Maxime Grégoire

260479270

Patrick White

260353491

Final report

ECSE 431

Introduction to VLSI CAD

McGill University

December 10th 2013

**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.

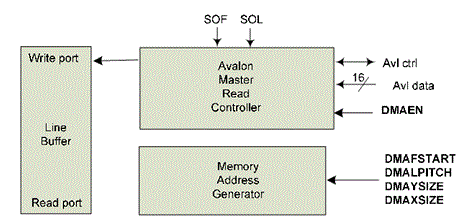


Figure 1 - DMA Engine Block Diagram

The DMA engine sends bursts of 120 bytes to the line buffer upon receiving the SOF and SOL signals (see appendix, Figure 4, Figure 5, Figure 6). It can write a full frame into the line buffer (see appendix, Figure 7). 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.

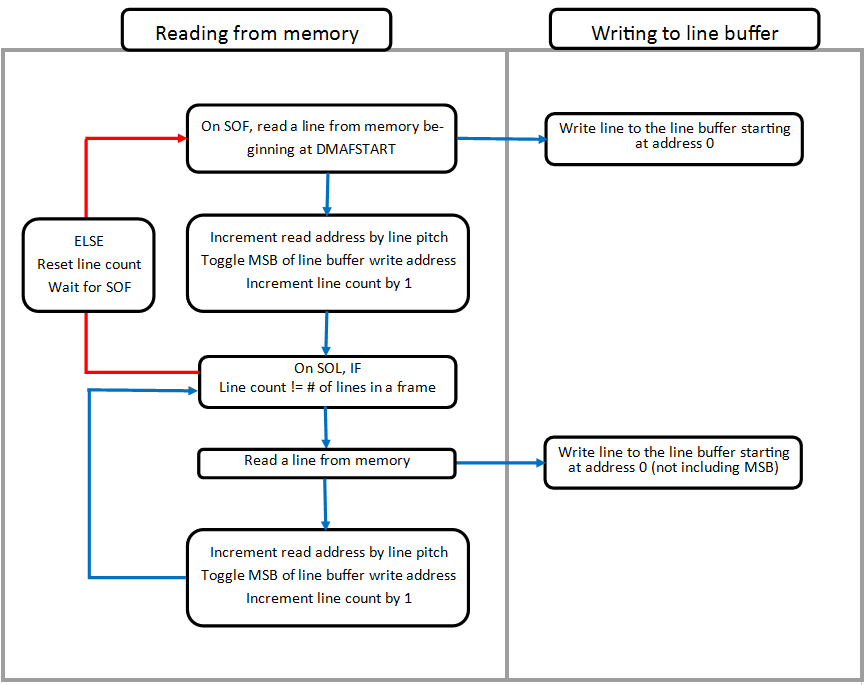


Figure 2 - DMA engine FSM

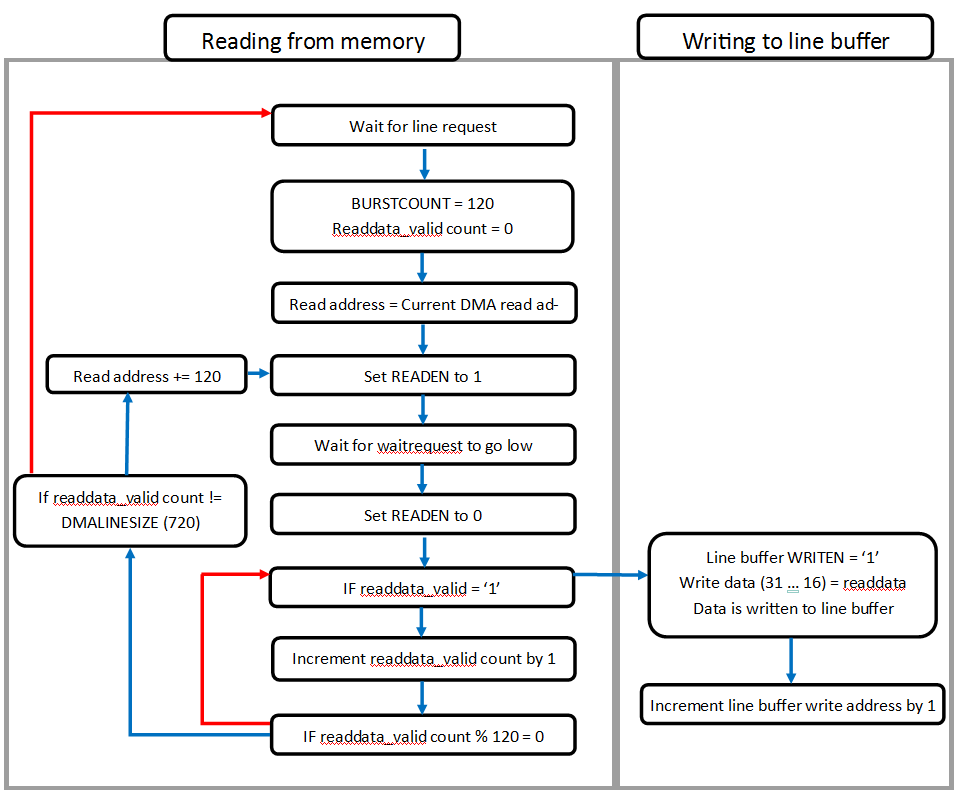


Figure 3: DMA engine line FSM

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

**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 original 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.

In simulation, the ADV7181b decoder provided a pixel ramp. The output of the VGA controller for this pixel ramp can be seen in Figure 12. The RGB values correspond to the ramp shifted by left by 2 bits.

**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 8)
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 9). It does the following things:

1. Resets the SOFISTS and EOFISTS signals
2. Toggle the GMODE from even to odd frame or odd to even frame
3. Updates GFSTART and DMAFSTART registers to a new line location (see appendix,
4. Sets the GSSHT signal (see appendix, Figure 10)

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. 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 grab a whole frame since we don’t want to create a useless block of memory.

**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 and 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 13 to Figure 23 of the Appendix. Our design used 20% of the logic elements available as well as 56% of the memory bits (see appendix, Figure 15). 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 17). The hold time of the 27 MHz and 100 MHz clock both have a slack of 0.391 (see appendix, Figure 18) 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 22). Note that we did not get any illegal clocks (see appendix, Figure 23).

