Maxime Grégoire

260479270

Patrick White

260353491

Final report

ECSE 431

Introduction to VLSI CAD

McGill University

December 10th 2013

Contents

[Description of the design 2](#_Toc374501493)

[PLL design 2](#_Toc374501494)

[DMA engine design 2](#_Toc374501495)

[Using the Grab Interface 4](#_Toc374501496)

[Modification to VGA module 4](#_Toc374501497)

[Using the Line Buffer 5](#_Toc374501498)

[C program and initialization 5](#_Toc374501499)

[QSYS component 5](#_Toc374501500)

[Processing 6](#_Toc374501501)

[Constraint file 6](#_Toc374501502)

[Compilation reports 6](#_Toc374501503)

[Sources of problems 6](#_Toc374501504)

[APPENDIX 7](#_Toc374501505)

[Figure 1 - DMA Engine Block Diagram 2](#_Toc374501463)

[Figure 2 - DMA engine FSM 3](#_Toc374501464)

[Figure 3 - DMA engine line FSM 3](#_Toc374501465)

[Figure 4 - Address map of the QSYS components 7](#_Toc374501466)

[Figure 5 - The connections of the QSYS components 7](#_Toc374501467)

[Figure 6 - When receiving SOF and SOL, the DMA Engine reads a line and writes into the line buffer 7](#_Toc374501468)

[Figure 7 - When receiving SOF signal, the DMA engine reads a line from memory 7](#_Toc374501469)

[Figure 8 - When receiving SOL, the DMA engine reads a line from memory 8](#_Toc374501470)

[Figure 9 - The DMA engine writes a full frame of 4 lines 8](#_Toc374501471)

[Figure 10 - GFSTART vs DMAFSTART registers 8](#_Toc374501472)

[Figure 11 - Grab Interface goes active after a snapshot 8](#_Toc374501473)

[Figure 12 - Initialization of the register file by the NIOS II 9](#_Toc374501474)

[Figure 13 - End Of Frame Interrupt 9](#_Toc374501475)

[Figure 14 - 2 Full Frames in the Grab Interface 9](#_Toc374501476)

[Figure 15 - Setting a snapshot 9](#_Toc374501477)

[Figure 16 - VGA Controller toggling read address 10](#_Toc374501478)

[Figure 17 - Pixel Ramp in VGA Controller 10](#_Toc374501479)

[Figure 18 - DMA engine and VGA controller using the line buffer 10](#_Toc374501480)

[Figure 19 - Odd field does not seem to work 10](#_Toc374501481)

[Figure 20 - Analysis & Synthesis Summary 10](#_Toc374501482)

[Figure 21 - Clocks present in our design 10](#_Toc374501483)

[Figure 22 - Flow Summary 11](#_Toc374501484)

[Figure 23 - Slow Model Maximum Frequency 11](#_Toc374501485)

[Figure 24 - Slow Model Hold Summary 11](#_Toc374501486)

[Figure 25 - Slow Model Minimum Pulse Width 11](#_Toc374501487)

[Figure 26 - Slow Model Recovery Summary 11](#_Toc374501488)

[Figure 27 - Slow Model Removal Summary 12](#_Toc374501489)

[Figure 28 - Slow Model Setup Summary 12](#_Toc374501490)

[Figure 29 - Unconstrained Paths 12](#_Toc374501491)

# Description of the design

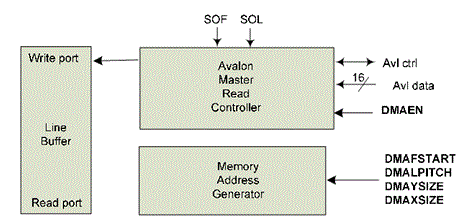
The FPGA is used to display images coming from a standard NTSC signal through the video decoder on a graphic monitor. The image is temporarily stored into memory before being displayed. The grab interface writes the image into memory, the NIOS processes it, and the DMA engine reads it and feeds the VGA controller that displays it. The NIOS also executes the driver code that sequences the acquisition *field by field*. The Qsys system operates at 100MHz.

