is based on the same principles as the wishbone master. It has the simplest hardware that can handle the wishbone communication. Therefore it only responds to wishbone master signals and passes any information received to its host’s units without any processing. It operates according to the following FSM that has two branches: one for reading cycles and the other for writing cycles.

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **type** | **width (bits)** | **description** |
| sys\_clk | Input | 1 | system clock |
| sys\_reset | Input | 1 | system reset |
| ADR\_I | Input | addr\_d\_g \* data\_width\_g | contains the address word |
| DAT\_I | Input | data\_width\_g | contains the data\_in word |
| WE\_I | Input | 1 | 1' for write, '0' for read |
| STB\_I | Input | 1 | 1' for active bus operation, '0' for no bus operation |
| CYC\_I | Input | 1 | 1' for bus transmition request, '0' for no bus transmition request |
| TGA\_I | Input | type\_d\_g \* data\_width\_g | contains the type word |
| TGD\_I | Input | len\_d\_g \* data\_width\_g | contains the length word |
| ACK\_O | Output | 1 | 1' when valid data is recieved from WS or for successfull write operation in WS |
| DAT\_O | Output | data\_width\_g | data recieved from WS |
| data\_in | Input | data\_width\_g | data that was transmited frome the user side |
| valid\_in | Input | 1 | data validity |
| we\_out | Output | 1 | determines if the WM did a make a read('0') or write('1') requeste |
| type\_out | Output | type\_d\_g \* data\_width\_g | type is the client which the data is directed to |
| len\_out | Output | len\_d\_g \* data\_width\_g | length of the data (in words) |
| addr\_out | Output | addr\_d\_g \* data\_width\_g | the address in the client that the information will be written to |
| user\_ack\_i | Input | 1 | the user uprise this signal in order to force ACK\_O to be '1' |
| cyc\_out | Output | 1 | the CYC\_I signal transfering to the user side |
| stb\_out | Output | 1 | the STB\_I signal transfering to the user side |
| valid\_out | Output | 1 | data output validity |
| data\_out | Output | data\_width\_g | data transmited to the user side |

Table - whishbone slave interface

Wishbone slave generics : same as whishbone master, see there.

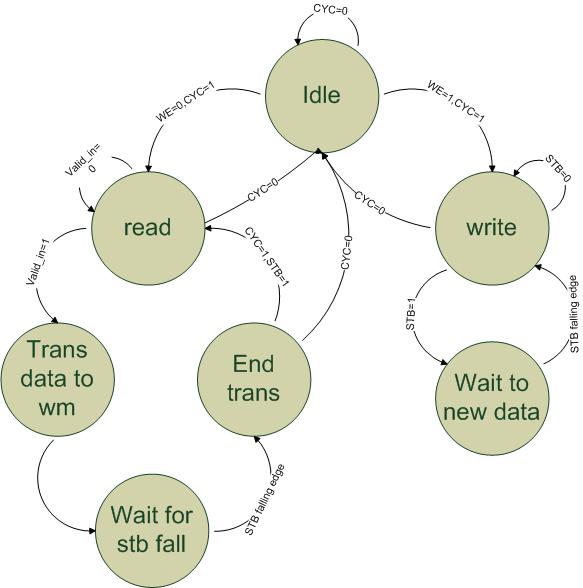


Figure - Wishbone Slave FSM

## 

## Clk & Reset

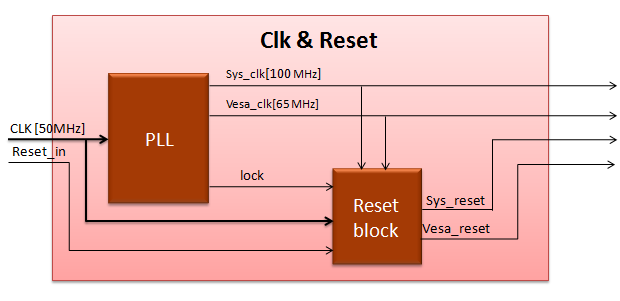


Figure - clk& reset

General Description

1. The Clk & Reset unit would generate the clock and reset signals for the system.
2. Inputs:

* 50MHz clock from DE2.
* Reset signal activated by user.