**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 13).

**APPENDIX**

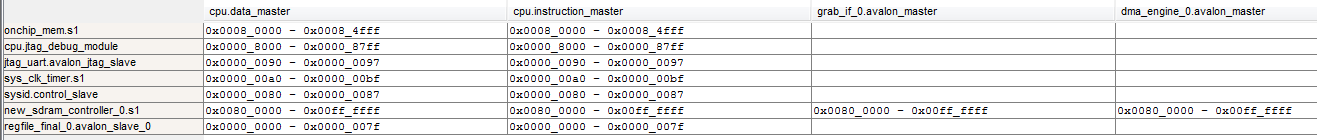


Figure 4 - Address map of the QSYS components

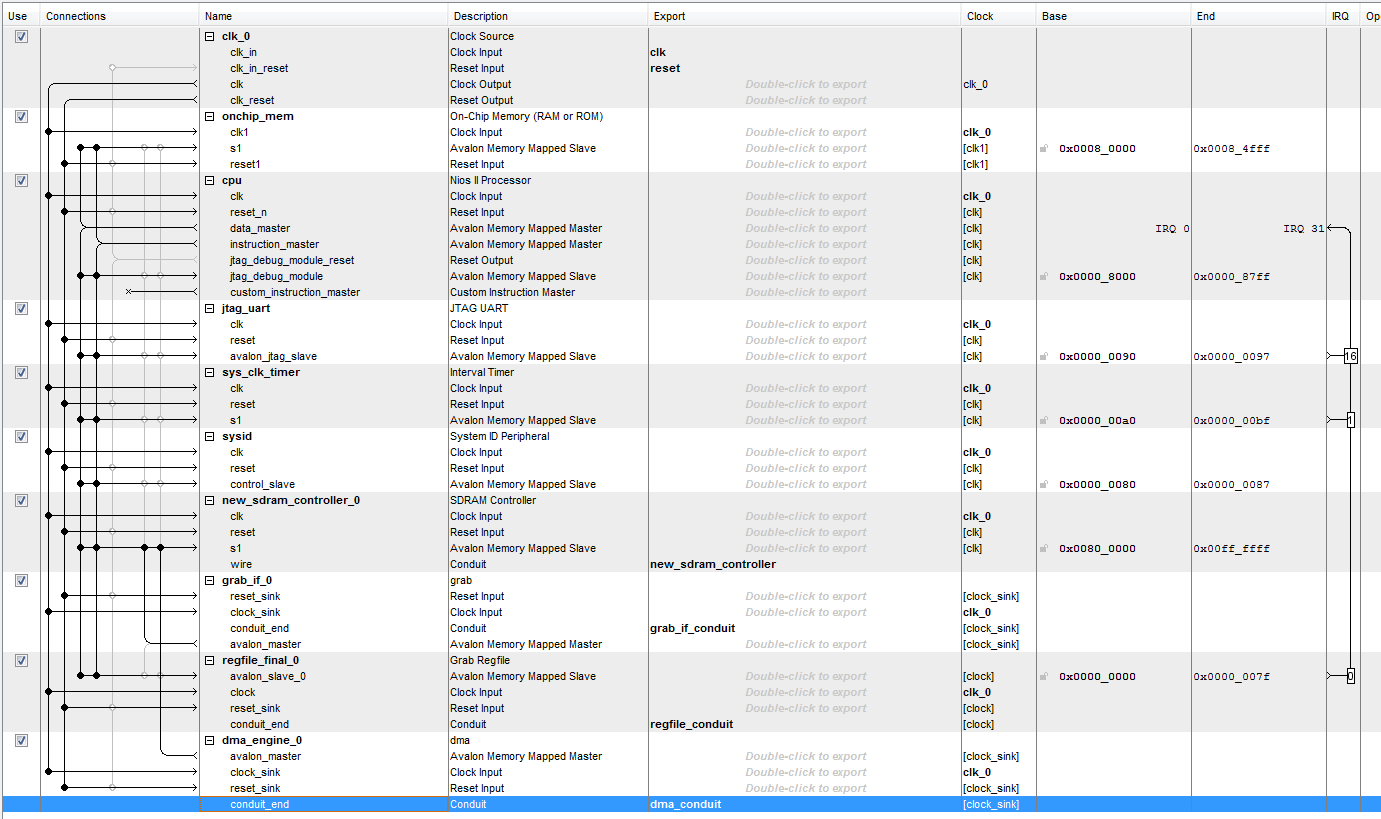
****

Figure 5 - The connections of the QSYS components

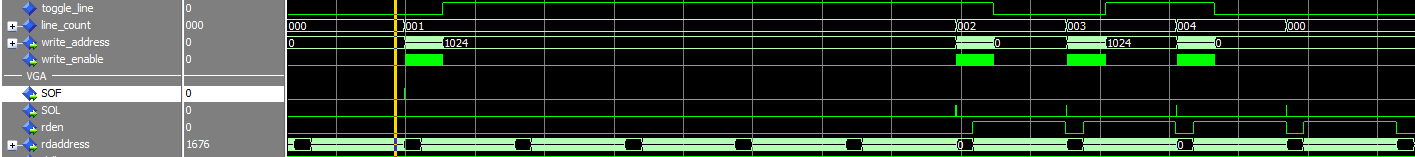


Figure 6 - When receiving SOF and SOL, the DMA Engine writes into the line buffer,