# PLL design

The altera DE2 board only has 2 clocks, one of 27MHz, which is used for the TV decoder, and one of 50MHz. To generate the 100MHz clock required for the Qsys design, a Phase Locked Loop was implemented based on the 50MHz. The 100MHz clock generated is in phase with the 50MHz clock and has a frequency equivalent to the 50MHz clock multiplied by a factor of 2. Also, the on board SDRAM requires an offset 100MHz clock. This is because there is a physical delay from the Cyclone II FPGA to the actual SDRAM chip. To create this delayed clock, the 50MHz clock was again used and multiplied by a factor of 2. To create the delay required an offset of -3ns was used. Therefore, the clock to the SDRAM goes high 3ns before the other 100MHz clock, which allows it sufficient time to react.

# DMA engine design

The DMA engine we implemented acted as an Avalon Master controller that writes data into a line buffer, as mentioned in the specifications. The engine was designed after the given block diagram.

Figure 1 - DMA Engine Block Diagram

The DMA engine sends burst read requests of 120 bytes to the Avalon Interface (with the address being the SDRAMs address) upon receiving the SOF and SOL signals (see appendix, Figure 6, Figure 7, Figure 8). It can write a full frame into the line buffer (see appendix, Figure 9). Figure 2 demonstrates the FSM used by the DMA engine in fetching an entire frame and writing the frame to the line buffer. Figure 3 demonstrates the FSM used by the DMA engine in fetching a single line from memory and writing that line to the line buffer. Combined, both FSMs make up the entire functionality of the DMA engine. Other functionalities described in the register file list, such as DMA line reversal, were not implemented due to project time constraints.

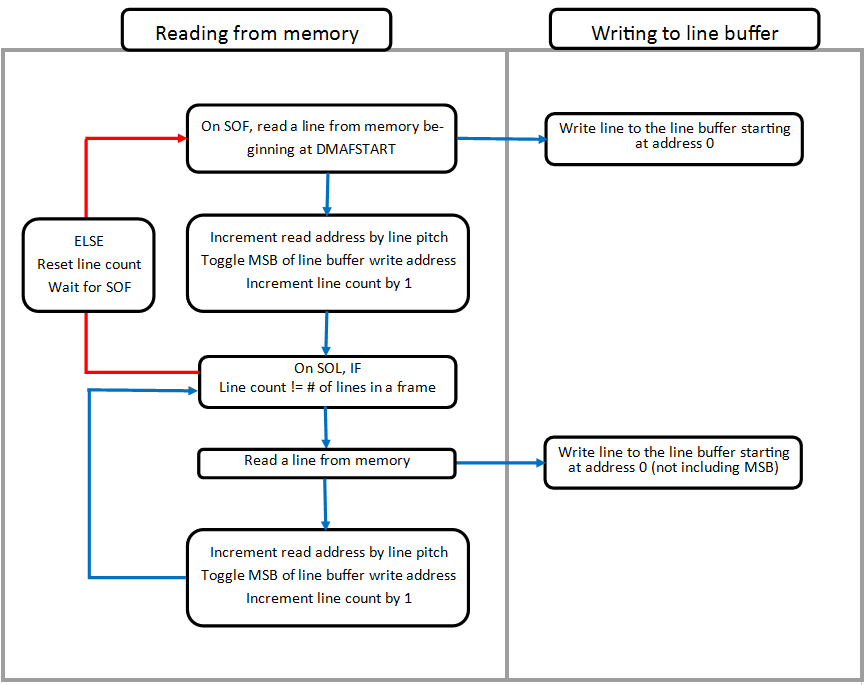
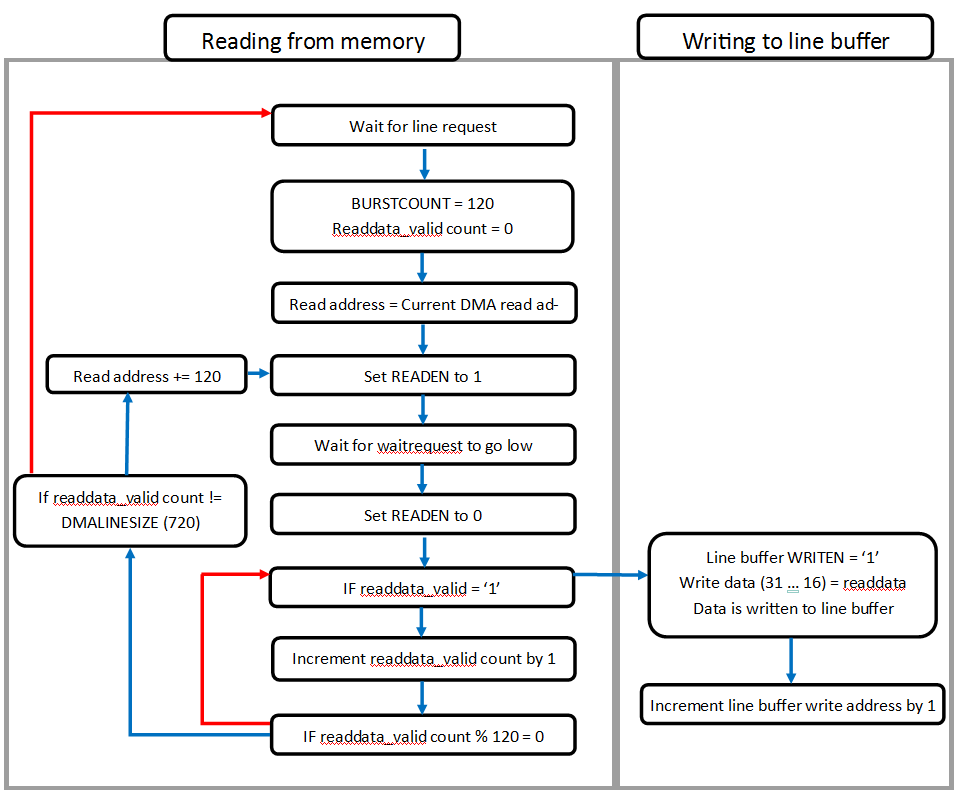
Figure 2 - DMA engine FSM

