**Technion**

*Electrical Engineering Department*

High Speed Digital System Lab

Modular

Decompression System Documentation

**Version**: 1.3

**Last update**: 14.05.2011

**Written By**: Beeri Schreiber and Alon Yavich

Table of content

[Table of Changes 5](#_Toc293219435)

[1 Intro 6](#_Toc293219436)

[1.1 Abstract 6](#_Toc293219437)

[1.2 Project Goal 6](#_Toc293219438)

[1.3 Project Requirements 6](#_Toc293219439)

[1.4 Compressed and Displayed Image 7](#_Toc293219440)

[2 General Description 8](#_Toc293219441)

[2.1 Data Flow 8](#_Toc293219442)

[2.2 Project Modular Components Scheme 9](#_Toc293219443)

[2.2.1 Implementation in this project 9](#_Toc293219444)

[2.3 Abbreviations 10](#_Toc293219445)

[3 Implementation 11](#_Toc293219446)

[3.1 Top Architecture 11](#_Toc293219447)

[3.2 SDRAM Controller 12](#_Toc293219448)

[3.2.1 General Description 12](#_Toc293219449)

[3.2.2 Supported commands 12](#_Toc293219450)

[3.2.3 Resources 12](#_Toc293219451)

[3.2.4 Burst Write – Wave 13](#_Toc293219452)

[3.2.5 Burst Read – Wave 13](#_Toc293219453)

[3.2.6 SDRAM Documentation 13](#_Toc293219454)

[3.3 UART Transmitter and Receiver 14](#_Toc293219455)

[3.3.1 UART Characteristics 14](#_Toc293219456)

[3.3.2 Resources 14](#_Toc293219457)

[3.3.3 RX Operation – Wave 15](#_Toc293219458)

[3.3.4 TX Operation - Wave 15](#_Toc293219459)

[3.3.5 UART Documentation 15](#_Toc293219460)

[3.4 Message Pack Description 16](#_Toc293219461)

[3.4.1 Message Pack Decoder 16](#_Toc293219462)

[3.4.2 Message Pack Encoder 16](#_Toc293219463)

[3.4.3 Checksum 17](#_Toc293219464)

[3.4.4 Resources 17](#_Toc293219465)

[3.4.5 Message Pack Decoder - Wave 17](#_Toc293219466)

[3.4.6 Message Pack and Checksum Documentation 18](#_Toc293219467)

[3.5 VESA Generator Controller 19](#_Toc293219468)

[3.5.1 DVI Operation 19](#_Toc293219469)

[3.5.2 VESA Generator Operation 20](#_Toc293219470)

[3.5.3 Frame 20](#_Toc293219471)

[3.5.4 Defenition of Terms 21](#_Toc293219472)

[3.5.5 Resources 21](#_Toc293219473)

[3.5.6 Complete Frame (Image) - Wave 21](#_Toc293219474)

[3.5.7 VESA Generator Documentation 21](#_Toc293219475)

[3.6 Global Nets 22](#_Toc293219476)

[3.6.1 Resources 22](#_Toc293219477)

[3.6.2 Global Nets - Wave 22](#_Toc293219478)

[3.6.3 Clock Block 23](#_Toc293219479)

[3.6.4 Reset Block 23](#_Toc293219480)

[3.6.5 Clock and Reset Documentation 25](#_Toc293219481)

[3.7 Top Architecture Details 26](#_Toc293219482)

[3.7.1 Memory Block 26](#_Toc293219483)

[3.7.2 Image Block 29](#_Toc293219484)

[3.7.3 TX Block 31](#_Toc293219485)

[3.7.4 RX Block 33](#_Toc293219486)

[3.7.5 Interconnects 34](#_Toc293219487)

[3.7.6 Wishbone Cycles 35](#_Toc293219488)

[3.8 Project's Directory Structure 37](#_Toc293219489)

[4 Testing 38](#_Toc293219490)

[4.1 Simulation 38](#_Toc293219491)

[4.2 Lab Examination 38](#_Toc293219492)

[4.2.1 Matlab's GUI 38](#_Toc293219493)

[4.2.2 Algorithm Example from Matlab 39](#_Toc293219494)

[5 Summary 40](#_Toc293219495)

[5.1 Project Usage 40](#_Toc293219496)

[5.2 Project Status 42](#_Toc293219497)

[5.3 Problems during the project 42](#_Toc293219498)

[5.4 Project Conclusions 43](#_Toc293219499)

[5.5 Coding Conclusions 43](#_Toc293219500)

Table of Tables

Table 1 – Table of changes 4

Tables of Figures

[Figure 1 – Compressed and Displayed Image 7](#_Toc276811633)

[Figure 2 – General Scheme 8](#_Toc276811633)

[Figure 3 – Detailed Scheme 9](#_Toc276811634)

[Figure 4 - Top Architecture 11](#_Toc276811633)

[Figure 5 – Write Burst 13](#_Toc276811634)

[Figure 6 – Read Burst 13](#_Toc276811634)

[Figure 7 – UART RX Operation 15](#_Toc276811634)

[Figure 8 – UART TX Operation 15](#_Toc276811634)

[Figure 9 – Message Pack Structure 16](#_Toc276811634)

[Figure 10 – Message Pack Decoder and Encoder Wave 17](#_Toc276811634)

[Figure 11 – Frame of Image 20](#_Toc276811634)

[Figure 12 – Defenition of Terms 21](#_Toc276811634)

[Figure 13 – VESA Complete Frame 21](#_Toc276811634)

[Figure 14 – Global Nets Hirarchy 22](#_Toc276811634)

[Figure 15 – Gloabal Nets Wave 22](#_Toc276811634)

[Figure 16 – Reset Block Wave 23](#_Toc276811634)

[Figure 17 – Reset filter working Method 24](#_Toc276811634)

[Figure 18 – Sync Reset Generator working Method 25](#_Toc276811634)

[Figure 19 – Memory Block 26](#_Toc276811634)

[Figure 20 – Image Block 29](#_Toc276811634)

[Figure 21 – TX Block 31](#_Toc276811634)

[Figure 22 – RX Block 33](#_Toc276811634)

[Figure 23 – Start Burst Write Cycle 36](#_Toc276811634)

[Figure 24 – End Burst Write Cycle 36](#_Toc276811634)

[Figure 25 – Directory Structure 37](#_Toc276811634)

[Figure 26 – Matlab's GUI 38](#_Toc276811634)

[Figure 27 – Algorithm Example 39](#_Toc276811634)

[Figure 28 – Possible Usage of the project in Cellular Phones 40](#_Toc276811634)

[Figure 29 – Possible Usage of the project in Hospital Monitors 40](#_Toc276811634)

[Figure 30 – Possible Usage of the project in Engraving Machines 41](#_Toc276811634)

[Figure 30a – 3D 'A' Shape 41](#_Toc276811634)

# Table of Changes

|  |  |  |
| --- | --- | --- |
| Version | Date of Change | Description |
| 1.0 | 11.12.2010 | Creation of documentation |
| 1.1 | 18.03.2011 | Unit Descriptions added |
| 1.2 | 7.05.2011 | Top Architecture added |
| 1.3 | 14.05.2011 | Re-arrangements |

Table 1 – Table of Changes

# Intro

## 1.1 Abstract

Transferring data between different media are essential and integral operation these days.

Data transmission time and the resources needed to process the information is one of the main issue rises while transferring information between media.

Data compression is commonly used in various applications:

1. Systems, where the **memory capacity** is limited.
2. Systems, which the **communication bandwidth** is limited.

## Project Goal

Implement a software compressor (Matlab) with a hardware extractor (FPGA). The project targets to create a scalable **modular generic** system, which can be operated with *various* communication protocols, *various* memory storage devices, *various* compression algorithms and *various* display standards.

Implementing a data compression system on *hardware* has many benefits, such as:

1. **Minimum resources** are required for the implementation.
2. **Reduced physical size** (Board with only communication components, memory storage devices, image-decoders and FPGA are smaller than a PC).
3. **Reduced power consumptions**, since the system is dedicated for a specific operation.
4. **Data Processing capability** is much higher, when compared to a SW based solution.

## Project Requirements

1. Transmit 640x480 BMP grayscale picture, using Matlab, to the DE2 board. JPG, PNG and GIF are also supported, though their compression time will be longer. The picture transmission will be cyclic, according to a given list, provider by user.
2. Maximum number of color repetition will be limited to 128 (7Fh + 1). Repetition will be represented as number of repetition – 1. For example: transmitted 0 repetitions mean 1 repetition.
3. A black screen will be displayed, until the first frame will be transmitted.
4. Data will be sent from host to the DE2 board using UART 115,200 bits/sec.
5. UART protocol:
   1. Line not active = '1'
   2. Parity bit will not be used, since checksum is being calculated
   3. 8 bits will be wrapped by *start bit*, represented by '0', and *stop bit*, represented by '1'.
6. Message Pack Structure:
   1. SOF – Start of Frame
   2. Type – Message Type (Data, Debug, summary chunk…)
   3. Address – registers address
   4. Data (payload) Length – number of bytes – 1 in data block
   5. Data (payload)
   6. CRC (Checksum will be used instead CRC)
   7. EOF – End of Frame
7. Main types ('b' in the Message Pack Structure):
   1. Compressed Data
   2. Last Packet Summary (End of packet) – contains the number of bytes, which had been transmitted for this picture (Size of compressed image in bytes)
   3. Reading / Writing registers
   4. Reading and writing from SDRAM for debug
   5. More types will be added later, in case such are required
8. SDRAM – Reading / Writing in up to 256 words burst.
9. Required size of one picture in SDRAM: 640x480x2x2 bytes (x2: one picture is read, while the other one is being; x2: Worst case of compression)
10. In case an error has occurred during transmission, next picture will be written to the same bank in memory, and the last fine picture will keep being transmitted to the screen. The corrupted picture will be discarded.
11. SDRAM Arbiter priority: Writing, then reading, in order to prevent data loss.
12. Clocks in systems:
    1. 40 MHz for VESA
    2. 133MHz for SDRAM
13. Checksum will be calculated on the TYPE, ADDRESS, LENGTH and DATA blocks, in that order.
14. During blanking, three horizontal lines will be written to FIFO.
15. 640x480 pictures will be compressed and transmitted. 800x600 pictures will be displayed (640x480 pictures, wrapped by black frame).