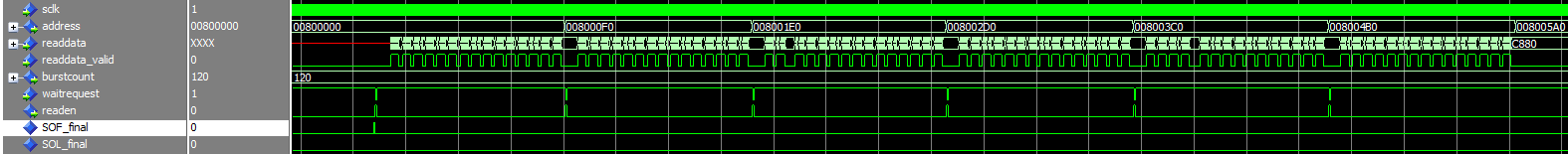


Figure 7 - When receiving SOF signal, the DMA engine writes into the line buffer

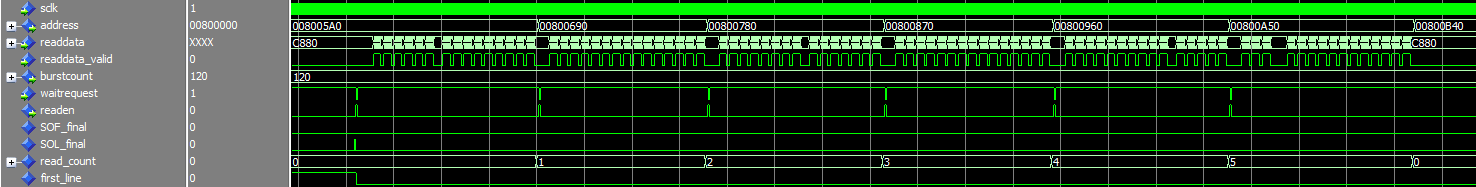


Figure 8 - When receiving SOL, the DMA engine writes into the line buffer

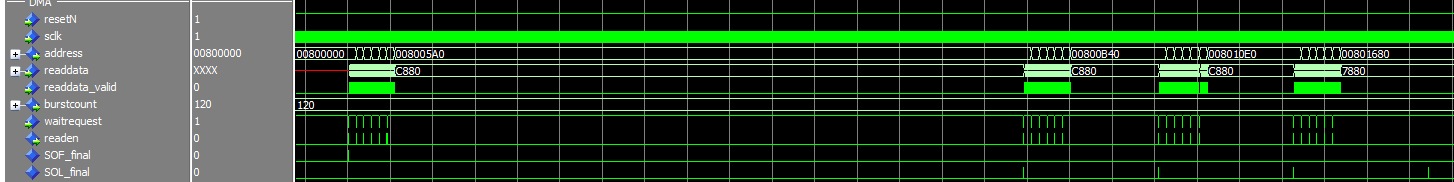


Figure 9 - The DMA engine writes a full frame of 4 lines

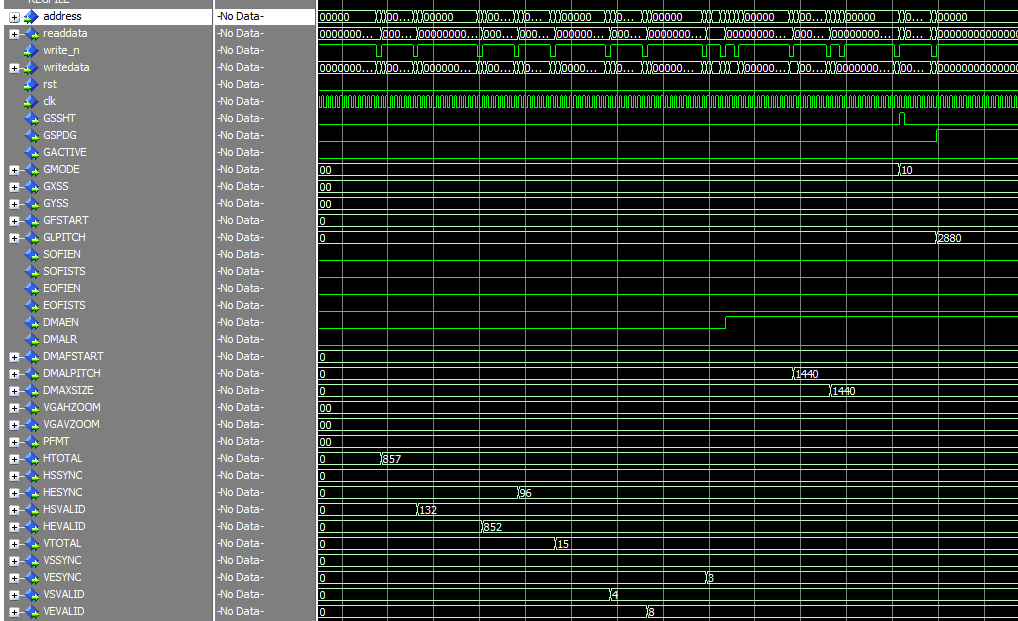


Figure 10 - Initialization of the register file by the NIOS II

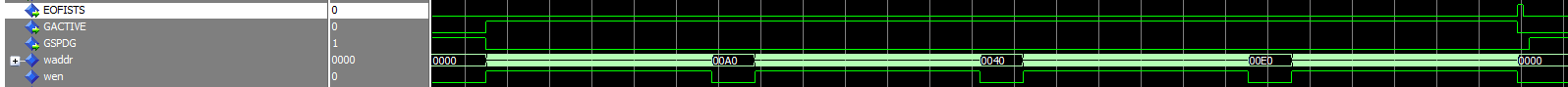


Figure 11 - End Of Frame Interrupt