Figure 3 - DMA engine line FSM

Note that the code for the "dma\_engine.vhd" code is in the "code.zip" archive, attached to this report.

# Using the Grab Interface

The grab interface used in the design was provided for us. It is programmable to capture variable sized lines and frames from the ADV7181b TV decoder and write them to SDRAM memory. However, the grab interface had to be interfaced properly in the Qsys system (see appendix, Figure 5). Also, the grab interface had to be programmed and controlled properly by the NIOS processor. This can be seen in Figure 14 where the grab interface captures 2 full frames from the ADV7181b decoder model. Because the grab interface only captures even or odd fields of a frame, a full frame is grabbed over the span of two active cycles. For example, with 1440 bytes per line, the grab interface line pitch is set to 2880 bytes. The even field lines of the frame will be grabbed first, and written to memory beginning at address 0 (each subsequent line will be written beginning 1 line pitch further in memory than where the last line begins). The odd field lines will be captured next and will be written into memory beginning at address 1440. Because the line pitch used is twice the actual line size, the odd field lines will be written into the memory gaps that were not filled by the even field lines. This creates a continuous block of memory that contains the entire frames information deinterlaced. The next frame will be written into memory beginning at an address equal to the total number of line per frame multiplied by the line pitch. Due to design decisions, by the time the second frame has been written into memory, the DMA engine has already read the first frame from memory. This can be seen by observing the opposing nature of the GFSTART and DMAFSTART registers in Figure 10. Therefore, the next frame (third frame) will overwrite the first frame and begin at address 0 of the memory.

The grab interface will begin its acquisition of a frames field a short period after it receives a snapshot signal from the register file. This can be seen in Figure 11 where the grab interface goes active and begins writing to memory a clock cycle after the snapshot pending signal drops low.

# Modification to VGA module

The VGA controller used in the design was a modified version of one that was provided to us. The original VGA controller was a slave in its implementation. In this implementation, the VGA controller is essentially the master of the DMA engine. The modification required to make this possible is for the VGA controller to generate SOF and SOL single clock cycle pulses to indicate to the DMA engine when to fetch and write lines to the line buffer. The SOF signal is generated on a falling edge of the VSyncN signal and indicates that a new frame has begun. The SOL signal is generated on a falling edge of the HSyncN signal and indicates that a new line has begun within the frame. The SOL signal is also used to toggle the upper most bit of the read address sent to the line buffer as seen in Figure 16.

In simulation, the ADV7181b decoder provided a pixel ramp. The output of the VGA controller for this pixel ramp can be seen in Figure 17. The RGB values correspond to the ramped value shifted left by 2 bits.