1. Outputs:

* *Sys\_clk* – 100MHz clock. This is the clock the system would use.
* *Vesa\_clk* – 65MHz clock. This clock would be directed to the Display unit.
* *sys\_reset* - The signal would reset the blocks in the system.
* *Vesa\_reset – the signal would reset the Display unit.*

1. The Clk & Reset block would consist of two sub – blocks:

* **PLL** – This block would receive a 50MHz clock and will generate three signals:

1 – *sys\_clk* (100MHz)

2 *– vesa\_clk* (65MHz)

3 – *lock* – a signal that would be asserted when the clock signals are ready and stable.

* **reset\_blk** – This block would receive the internal *lock* signal and an external *reset\_in*  signal. When both signals are asserted the unit would assert the *sys\_rst and vesa\_rst* signals that would reset the system blocks.

1. The PLL block would be generated by Quartus Megawizard.

Signals

|  |  |  |  |
| --- | --- | --- | --- |
| **signal name** | **Type** | **width (bits)** | **description** |
| clk[50MHz] | Input | 1 | clock from DE2 |
| reset\_in | Input | 1 | reset from switch on DE2 |
| sys\_clk | Output | 1 | clock for system blocks |
| vesa\_clk | Output | 1 | clock for VESA block |
| sys\_reset | Output | 1 | reset for system blocks |
| vesa\_reset | Output | 1 | reset for VESA block |

## Message Pack Description

Figure 9 – Message Pack Structure

### Message Pack Decoder

Message Pack Decoder receives a message pack, built from the following blocks:

1. **SOF** – Start of Frame
2. **Type** – Message type
3. **Address** – Address for the data, or for other purposes
4. **Length** – Data length. Data length CANNOT be less than 1.
5. **Data** **(Payload)** – The wrapped data
6. **CRC** – CRC of Type, Address, Length and Data blocks
7. **EOF** – End of Frame

The MP sniffs the data line, until it receives SOF block. Then it decodes the message: Type, Address and Length will be stored into registers, and will be valid when the EOF is received. Data will be stored into RAM.

Special problematic SOF words are being handled:

Suppose SOF = 0xAABBCC.

A message of 0xAABBAABBCC... will be decode correctly by the MP decoder.

### Message Pack Encoder

Message Pack Decoder Encoder transmits data from the Type and Address registers, and from the RAM, in a Message Pack format:

1. **SOF** – Start of Frame
2. **Type** – Message type
3. **Address** – Address for the data, or for other purposes
4. **Length** – Data length. Data length CANNOT be less than 1.
5. **Data (Payload)** – The wrapped data
6. **CRC** – CRC of Type, Address, Length and Data blocks
7. **EOF** – End of Frame

### Checksum

The Checksum receives data from the Message Pack Decoder / Encoder, and calculates the Checksum for the received data. It is possible to define greater output length than the input length.

**IMPORTAT**: Message Pack uses CRC block. In this project Checksum replaces the CRC block.

### Resources

Required resources, when synthesizing, using Quartus, for Altera's Cyclon II FPGA:

* 4 AND gates
* 2 OR gates
* 179 DFF
* 1 State Machine
* 154 MUX
* 1 Addition Operator
* 1 'Less Than' Operator
* 2 Selector Operator
* 5 Equal Operator

**Maximum Working Frequency**: 180MHz

### Message Pack Decoder - Wave

Figure 10 – Message Pack Decoder and Encoder Wave

The wave is divided into 6 sections:

**Data from UART Generator**

The UART generator generates UART transmission, which is NOT relevant for the MP blocks. MP Decoder receives the data from *din*, together with the *valid* signal.

When SOF (64hex in this example) is being received – the message decoding process is initialized.

**RAM Handshake**

MP Decoder transmits the received payload into the RAM. Data is being transmitted, together with the RAM address and valid signal

**CRC Handshake**

MP Decoder transmits the received Type, Address, Length and Payload data to the CRC block. See Checksum description for handshake explanation.

**Output Registers**

When correct EOF is received, *mp\_dec\_done* flag will be raised. Type, Address and Length will be available from that point.

**Error Flags**

There are two error output flags:

1. **CRC Error** – will be raised in case received CRC and calculated CRC are not equal.
2. **EOF Error** – will be raised in case received EOF and defined EOF (by generic parameter) are not equal. In case such error has occurred – *mp\_dec\_done* flag will not be raised.

**Message Pack Decoder Done**

When correct EOF is received, *mp\_dec\_done* flag will be raised. In case of EOF error – this flag will not be received.