## Compressed and Displayed Image

Figure 1 – Compressed and Displayed Image

# General Description

The project is **modular**. It is composed of blocks, which are implemented in compliance with ***Wishbone*** standard. Each block is composed of *generics* IPs, therefore, it can be **re-used** with different parameters in other projects.

The project is **scalable**. Multiple units of the *modular de-compressor*, each one with different configuration and different protocols, can operate on the same FPGA.

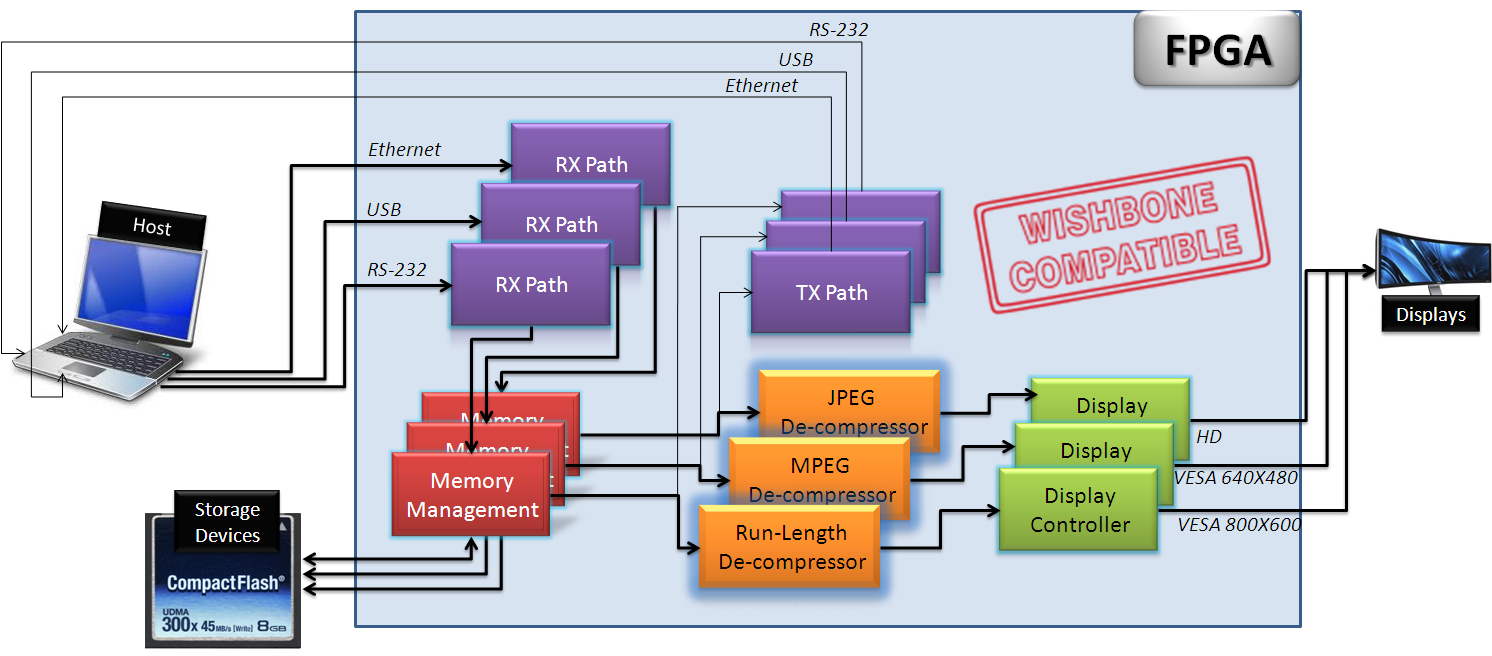


Figure 2 - General Scheme

## Data Flow

**Host** transmits wrapped message, composed of compressed data, to the **RX Path**. Message is decoded, transmitted to the **Memory Management** block, and stored into **Storage Device**. Data is read from the memory, de-compressed by the **De-Compressor**, and transmitted to the **Display** through the **Display Controller**. Status and debug signals can be sent through the **TX Path** to the **Host**.

## Project Modular Components Scheme

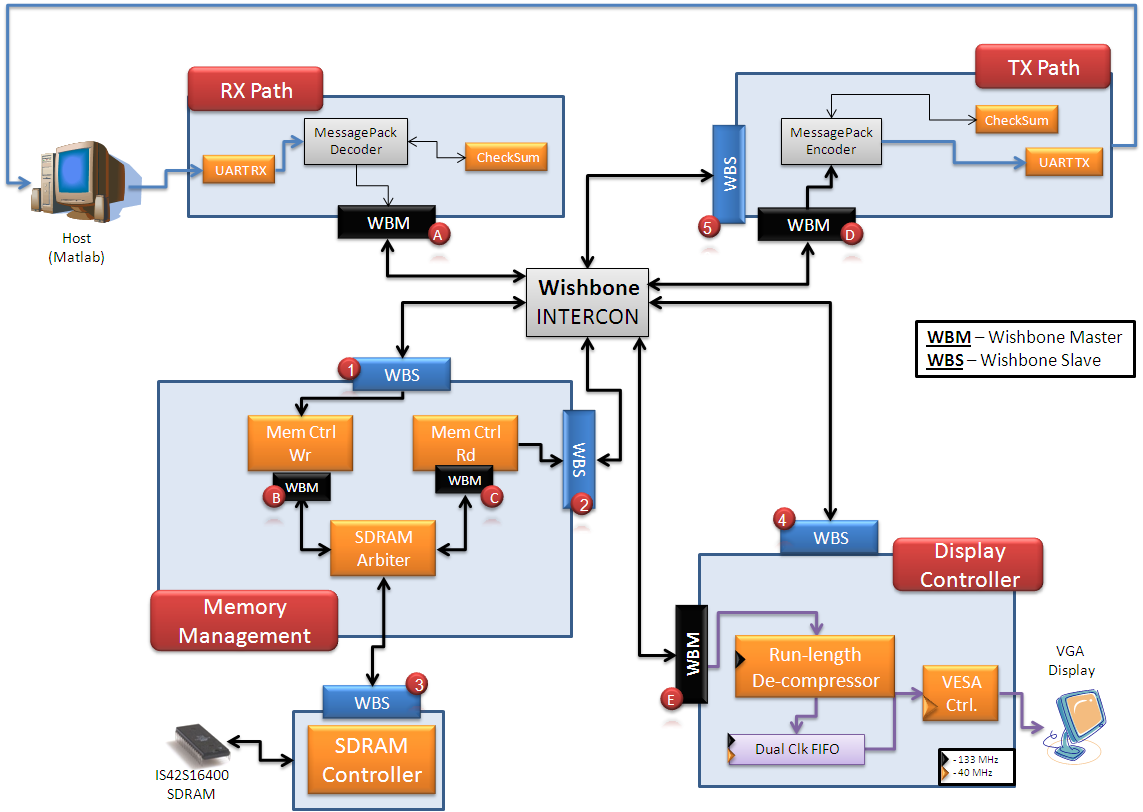


Figure 3 - Detailed Scheme

### Implementation in this project

Image is compressed using **Improved *Run-length******algorithm*** (Repetition of up to 128 consecutive **identical horizontal pixels** orup to 128 consecutive **identical vertical lines** are being searched for) in *Matlab* environment and transmitted, as a wrapped message, to the Altera DE2 board, through **UART** protocol.

The FPGA decodes the wrapped message (**Message Pack Decoder**), validates **Checksum** and correctness of message length, and stores the data into **SDRAM**, through the **Memory Management** block. The compressed image is loaded from the SDRAM, and then a **de-compressor**, located within the **Display Controller**, decompresses the image and transmits it to the **VGA** **display**.

For debug purposes, *status signals* and *data from SDRAM* can be read, and transmitted as a wrapped message, through the **TX Path** to the **host**.

Modularity:

Each one of the  components in *figure 2* may be replaced with another component as required, according to the following list:

**RX Path** and **TX Path**

1. *UART RX* and *TX* components may be replaced with any other protocol. Alternately, may be operated in other system *clock frequency* and other *baud rate*.
2. *Checksum* component may be replaced with other data integrity check component.
3. SOF and EOF (**S**tart and **E**nd **o**f **F**rame) of wrapped message may be any value with any width.

**Display Controller**

1. Run-length *de-compressor* may be replaced with other decompression algorithm component.
2. *VESA Controller* may be replaced with other video standard controller. Alternately, it may be operated with any of the non-interlaced VESA standards, according to generic parameters.

**Memory Management**

1. *Memory Write / Read controller* and *Arbiter* may be replaced with different memory control system.
2. SDRAM controller may be replaced with any other storage device controller.

## 

## Abbreviations

1. SDRAM – Synchronous Dynamic Random Access Memory
2. RAM – Random Access Memory
3. TX – Transmission
4. RX – Receive
5. FIFO – First in First out
6. PLL – Phased Locked Loop
7. TB – Test bench
8. SOF – Start of frame
9. EOF – End of frame
10. CRC - Cyclic redundancy check
11. MP – Message Pack
12. TB – Test Bench
13. UART – Universal Asynchronous Receiver Transmitter
14. VESA - Video Electronics Standards Association
15. VGA - Video Graphics Array
16. DVI - Digital Visual Interface
17. IP – Intellectual Property

# Implementation

This section contains general information about the coded components in this project. For more information, refer to the component documentation. A link to the documentation can be found under the general description of each component.