Other functionalities described in the register file list, such as VGA zoom, were not implemented due to project time constraints.

# Using the Line Buffer

Because of the design decisions made when creating the VGA controller, the line buffer must be used in a very specific way. The VGA controller only uses the top 16 bits of the 32 bit data it receives from the line buffer. Therefore, any data writing into these 32 bits from the DMA engine must only be written into bits 31 down to 16. This can be seen in Figure 3 when the DMA engine is writing data into the line buffer.

The address of data being written to the line buffer from the DMA engine must also not conflict with the current address of the data being read from the VGA controller. This ensures that the VGA controller will always be reading a line that has been completely written to the line buffer, while at the same time, the DMA engine will be writing the next line. This can be seen in Figure 18 where the DMA engine writes to the upper address of the line buffer while the VGA controller is reading from the lower address.

# C program and initialization

For an easy switch between simulation and hardware modes, two set of values have been defined for the initialization of the register file. To change the mode, we simply change the value of the "sim" variable to either "SOFTWARE" or "HARDWARE".

Here is a list of the actions done by the NIOS II:

1. It writes the fields of the register file (see appendix, Figure 12)
2. It registers the IRQ (interrupt request)
3. It enables the DMA engine and writes its parameters
4. It sets a snapshot (GSSHT register)

The Interrupt Service Routine occurs when there is an "End Of Frame" (see appendix, Figure 13). It does the following things:

1. Resets the SOFISTS and EOFISTS signals (only EOFISTS is actually used)
2. Toggle the GMODE from even to odd field or odd to even field
3. Updates GFSTART and DMAFSTART registers to a new frame location
4. Sets the GSSHT signal to activate the acquisition (see appendix, Figure 15)

The complete code can be found under the name "FPGA.c" in the "code.zip" archive attached to this report.

# QSYS component

The address map of the QSYS components is defined on Figure 4 of the appendix.

The connections of the QSYS components are defined on Figure 5 of the appendix.

# Processing

Since we were only two students to do this project, we could not get any image processing done in time. However, if we had to implement processing, we would stop grabbing frames, read a chunk from memory, modify it, and write it back into memory. We would have to be careful not to read a whole frame from memory all at once since we do not want to duplicate the memory space already provided by the SDRAM.

# Constraint file

The full constraint file used to compile our design can be found under the name "Real constraint file.txt" in the "code.zip" archive attached to this report. Two clocks were created: 27 MHz, 50 MHz, while the 100 MHz clock was generated by a PLL. False paths were then set from the 27 MHz clock to the 100 MHz clock and vice versa.

# Compilation reports

The screenshots of the compilation reports can be found in Figure 20 to Figure 29 of the Appendix. Our design used 20% of the logic elements available as well as 56% of the memory bits (see appendix, Figure 22). According to the “slow model”, we could have used a maximum frequency of 103.32 MHz for the 100 MHz clock and 148.63 MHz for the 27 MHz clock (see appendix, Figure 23). The hold time of the 27 MHz and 100 MHz clock both have a slack of 0.391 (see appendix, Figure 24) while the 27 MHz has a slack of 30.272 in its setup time and the 100 MHz has a slack of 0.321 (see appendix, Figure 28). Note that we did not get any illegal clocks (see appendix, Figure 29).

# Sources of problems

In the demonstration, we were able to display video from a source, though part of the screen was blurry. Afterwards, we noticed that the grab interface is not writing enough, causing the DMA to read invalid data on odd fields only (see appendix, Figure 19). We were unable to correct this issue.

# APPENDIX

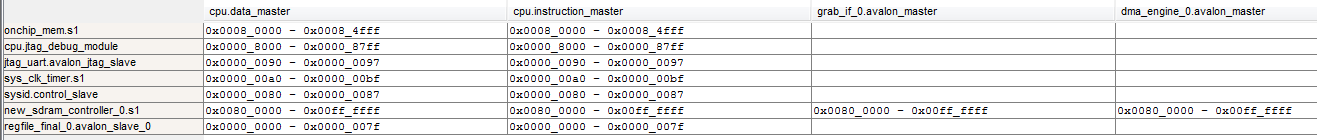
Figure 4 - Address map of the QSYS components