TOP ARCHITECTURE

DATA TRANSFER

1. Setting configuration by the user in the GUI. (trigger type, recording time, signal's number).

Determine in the GUI, transfer through UART IN into RX PATH, from there to the WBI, to the signal generator and in the end the ILA.

1. After we saved our configuration in the ILA, we inject the signal's behavior.

Built in the GUI by the user, going through UART IN into RX PATH,into the WBI and ending in the signal generator.

1. The chosen signals are creating in the signal generator, and transferring into the ILA.
2. The data is transferring out to the user.

The ILA is sampling the signals according the chosen configuration, transfer them to the WBI, from there to the TX PATH, and trough UART OUT back to the GUI and present it to the user.

CORE

**REGISTERS**

**Registers types: (each register is 8 bit )**

1. Type of trigger
2. TRIGGER POSITION- 50%,30%,70%
3. RECORDING TIME
4. Strart- the system will start working after writing data to it

**Status of the system**

1. Read\Write only- counter. Do we write or read at the moment

(control signal- for example MSB, if 1- read, if 0 write)

**RAM**

Every RAM have a 8 data signals input.

The RAM depth is 256 byte, every byte is 8 bit, so one RAM will contain in the worst case (8 signals are sampled) 256 samples for each signal.

For recording time bigger than 256, we will need another RAM.

signal #i will be saved in the i row of the RAM

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **0signal** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **1 signal** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **Signal2** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **3 signal** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **Signal4** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **5 signal** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **Signal6** |
| 128 | 127 | **...** | **...** | 3 | 2 | 1 | **7 signal** |

For a word shorter then 256, we will save every signal in a new row.

For example: recording time of 5 units.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 5 | 4 | 3 | 2 | 1 | **0signal** |
|  |  | 5 | 4 | 3 | 2 | 1 | **1 signal** |
|  |  | 5 | 4 | 3 | 2 | 1 | **Signal2** |
|  |  | 5 | 4 | 3 | 2 | 1 | **3 signal** |
|  |  | 5 | 4 | 3 | 2 | 1 | **Signal4** |
|  |  | 5 | 4 | 3 | 2 | 1 | **5 signal** |
|  |  | 5 | 4 | 3 | 2 | 1 | **Signal6** |
|  |  | 5 | 4 | 3 | 2 | 1 | **7 signal** |

For a longer word then 256, we will save every signal in a following rows.

For example: recording time of 260 units.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **0signal** |
|  |  |  | 260 | 259 | 258 | 257 | **0 signal** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal1** |
|  |  |  | 260 | 259 | 258 | 257 | **1 signal** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal2** |
|  |  |  | 260 | 259 | 258 | 257 | **Signal2** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal3** |
|  |  |  | 260 | 259 | 258 | 257 | **3 signal** |

We can see that we need to use in another RAM for keeping all the data.

As we can see, a waste of space can take place but the advantage is in the simple calculations in the RAM.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **4signal** |
|  |  |  | 260 | 259 | 258 | 257 | **4 signal** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal5** |
|  |  |  | 260 | 259 | 258 | 257 | **5 signal** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal6** |
|  |  |  | 260 | 259 | 258 | 257 | **Signal6** |
| 256 | ... | 5 | 4 | 3 | 2 | 1 | **Signal7** |
|  |  |  | 260 | 259 | 258 | 257 | **7 signal** |

Meaning:



### TX Block

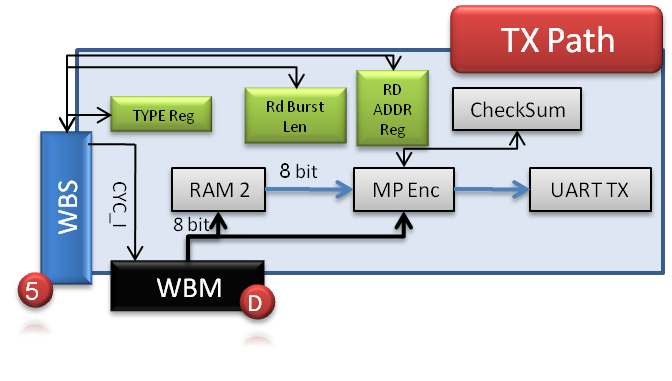


Figure 21– TX Block

The TX Block transmits data for debug purposes, using UART protocol, to the host (Matlab, which is running on a PC).