## Top Architecture

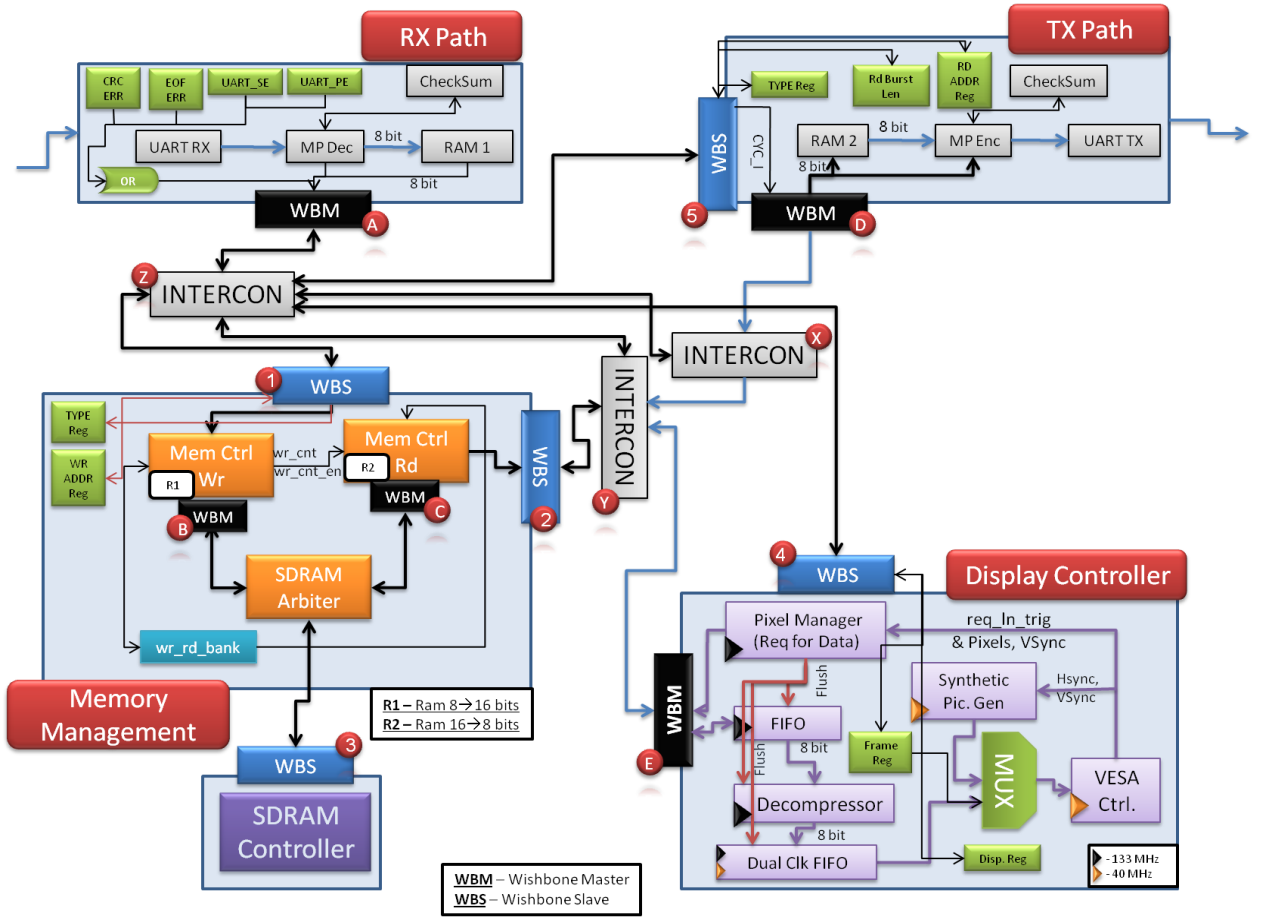


Figure 4– Top Architecture

For detailed information refer to [Top Architecture Details](#_Top_Architecture_Details).

## SDRAM Controller

### General Description

The SDRAM controller implements the IS42S16400 SDRAM Controller, with the following characteristics:

1. Row width: 12 bits
2. Column width: 8 bits
3. Bank width: 2 bits
4. Address structure:
   1. Bank (21 downto 20)
   2. Row (19 downto 8)
   3. Column (7 downto 0)
5. CAS Delay = 3 (required for 133MHz clock)
6. Burst Length = Full Page (256 words - cyclic)
7. 4096 refreshes cycles will be automatically execute per each 64 ms.
8. Maximum read/write burst length is 256. In case 256-(column address) is less than the burst length – only 256-(column address) words will be written / read, and an error will be reported (WBS\_ERR\_O signal).

The SDRAM Controller is *Wishbone Compatible* with one exception: Given address at start of burst is auto-incremented by SDRAM, independently in the following given addresses (WBS\_ADR\_I).

### Supported commands

Two ACT commands are supported by this controller:

1. Write burst command – up to 256 words (512Kbyte)
2. Read burst command – up to 256 words (512Kbyte)

The SDRAM controller is **Wishbone Compatible**.

### Resources

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

* 16 Buffers (Directional)
* 11 AND gates
* 17 OR gate
* 146 DFF
* 375 MUX
* 32 I/O Buffer (TRI)
* 2 State Machines
* 9 Addition operator
* 4 Equal Operator
* 8 Less than operator
* 33 Selector Operator

**Maximum Working Frequency**: 206MHz

### Burst Write – Wave

Figure 5 - Write Burst

### Burst Read – Wave

Figure 6 - Read Burst

### SDRAM Documentation

[https://runlen.googlecode.com/svn/branches/Doc/SDRAM/SDRAM\_doc.docx](https://runlen.googlecode.com/svn/branches/Doc/SDRAM/SDRAM_doc.docx%20)

## UART Transmitter and Receiver

This component is used for asynchronous serial data channel.

The receiver converts serial start bit, data, parity and stop bit to parallel data.

The transmitter converts parallel data into serial form and automatically adds start bit, parity and stop bit.

The data word length can be 5-8 bits, according to generic parameter. Parity bit can be odd or even or if decided can be inhibited, according to generic parameters.

All inputs and outputs are synchronized with the positive edge of the clock.

Any system clock and any baud rate are supported, according to generic parameter.

### UART Characteristics

The UART is implemented with 2 components:

1. UART RX – UART Receiver
2. UART TX – UART Transmitter

Both of the components are based on the same characteristics, the parameters are generics:

1. **Start bit**: '1' or '0' according to user's choice (*uart\_idle\_g*).
2. **Data length**: 5-8 data bits (between Start and stop bit).
3. **Parity bit**: can be added after the data bits frame, include 3 generic 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.
4. **Stop bit**: opposite of Start bit, 1 bit in length.
5. **Reset**: Reset polarity can be chosen.
6. **Baud Rate**: Transmission rate.
7. **System Clock**: 133MHz

### Resources

#### RX

* 6 AND gates
* 6 OR gate
* 41 DFF
* 123 MUX
* 1 State machine
* 4 additional operators
* 7 'Less Then' operators
* 24 selector operators
* 4 Equal operators

Maximum Frequency: 160MHz

#### TX

* 1 AND gates
* 28 DFF
* 66 MUX
* 1 State machine
* 2 additional operators
* 2 Equal operators

 Maximum Frequency: 300 MHz

### RX Operation – Wave

Figure 7 – Uart RX Operation

This simulation shows the RX operation in all its stages. Detecting the start bit afterwards 8 data bits and a stop bit. Valid flag rises to high level when whole word received and *dout* has the relevant parallel data.

### 

### TX Operation - Wave

Figure 8 – Uart TX Operation

This simulation shows the TX operation in all its stages. Generating the start bit and after FIFO Valid flag is in a high level transmitting the data itself, At the end generating parity bit ( if needed ) and stop bit.

### UART Documentation

<https://runlen.googlecode.com/svn/branches/Doc/UART/UART_doc.docx>

## 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.

### Message Pack and Checksum Documentation

<https://runlen.googlecode.com/svn/branches/Doc/Message_Pack_and_Checksum/Message_Pack_and_Checksum.docx>

## VESA Generator Controller

### DVI Operation

A DVI video signal contains 6 signals:

* Clock – Pixel Clock
* HSync – New Line signal
* VSync – New Frame (image) signal
* Blank – Blanking signal to the screen (ignore the RGB inputs)
* RED – Analog signal, used to control the color
* GREEN – Analog signal, used to control the color
* BLUE – Analog signal, used to control the color

Colors are produced by changing the analog levels of the three RGB signals.

The screen refresh process begins in the top left corner and paints 1 pixel at a time from left to right. At the end of the first row, the row increments and the column address is reset to the first column. Once the entire screen has been painted, the refresh process begins again.

The video signal must redraw the entire screen 60 times per second to provide motion in the image and to reduce flicker. This period is called the refresh rate.

In 800X600 pixel mode, with a 60 Hz refresh rate, this is approximately 25 ns per pixel, which requires a 40 MHz.

The **VSync** signal commands the monitor to start displaying a new frame, and the monitor starts in the upper left corner with pixel (0, 0).

The **HSync** signal commands the monitor to refresh another row of 800 pixels.

After 600 rows of pixels are refreshed with 600 HSync signals, a VSync signal resets the monitor to the upper left corner and the process continues.

During the time when pixel data is not being displayed (**Blanking**) and the beam is returning to the left column to start another horizontal scan, the RGB signals should all be set to black color (all zero).