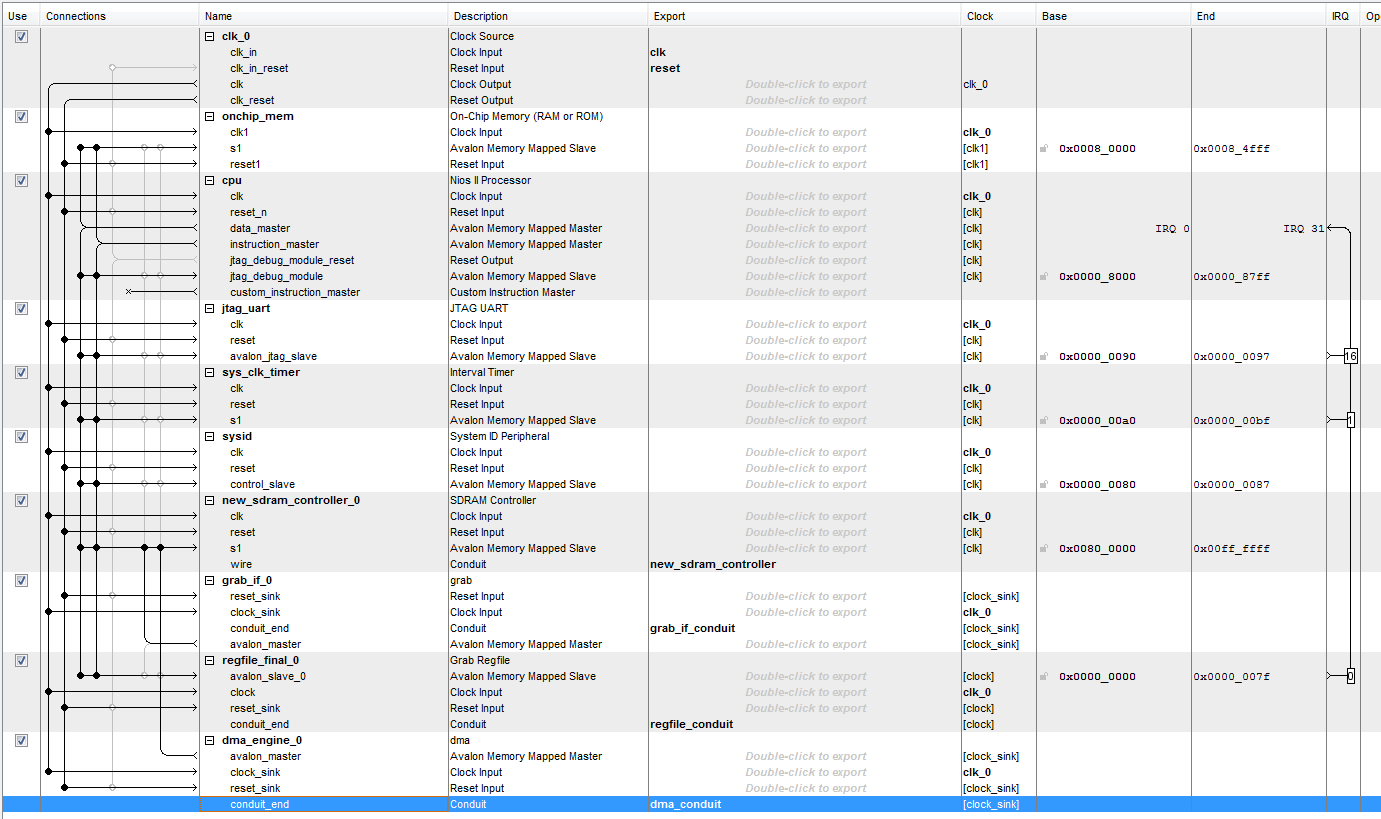
Figure 5 - The connections of the QSYS components