#### Registers in the TX Block

The following registers will be placed in the TX block:

(W.O = Write Only, R/W = Read and Write)

* **Type Register** – Message type [W.O]
* **Read Burst Length Register** – Number of words to read from SDRAM, in debug mode [R/W]
* **Read Address Register** – Read address from SDRAM / Registers [W.O]

#### Operation

In case of request, from Matlab, from SDRAM / Register, the WBM (A) of RX block will initiate a write transaction to the TX Block, through WBS (5), to command it to initiate a read transaction, which will be according to the Type Register value: transmit data from the SDRAM or registers. The SDRAM address / register will be determined according to the Read Address Register. INTERCON (Y) will decide who will be the active data path, according to the TGC\_O indication from TX Block, which is derived from the Type Register. When WBS(5) CYC\_I will indicate of end of write to registers operation, and start of data transaction, it will command the WBM(D) to start data transmission. The following steps will be executed:

#### Read from SDRAM

WBS (5) will command WBM (D) to initialize a read transaction. SDRAM read transaction will be initiated, through the INTERCON (X) and INTERCON (Y). Read address will be dictated from Read Address Register. Burst length will be dictated from the Burst Length register.

#### Read from Register Block

WBS (5) will command to WBM (D) to initialize a read transaction. Registers read transaction will be initiated, through the INTERCON (X), and through the INTERCON (Z). The required read register will be determined by the ADR\_O[7..0] signal.

#### UART Transmission

The TX Block will initiate a UART transmission to the Matlab, with the following parameters:

1. SOF
2. TYPE – Type of data
3. Address – Read address to SDRAM / Read register address
4. Data Length
5. Data – Data read form SDRAM / Registers.
6. Checksum
7. EOF

### RX Block

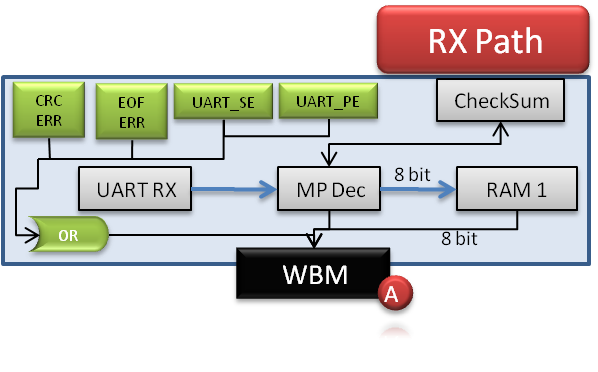


Figure 22– RX Block

When a transmission is being received to the RX block, it is being unwrapped by the Message Pack Decoder. As soon as a full message (EOF has been received) has been received, the *mp\_done* signal will be asserted. Write transaction is being initialized by the RX's WBM (A) through the INTERCON (Z) to the required blocks if the following terms are satisfied:

1. CRC\_ERR = '0' (CRC Error)
2. EOF\_ERR = '0' (EOF has not been received)
3. UART\_SE = '0' (UART Stop Bit Error)
4. UART\_PE = '0' (UART Parity Bit Error)

Two writes modes are possible:

* TGC\_O = '1': Write to Registers
* TGC\_O = '0': Write data to blocks

#### Write to Registers

When TGC\_O = '1': Write to Registers, the addressed register, which is determined by ADR\_I [8..0] should receive and save the received DAT\_I [8..0] value.

When TGC\_O = '0': Write data to blocks, the addressed block, which is determined by ADR\_I [8..0] should receive and store the data in the required target (SDRAM, FIFO, etc…)

#### Internal Registers in RX Block

The following internal, non readable by other blocks, registers will be placed in the RX Block

* **CRC Error** – CRC error has been detected in the RX path for the current message – Clear on Read Register.
* **EOF Error** – EOF error has been detected in the RX path for the current message – Clear on Read Register.
* **UART Stop Bit Error** – Stop bit has not been received in UART RX.
* **UART Parity Error** – Parity bit error has been detected in UART RX.

**ABBREVIATIONS** (יופיע בסוף המסמך)

ILA – Internal Logic Analyzer

RAM - Random Access Memory

TX – Transmission

RX – Receive

WBM – WhishBone Master

WBS – WhishBone Slave