### VESA Generator Operation

VESA Generator is a VESA Non-Interlaced Controller, which transmit to the DVI screen the following data and signals:

* (R, G, B) Pixels
* Horizontal and Vertical Sync
* Blanking

**IMPORTANT:** The DVI should be supplied by pixel-clock. The VESA generator does notsupplies the pixel-clock. The user shall ensure that the DVI has the same clock input as the VESA generator.

The (R, G, B) Pixels should be supplied to the VESA generator by an outside component.

Back Porch, Front Proch, Left - Right - Upper - Lower borders, Sync Time and Active pixels / lines should be defined, according to the VESA standard, using the generic parameters.

The R, G and B port sizes may be changed individually, using a generic parameter.

The R, G and B values will be transmitted, one by one, in the each pixel-clock. See clock explanation in the DVI operation (40MHz in the example).

### Frame

A generic-color frame can wrap the transmitted image, using the left, right, lower and upper input registers.

Figure 11 – Frame of Image

### Defenition of Terms

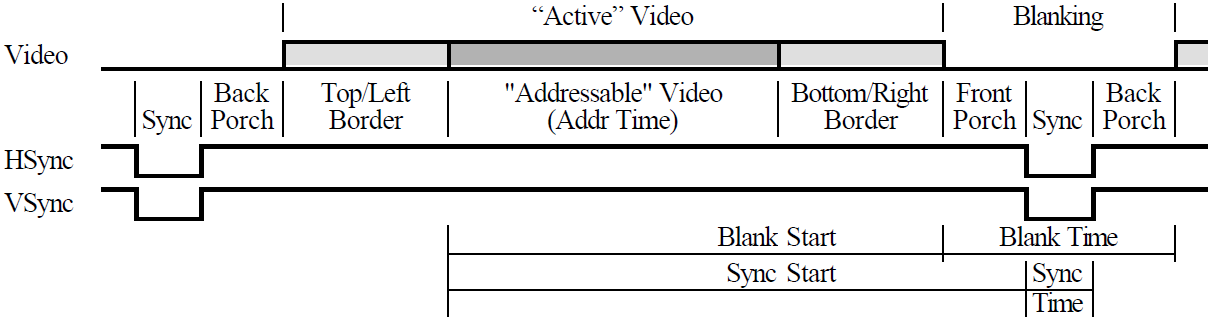


Figure 12 – Definition of Terms

### Resources

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

* Combinational logic cells: 161
* DFF – 68

**Maximum frequency**: 171MHz

### Complete Frame (Image) - Wave

Figure 13 – VESA Complete Frame

In the wave above, one frame (image) is being transmitted, wrapped by two VSync signals.

### VESA Generator Documentation

<https://runlen.googlecode.com/svn/branches/Doc/VESA/VESA_Generator.docx>

## Global Nets

These are the global nets in the FPGA system. In is consumed of:

1. **Clock Block Top** – which responsible to create the required clocks in the system, using PLL
2. **Reset Block Top** – which responsible to filter the FPGA reset, and create synchronized reset de-assertion to each clock, with PLL-locked dependent.

Figure 14 – Global Nets Hirarchy

### Resources

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

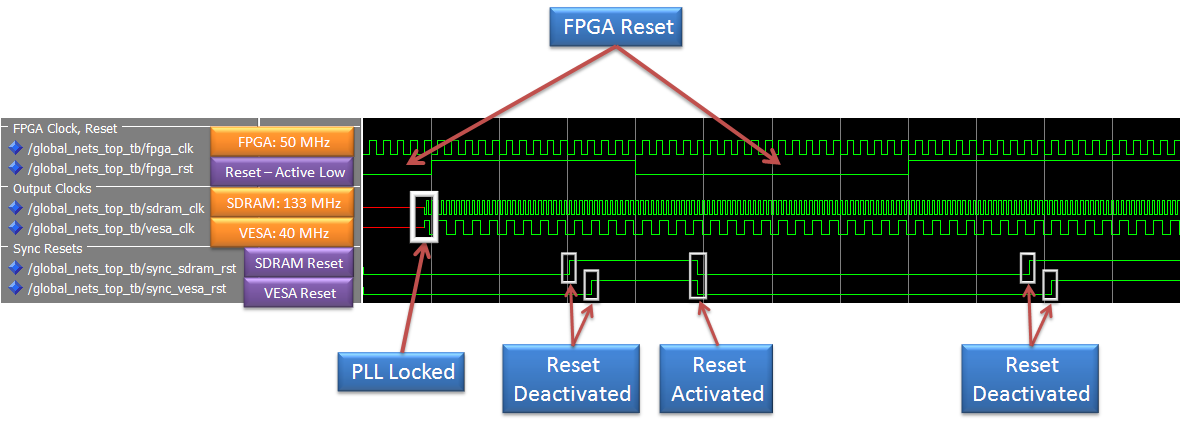
* 5 AND gates
* 1 OR gate
* 18 DFF
* 4 MUX
* 1 PLL

**Maximum Working Frequency**: 610MHz

### Global Nets - Wave

The waves are divided into 3 sections:

1. **FPGA Clock and Reset** – Input 50MHz clock, Asynchronous reset
2. **Output Clocks** – SDRAM Clock (133MHz) and VESA Clock (40MHz)
3. **Synchronous Resets** –Synchronous resets to the SDRAM and VESA clocks

 Figure 15 – Global Nets Wave

In the above wave, both SDRAM and VESA clocks are available as soon as the PLL is locked. Note that the reset deactivation ('0' = active, '1' = inactive) occurs in a delay, since it is being filtered and matched to each clock. The reset activation occurs at the same time to all clock domains, after filtration.

### Clock Block

The clock block generates the required clocks to the system, from an input clock. It is consumed from PLL instantiation only.

### Reset Block

The Reset block is consumed of two blocks:

1. **Reset Filter** – Filters the reset from the FPGA and the PLL-locked signal, and generate synchronized reset from both. In case PLL is not locked – system will be at reset state. In case PLL is locked – system will be at reset state only if the FPGA reset is active.
2. **Sync Reset Generator** – Generate synchronized reset to a given clock.

#### Reset Block - Wave

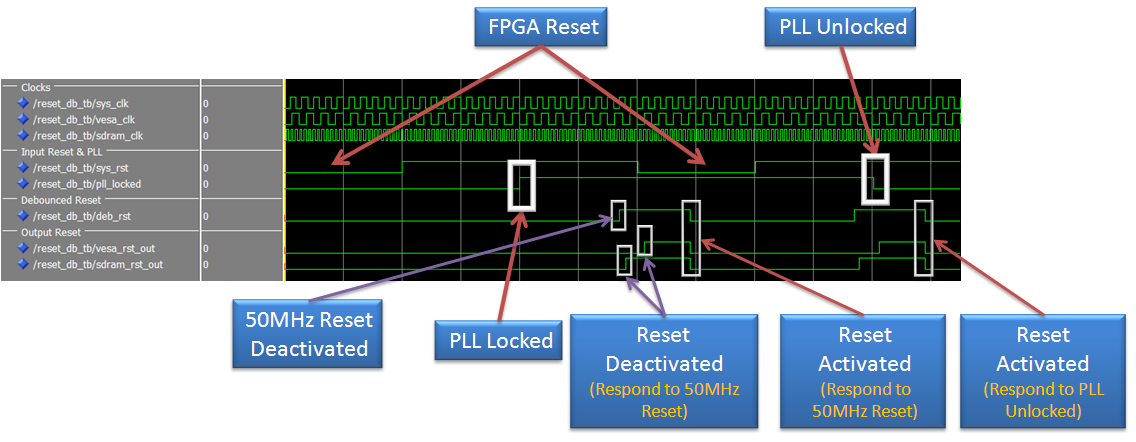


Figure 16 – Reset Block Wave

In the above wave, it is observable that when PLL is not locked – VESA and SDRAM sync resets are activated.

The reset activation occurs at the same time to all clock domains, after filtration.

#### Reset Filter

The Reset Filter filters the input reset and PLL-locked signal, and generate synchronized reset from the AND condition of both signals.

#### Reset Filter Working Method

Figure 17 – Reset Filter Working Method

'A1', 'A2', 'B1' and 'B2' FF purpose is to prevent **metastability** from entering the system, where the first FF ('A1', 'B1') are probably in metastabiliy state. The chance that the metastability will pass beyond the second FF ('A2', 'B2') is almost zero.

The 4 FPGA-Reset's FF and the 4 PLL-Locked-Reset's FF are FF without CLR port, where the last 4 FF, of the Sync-Reset, have an asynchronous CLR (Active Low Reset) port.

**FPGA Asynchronous reset** is sampled by FF, which marked 'A1'. Together with 'A2', both FF prevents metastability. The rest 2 flip-flops purpose is to filter the reset. Only if all last 3 FF are '1', then the output of the CL will be '1', and only if all last 3 FF are '0', then the output of the CL will be '0'. The reset **activation** occurs at the same time to **all clock domains**, after filtration, since all FF are connected to the FPGA clock. Reset deactivation will occur at different time to each clock domain, but very close to each other.

**PLL Locked** is sampled by FF, which marked 'B1'. Together with 'B2', both FF prevents metastability. The rest 2 flip-flops purpose is to filter the PLL signal. Only if all last 3 FF are '1', then the output of the CL will be '1', and only if all last 3 FF are '0', then the output of the CL will be '0'.

AND (An inverter might be placed before the AND, in the reset line, in case it is active low) gate causes all last 4 FF to be to be at reset state when reset is activated / PLL is not locked filtered signal is detected. When reset is deactivated, '1' (or '0' in case of active high reset) will enter to the first FF of the last 4 FF. Since after the PLL Locked and FPGA Reset FF there is a AND gate and CL, the filtered reset is again not synchronized to the clock. The last FF's job is to synchronize the reset deactivation to the clock's rising edge, but to respond immediately to filtered reset activation / PLL not locked filtered signal.