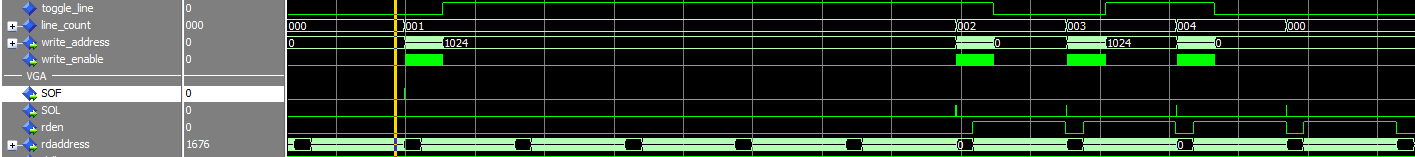
Figure 6 - When receiving SOF and SOL, the DMA Engine reads a line and writes into the line buffer

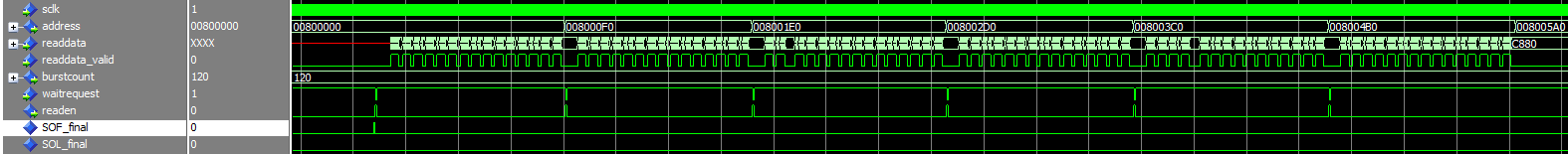
Figure 7 - When receiving SOF signal, the DMA engine reads a line from memory

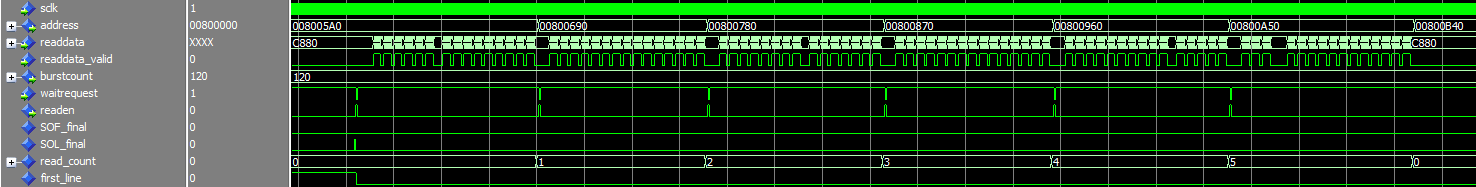
Figure 8 - When receiving SOL, the DMA engine reads a line from memory

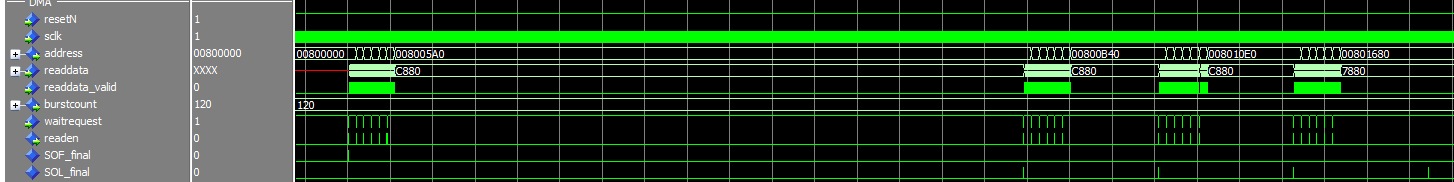
Figure 9 - The DMA engine writes a full frame of 4 lines

Figure 10 - GFSTART vs DMAFSTART registers

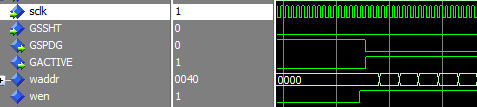
Figure 11 - Grab Interface goes active after a snapshot

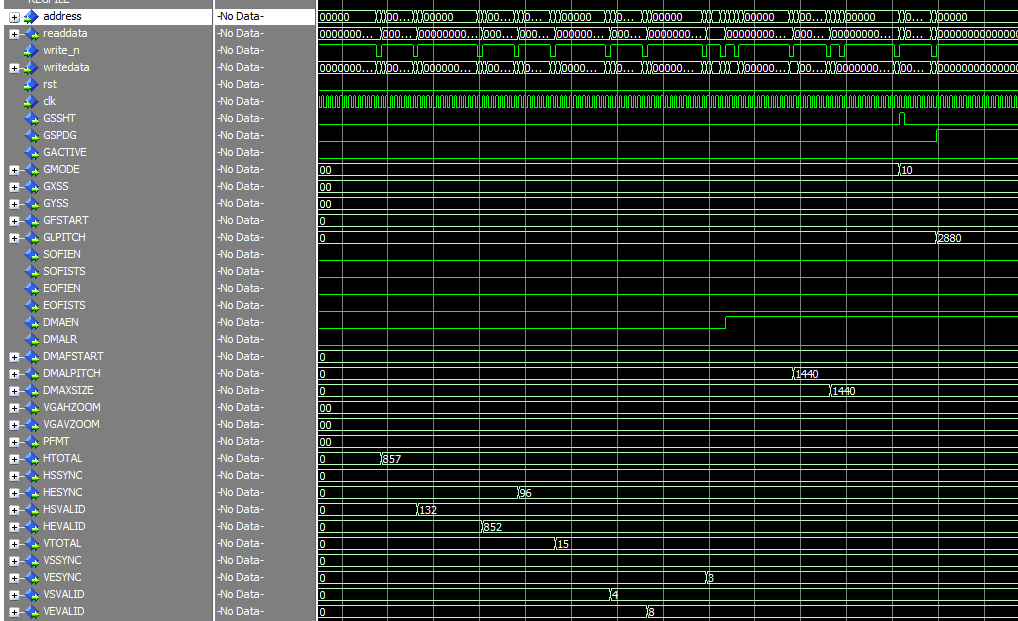
Figure 12 - Initialization of the register file by the NIOS II

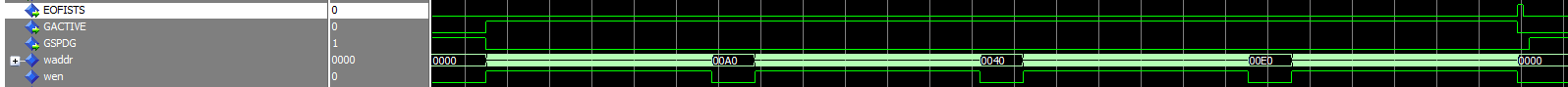
Figure 13 - End Of Frame Interrupt

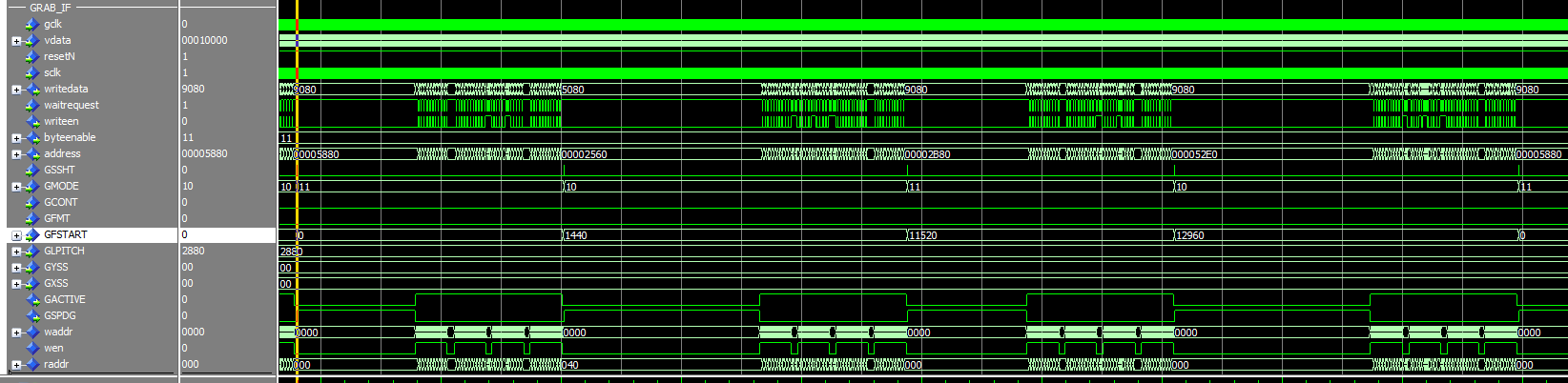
Figure 14 - 2 Full Frames in the Grab Interface