#### Sync Reset Generator

The sync reset generator generates synchronized-to-the-clock reset.

#### Sync Reset Generator Working Method

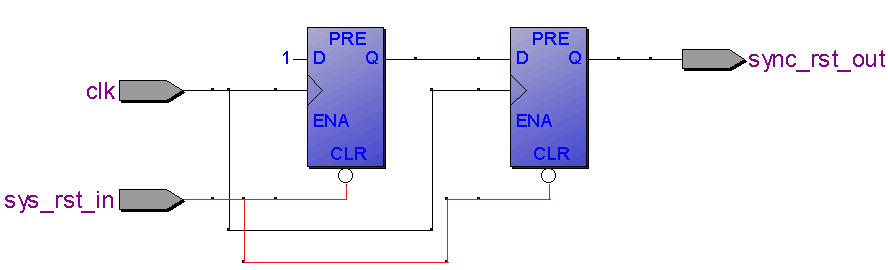


Figure 18 – Sync Reset Generator Working Method

When reset is being activated, the output reset will be activated immediately, independently on the clock. When the reset is being deactivated, '1' (or '0' in case of active high reset) is being entered to the first FF and from there to the second FF. In this method – the reset deactivation is synchronized to the given clock.

**Important**: The input reset is sampled from the filtered FPGA reset, which is synchronized to the FPGA clock and not to the given clock. In such implementation, the reset activation will occur to all clock domains at the same time, and the reset deactivation will occur as short time as possible to each other clock.

### Clock and Reset Documentation

<https://runlen.googlecode.com/svn/branches/Doc/Clock_and_Reset/Clock_and_Reset.docx>

## Top Architecture Details

* **WBM –** Wishbone Master
* **WBS –** Wishbone Slave

### Memory Block

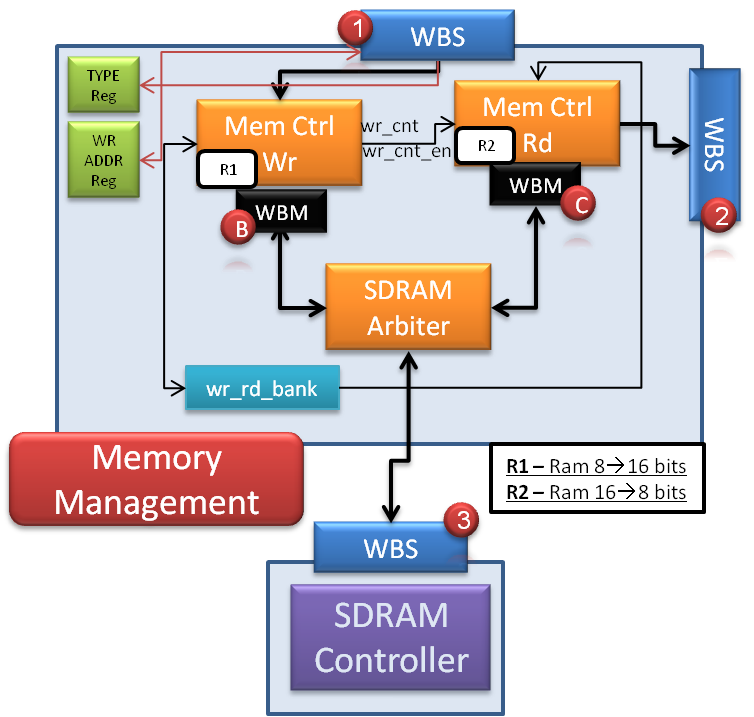


Figure 19– Memory Block

#### Registers in Memory Block

The following registers will be placed in the memory block:

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

* **Type Register** – Message type [W.O]
* **Address Register** – Start SDRAM address to write to / read from [R/W]

#### Write Target

WBS (1) receives a TGC\_I signal, which is the data's target:

* **TGC\_I = '1'**: Write to block's registers
* **TGC\_I = '0'**: Write to block's components

#### RAMs in the Memory Block

*Mem Ctrl Wr* has an 8 bits input (from RX Path) and 16 bits output (to SDRAM) RAM.

*Mem Ctrl Rd* has a 16 bits input (from SDRAM) and 8 bits output (to TX path) RAM.

#### SDRAM Bank Select

The block named *wr\_rd\_bank­* hasan internal register, which its values are '0' or '1'.

* **When '0'**: *Mem Ctrl Wr* will receive the value of '00' (Write to Bank 0 and 1), and *Mem Ctrl Rd* will receive the value of '10' (Read from Bank 2 and 3).
* **When '1'**: *Mem Ctrl Wr* will receive the value of '10' (Write to Bank 2 and 3), and *Mem Ctrl Rd* will receive the value of '00' (Read from Bank 0 and 1).

When the *Type Register*'s value in the memory block indicates that the total sent bytes to the SDRAM is being received, *Mem Ctrl Wr* compares the received amount of received bytes with the number of bytes that it wrote to the SDRAM (Total amount =*wr\_cnt)*. In case *wr\_cnt* is equal to that number, *Mem Ctrl Wr* will signal the *wr\_rd\_bank*, and the above values will switch ('00'↔'01'), so the next image read data from SDRAM will be the last correct received image.

#### Write and Read Counters

*Mem Ctrl Wr* shall have an internal register, named *wr\_cnt.*

*Mem Ctrl* Rd shall have an internal register, named *rd\_cnt.* Its initial value will be 0, so as long as the first correct frame will not be received, no data will be read from the SDRAM to the Image Block.

Details about these two register is available in the *Write to SDRAM* and *Read from SDRAM* chapters.

#### Write to SDRAM

***Mem Ctrl Wr***: WBM (A) initiate a memory write, with burst length, which is the amount of the current available words in the *Mem Ctrl Wr*'s RAM (which is Message Pack Decoder's LENGTH/2). Burst length will be transmitted to the SDRAM [WBM (B) to WBS (3)] using TGA\_O [8..0], which can represent the numbers 0🡪255 (burst length of 1🡪256 words). This burst will be executed more than one time, when the write burst length is greater than 255 (=256 bytes), which is the default burst value.

*Mem Ctrl Wr* shall contain the following internal registers:

1. ***cur\_wr\_addr[21..0]***, which holds the next address, which data should be written in the SDRAM, and it will write data from that address. This register will increment its value.
2. ***wr\_cnt*** (integer range 0🡪640X480 - 1), which holds the number of written words (16 bits) to the SDRAM. Each time a new image transaction will be written to the SDRAM, *rd\_cnt* will be set to *wr\_cnt,* and *wr\_cnt* counter will be set to 0.

Burst operation algorithm:

1. *avail :=* (256 – cur\_wr\_addr[7..0])
2. *blen :=* Required Burst Size, according to current available words in RAM1
3. *Loop while blen > 0*
   1. Execute Write Burst to SDRAM, where burst length is *(blen* ***mod*** *avail*)
   2. *wr\_cnt := wr\_cnt + (blen* ***mod*** *avail*)
   3. *cur\_wr\_addr := cur\_wr\_addr + (blen* ***mod*** *avail*)
   4. *avail :=* 256
   5. *blen :=* (blen **div***avail)*
4. In case the system is working in debug mode, *rd\_cnt* value will be set to *wr\_cnt*.

#### Read from SDRAM

***Mem Ctrl Rd***: WBM (D) or WBM (E) initiate a memory read, which will cause WBS (2) to initiate a memory read from SDRAM, through WBM(C) to WBS(3), where TGA\_O [8..0] represents the burst length. This burst will be executed more than one time, when the Read Burst Length Register (See TX Block) is greater than 255 (=256 bytes).

See *Write to SDRAM* for more information.

*Mem Ctrl* Rd shall contain the following internal registers:

1. ***cur\_rd\_addr[21..0]***, which holds the next address, which data should be read from the SDRAM, and it will read data from that address. This register will increment its value at each burst, and will be zeroed when *cur\_rd\_addr*[19..0] = *rd\_cnt*\_i*,* which is the value of *rd\_cnt* at the beginning of the read transaction. *cur\_rd\_addr[21..20]* value will be set to the value, transmitted from *wr\_rd\_bank* when *cur\_rd\_addr*[19..0] = *rd\_cnt*\_i.
2. ***rd\_cnt*** (integer range 0🡪640X480 - 1), which holds the number of written words (16 bits) to the SDRAM in the last completed transaction. When *cur\_rd\_addr*[19..0] = *rd\_cnt*\_i*,* rd\_cnt\_ivalue will be set to *rd\_cnt*.

Burst operation algorithm:

1. *avail :=* (256 – cur\_wr\_addr[7..0])
2. *blen :=* Required Burst Size: Burst Length Register's value
3. *Loop while blen > 0*
   1. Execute Read Burst from SDRAM, where burst length is *(blen* ***mod*** *avail*)
   2. *rd\_cnt := rd\_cnt + (blen* ***mod*** *avail*)
   3. *cur\_rd\_addr := cur\_rd\_addr + (blen* ***mod*** *avail*)
   4. *avail :=* 256
   5. *blen :=* (blen **div***avail)*
4. In case the system is working in debug mode, *rd\_cnt\_i* value will be set to *rd\_cnt* each clock.

#### SDRAM Arbiter

Internal MUX will switch between WBM (B) and WBM (C). Two CYC\_I inputs will be received from the *Mem Ctrl Rd* and *Mem Ctrl Wr*.