Figure 15 - Setting a snapshotC:\Users\pwhite8\vlsi\Screenshot\Setting a snapshot.png

Figure 16 - VGA Controller toggling read address

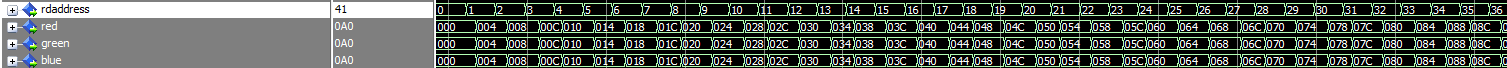
Figure 17 - Pixel Ramp in VGA Controller

Figure 18 - DMA engine and VGA controller using the line buffer

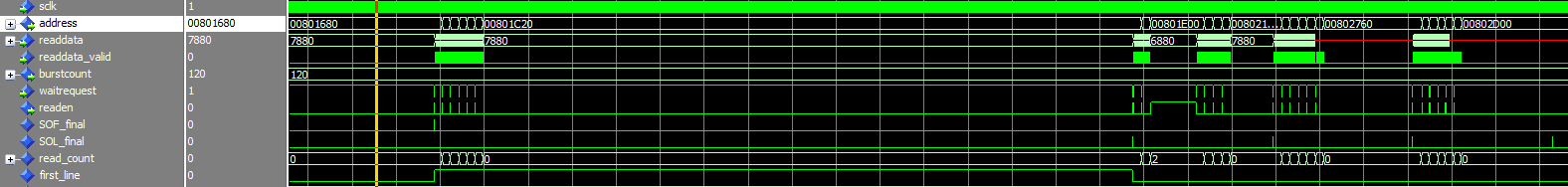
Figure 19 - Odd field does not seem to work

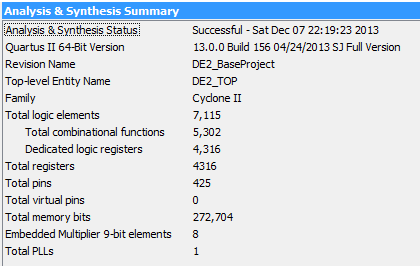
Figure 20 - Analysis & Synthesis Summary

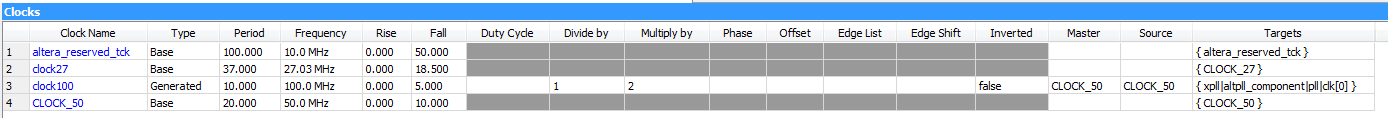
Figure 21 - Clocks present in our design

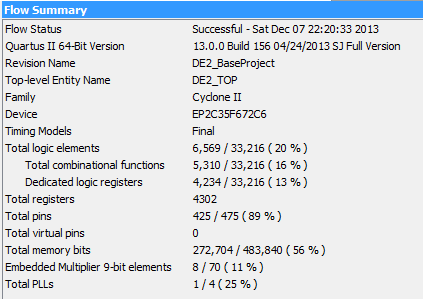
Figure 22 - Flow Summary

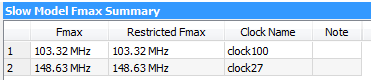
Figure 23 - Slow Model Maximum Frequency

Figure 24 - Slow Model Hold Summary

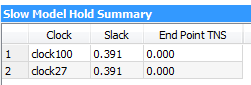


Figure 25 - Slow Model Minimum Pulse Width

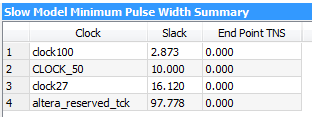


Figure 26 - Slow Model Recovery Summary



Figure 27 - Slow Model Removal Summary

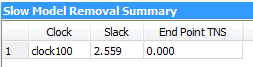


Figure 28 - Slow Model Setup Summary

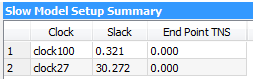


Figure 29 - Unconstrained Paths