* When *Mem Ctrl Rd* demands control on the SDRAM (using CYC\_O), in case SDRAM is not busy in a write transaction, *rd\_gnt* will be asserted, until CYC\_I will be negated.
* When *Mem Ctrl* Wr demands control on the SDRAM (using CYC\_O), in case SDRAM is not busy in a read transaction, *wr\_gnt* will be asserted, until CYC\_I will be negated.
* A high priority is given to *Mem Ctrl Rd*. In case both *Mem Ctrl Rd* and *Mem Ctrl Wr* demands control on SDRAM at the same time – *rd\_gnt* will be asserted.

### Image Block

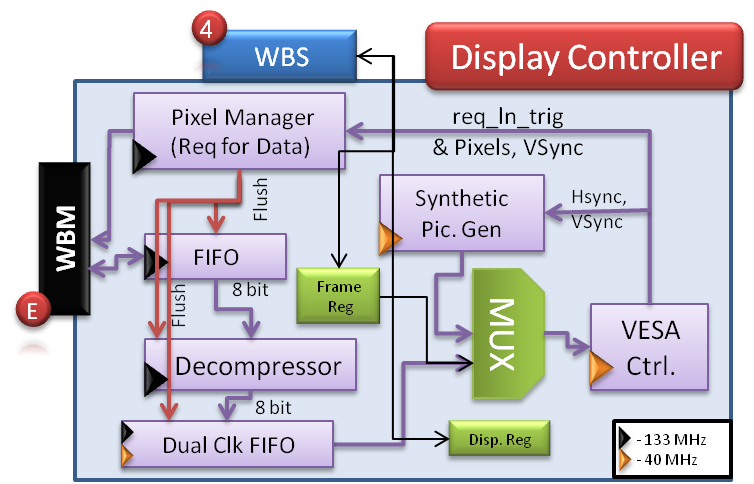


Figure 20– Image Block

The image block transmits the data from the SDRAM / Synthetic Picture Generator, through the DAC on the DE2, to the VGA.

The FIFO's depth is 256X2X3 = 1536 (X8 bits), for 3 SDRAM read burst.

The Dual Clock FIFO's depth is 640X6 = 3,840 (X8 bits), for 6 lines.

#### Registers in the Image Block

The following registers will be placed in the image block:

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

* **Display Register** – Display Synthetic image / Image from SDRAM [R/W]
* **Frame Register** – X and Y coordinates of the frame [R/W]

#### Receive Data

Data transfer is being initialized by the *Pixel Manager*. The burst length will ALWAYS be 256 read cycles. This block has a counter, which holds the current number of received pixels. For example: Suppose that one WORD that has been received is 0xFF03, where 0xFF is the received color and the 0x03 is the repetition + 1. For that transaction, the counter will add '4decimal' to its current value. When the counter's value is *Active Horizontal Pixels X Active Vertical Lines* (i.e.: 640X480 = 307,200 pixels), the WBM is expected to receive [ERR\_I] with [TGD\_I()] from the WBS (3) of the memory block.

There is one exception: In case the last SDRAM transaction, which contains all the end-of-picture pixels, is exactly 512 Bytes. In that case, the transaction will terminate normally ([ACK\_I]), and the same data tag information [TGD\_I()] will be received.

In case of error, which means that the [TGD\_I()]'s value is not as expected, the Dual Clock FIFO will be flashed and the Runlen Decompressor will return to its initial value. New read transaction will start after the *Pixel Manager* will receive *VSync* signal from the SDRAM Controller.

#### Synthetic Picture Generator

According to the Display Register value, an image from the SDRAM will be displayed on the screen, or a synthetic image, generated by the *Synthetic Picture Generator*.

### TX Block

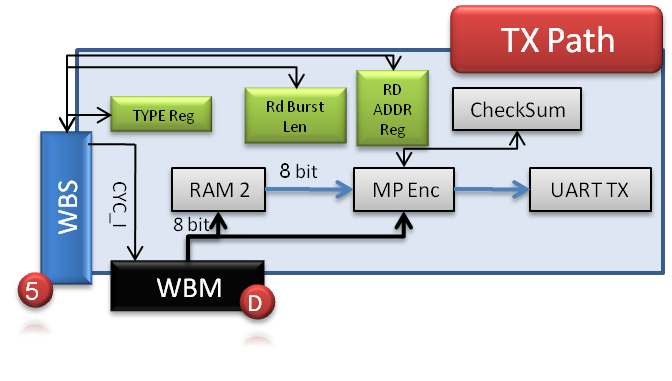


Figure 21– TX Block

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

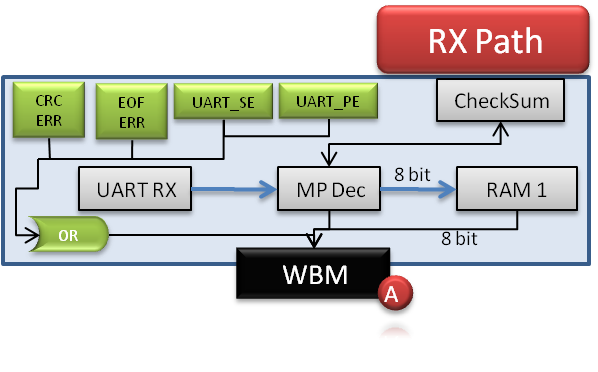


Figure 22– RX Block

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.

### Interconnects

#### INTERCON (Z)

Two Wishbone Masters are connected to this interconnection:

1. WBM (A) from RX Path, which responsible to write data (SDRAM / Registers) to Wishbone Slaves.
2. WBM (D) from TX Path, which responsible to read data (SDRAM / Registers) from Wishbone Slaves.

WBM (A) has a higher priority. WBMs take control on the INTERCON (Z) by raising the CYC\_O signal. When one WBM grants control on the INTERCON, the INTERCON transmit STALL\_O to the inactive WBM.

When writing to TYPE register, more than one block will negate the STALL\_O signal. In this case, higher priority will be given to the Memory Block. The consequence will be that only the memory block's ACK\_O will be latched by the WBM (A).

#### INTERCON (Y)

This interconnection receives the TYPE register, which is transmitted from the WBM (A). In case of debug mode, WBM (D) will be connected to the path. In case of normal mode, WBM (E) will be connected to the path.

#### Registers in the INTERCON (Y)

The following registers will be placed the INTERCON (Y), and will not be shown in the scheme, because of lack of place in the scheme:

(W.O = Write Only)

* **Type Register** – Message type [W.O]

#### INTERCON (X)

According to WBM (D) command (TGC\_I), routes the path toward reading from SDRAM in the Memory Block or from the registers, through INTERCON (Z):

* **TGC\_I = '1'**: Route path to Memory Block
* **TGC\_I = '0'**: Route path to INTERCON (Z)

### Wishbone Cycles

#### General Rules

* All Wishbone Slaves MUST assert STALL\_O, unless they are addressed to.
* TYPE Register is the only register, which exist in more than one block. When writing to them, the main WMB (A) should wait for the ACK\_I signal from only one block, which will be chosen by the INTERCON (Z).

This project uses Read and Write burst, initiated by the Wishbone Master.

* STALL\_I will be use especially for SDRAM (until data is valid: Time of RAS, CAS…) and in the arbiters in the design.
* Wishbone Pipeline Classic Cycle only will be used here.

#### Wishbone BURST (Block) Read Transaction

* In case STALL\_I is asserted, WBM will present the same signals' values as the previous clock edge.
* In case of ERR\_I, skip to step 'Clock edge N+1'. TGD\_I tag will contain the error reason.

**Clock Edge 0**

* MASTER negates WE\_O, asserts CYC\_O, STB\_O and places valid value in ADR\_O, TGA\_O, TGC\_O and SEL\_O

**Clock Edge 1, 2, 3…. N-1, where N is burst size**

* MASTER places new valid values in ADR\_O and TGA\_O
* SLAVE presents valid data on DAT\_I, and asserts ACK\_I

**Clock Edge N**

* MASTER places new valid values in ADR\_O and TGA\_O
* SLAVE presents valid data on DAT\_I, and asserts ACK\_I (Which is already asserted)

**Clock Edge N** **+1**

* MASTER negates STB\_O
* SLAVE presents valid data on DAT\_I, and asserts ACK\_I (Which is already asserted)

**Clock Edge N** **+2**

* MASTER negates CYC\_O
* Slave negates ACK\_I

#### Burst Write Transaction – Start of Cycle

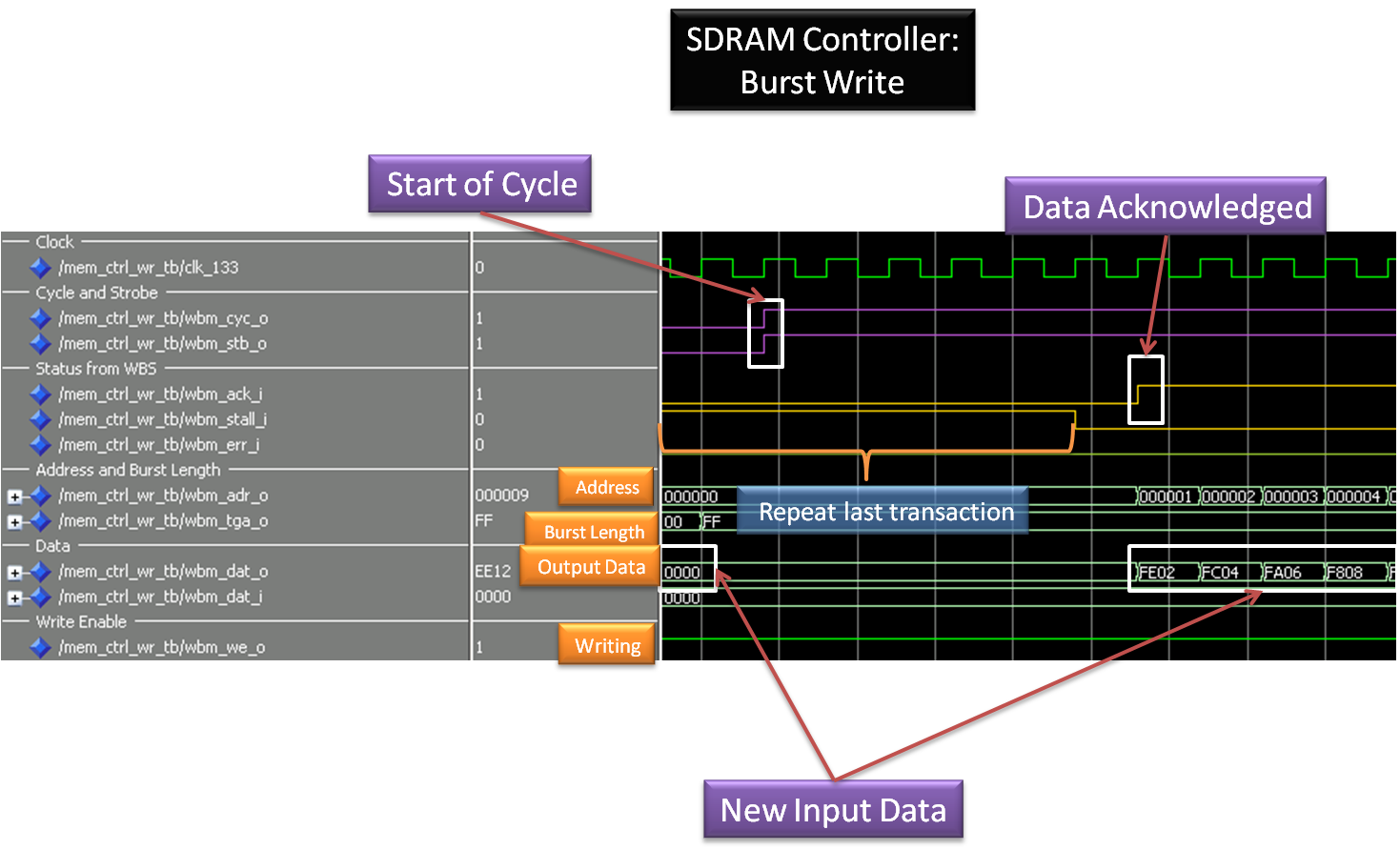


Figure 23 - Start Burst Write Cycle

#### Burst Write Transaction – End of Cycle

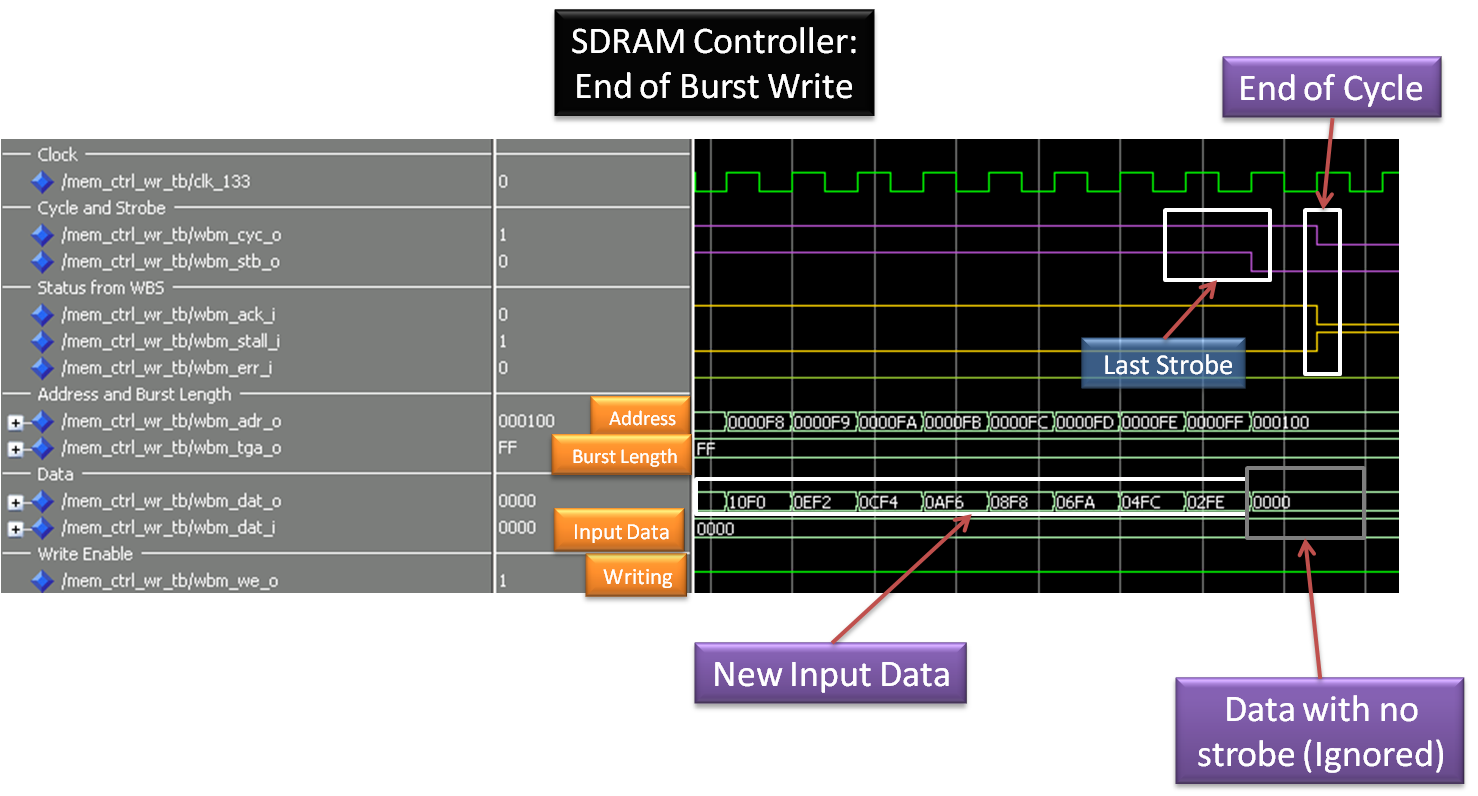
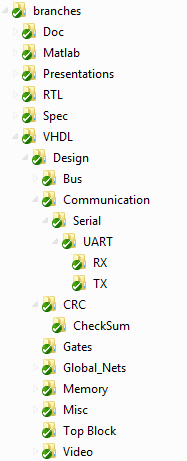


Figure 24 – End Burst Write Cycle

## Project's Directory Structure



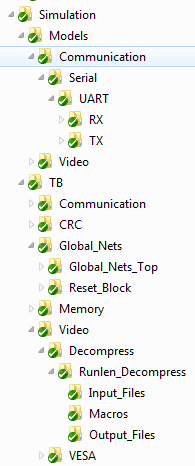


Figure 25– Directory Structure

# Testing

## Simulation

With each implementation completed a test bench is built to test the integrity of each IP. With these simulation arrays all the generic parameters are tested. Various **Generators** and **Macro (do) files** are created to ease up and automate the process and to compare the actual data to the expected data. Test plans and lists are available in each relevant document. Each test generates appropriate Wave images for observation proposes and some also generate output text files automatically that give indication about the properness of the relevant test (Pass/Fail).

## Lab Examination

**Matlab** was used as the algorithm compressor (Run-length) and a **GUI** was created, for algorithm validation, TX and RX Path validation.

Serial data ports (RS-232) opened with Matlab and used to transmit data to check integrity of the communication set (UART TX, RX formation).

### Matlab's GUI

The GUI is a tool for Debug purposes:

* Can load images, compress and de-compress them and display the original and the de-compressed image (as shown in Figure 27).
* User can choose the data packet values.
* Packets can be transmitted through UART.
* Read or write to registers.
* Read or write to SDRAM for debugging.
* More functions will be added to the GUI as the project progress.

Figure 26 – Matlab's GUI

### Algorithm Example from Matlab

In the example below, the algorithm searches for color repetitions. Only two parameters are transmitted: color value (0h🡪FFh) and number of its repetition (Figure 28). This simple algorithm made especially for applications, where the data has small and specific number of changing areas.

The following figure demonstrates the basic idea of this algorithm:

|  |  |  |  |
| --- | --- | --- | --- |
| De-Compressed Data |  | Compressed Data | |
| Color |  | Color | Repetition |
| FF |  | FF | 4 |
| FF |  | A1 | 2 |
| FF |  | C2 | 1 |
| FF |  | B6 | 3 |
| A1 |  | FF | 5 |
| A1 |  |  |  |
| C2 |  |  |  |
| B6 |  |  |  |
| B6 |  |  |  |
| B6 |  |  |  |
| FF |  |  |  |
| FF |  |  |  |
| FF |  |  |  |
| FF |  |  |  |
| FF |  |  |  |

Figure 27 – Algorithm Example

# Summary

## Project Usage

Examples of systems, which can use this project:

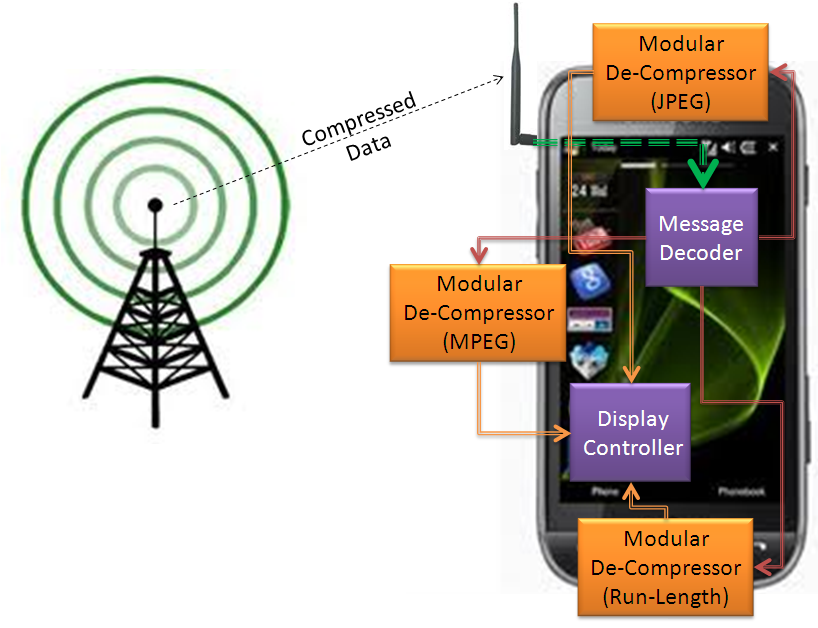
1. **Cellular phones**. Broadcasted data is can be composed of various types of compressed data, in order to reduce the air BW. Compressed data is received and decoded by the cellular's RF interface. Message is unwrapped, and, according to its type, routed to the associated *de-compressor*. In such an application, TX compression can also be implemented, but is not discussed here***.  
   ***

Figure 28 – Possible Usage of the Project in Cellular Phone

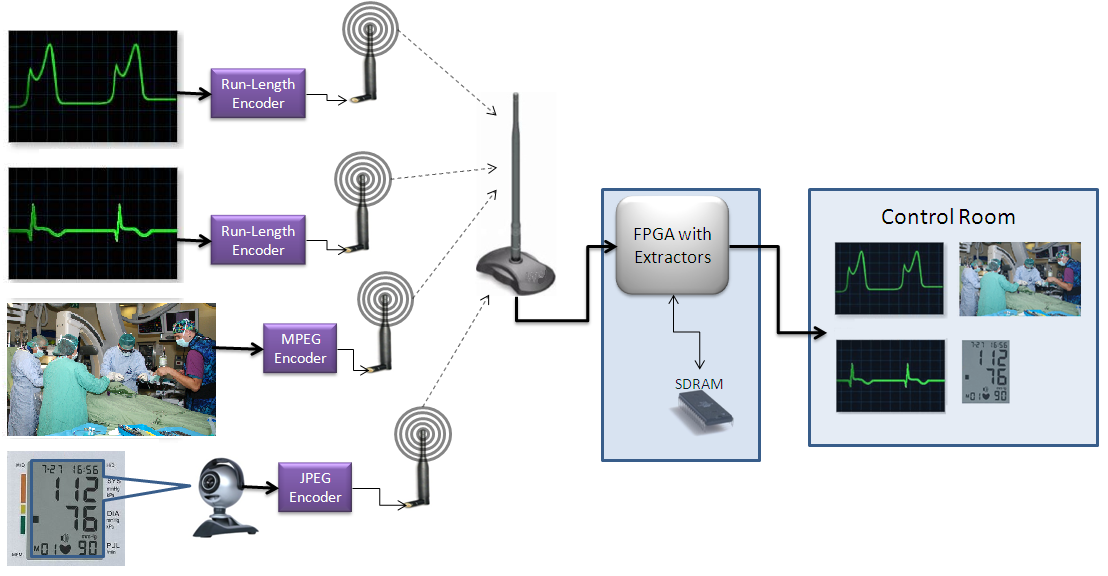
1. ***Wireless monitors***. For example: Hospital, which has thousand of monitors such as heart monitors, blood pressure monitors, etc:  
   Compressed data from many monitors is transmitted over the wireless network, received and routed to one of the many *extractors* in the FPGA, and displayed in the control room either on one screen (using MUX) or multiple screens.  
   **

Figure 29 - Possible Usage of the Project in Hospital Monitors

1. *Fax machines switchboard*. *Most* faxed documents are mostly white space, with occasional interruptions of black, therefore can use this project, with *Run-length*.
2. In ***a Diamond spindle*** machine, Fast Tool Servo (FTS) receives decompressed data, which comes back as a feedback from the engraving motors. This data should be compressed and de-compressed, since the FTS controller has a limited memory capacity and not fast enough to handle the original number of Data arrays needed to form 3D cuts. Run-length algorithm reduces the number of arrays significantly. For example, curving the letter 'A' (see *Figure 30*) reduces the number of arrays from 500,000 to 403. This allows the FTS handle the data more efficiently and its memory does not have to be large.

The analogy to our project is imminent as the 'Spindle encoder feedback' (*Figure 30a*) is similar to our **RX Path**, the run-length decoding data comes from the **Memory Controller** and the 'FTS controller' replaces the **Display Controller**.

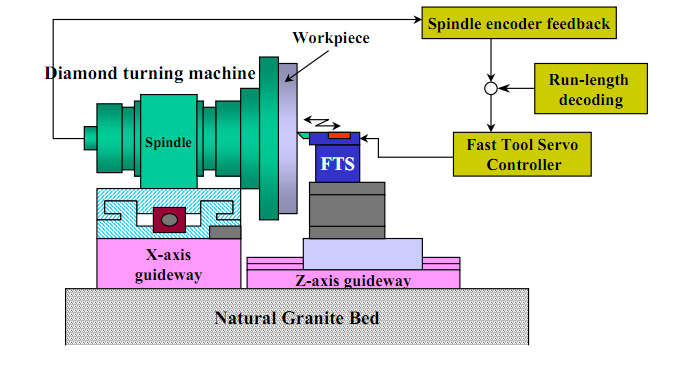
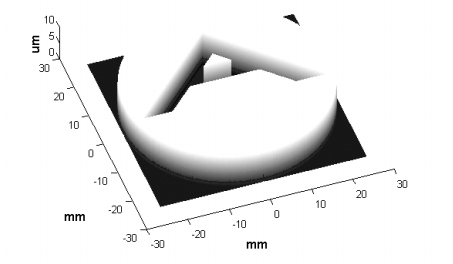
**

Figure 30 - Possible Usage of the Project in Engraving machines Figure 30a – 3D 'A' Shape

## Project Status

The project is a bi-semester project. Only one semester had passed.

At this very moment, image can be transmitted from the host, stored in SDRAM, read from SDRAM, de-compressed and displayed on the VGA screen. *TX Path* is partially implemented. Therefore, read status and SDRAM data by the host for *debug* purposes in not functional yet.

Some of the coded **IP**s in this project are **already in use in other projects** (*UART RX* and *TX*, *Clock and Reset block*), while some of them are planned to be used (*SDRAM Controller, VESA Controller, Message Packs, Checksum*).

## Problems during the project

We have encountered some obstacles during the project. The major obstacles are detailed here:

1. **Problem:** UART RX – When comparing the sent data from the host to the received data in the FPGA, we have found out that some of the data is the same, while some is not. The phenomenon stopped when we defined 2 stop bits instead of 1 in the host, but this is not a solution for the problem. The reason for the problem was that the data was sampled directly from the pin, which caused the first FF to be in metastability.  
   **Solution:** Adding a filter (two FF) before the entrance to the UART RX core fixed the problem. The first two FF prevent from the metastability from entering the system.
2. **Problem:** SDRAM Controller. The SDRAM Controller shall work at the SDRAM frequency, which is 133MHz. After synthesis, we have found out that its estimated frequency is only 117MHz.   
   **Solution:** 
   1. Adding pipeline improved the estimated frequency by 70MHz.
   2. Changing statements, as "if (x <=y-1)" to "if (x < y)"
   3. Limiting *natural* types to specific range improved the estimated frequency by 10MHz.
3. **Problem:** Compressing a 640X480 image in Matlab took about 70 seconds.  
   **Solution**: Limiting the *find* command to 256 or 128 values improved this time to less than 1 second (about 0.3 seconds on Pentium 4, CPU of 2GHz, and 1GB of RAM).
4. **Problem:** Message Pack Decoder: When a SOF is defined as "BABA", and a message received as "BAB**BABA**", the SOF was not recognized.  
   **Solution:** SOF is received into Shift Register, so only the last received data is compared to expected SOF.
5. **Problem:** RAM component is implemented in this project in pure VHDL. When the first implementation was synthesized, we found out the RAM was synthesized to a logic-RAM, with a lot of FF, MUX etc, instead of RAM.  
   **Solution:** Implementing the data\_out process without reset dependency caused the synthesizer to create real RAM, with no extra logic cells.

## Project Conclusions

While designing the project, two goals lead our way:

1. A system, which can be operated in various ways.
2. A scalable system, which can be duplicated, and be used differently in each of its implementation.

These goals were achieved by designing the project as **modular** and **generic**, and by using **known standards**. Therefore, it can be operated in various ways, and **re-used** in other projects easily.

In addition, goals had been achieved by:

1. Organized working methods that saved time.
2. SVN – Difference between previous versions
3. Code Review – Improves our knowledge
4. Documentations – Causes to understand better what we are doing
5. Comprehensive SIMULATIONS – It is much easier to find bugs in simulation than in the lab 🡺 Saves Time!

## Coding Conclusions

1. 2 FF for every a synchronized input FPGA pin is essential, to prevent metastability from entering the system.
2. Using Shift Register makes the design work in greater clock frequency.
3. Pipeline can improve performance.
4. RTL view can teach a lot: RAM can be implemented in many ways, and only RTL can show what the correct way is.
5. Synchronizing the reset negation to the clock’s rising edge is essential, to prevent metastability from entering the system.
6. It is recommended that output ports, from the FPGA to the board, will not be used as internal signals in the FPGA, since the components DFF will be placed somewhere in the FPGA instead on the output FPGA pins PADs, therefore the TCO of the output signals of this components to the board will not be exactly synchronized. This comment is relevant especially for fast clocks, where the TCO is critical.