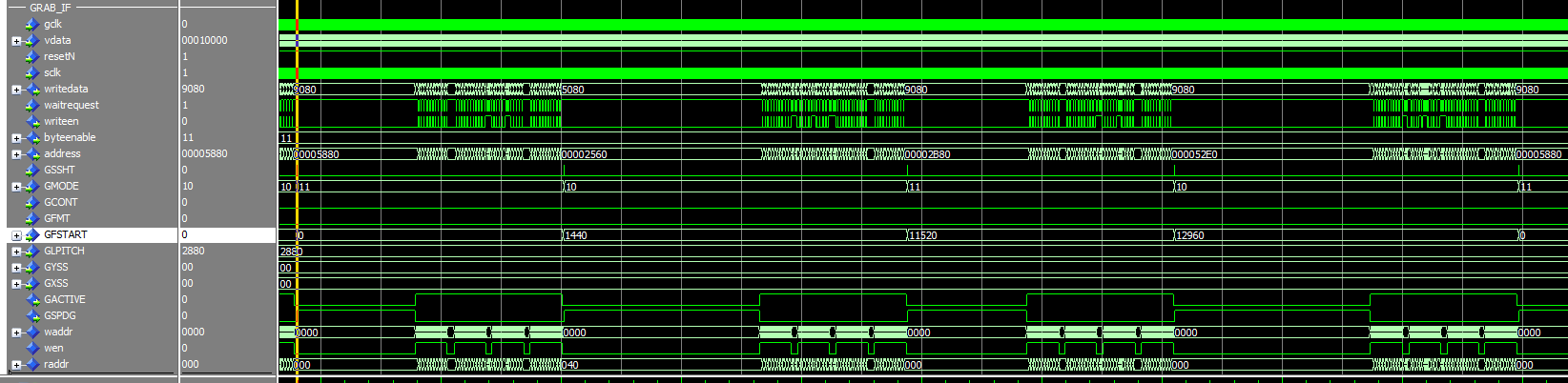


Figure 12: 2 Full Frames in the Grab Interface

C:\Users\pwhite8\vlsi\Screenshot\Setting a snapshot.png

Figure 13 - Setting a snapshot

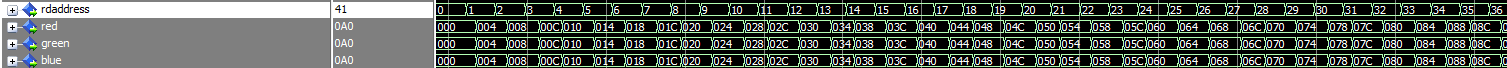


Figure 14: Pixel Ramp in VGA Controller

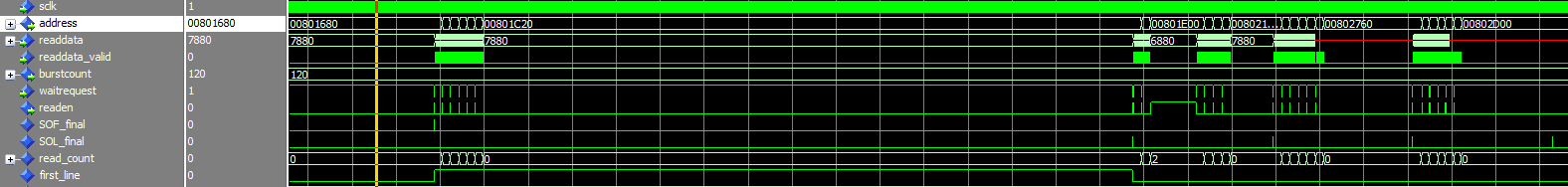


Figure 15 - Odd field does not seem to work

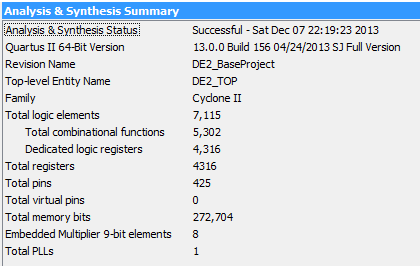


Figure 16 - Analysis & Synthesis Summary

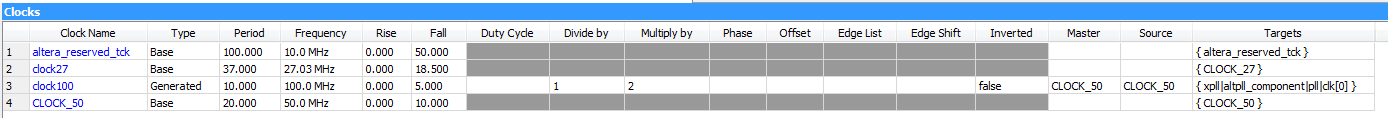


Figure 17 - Clocks present in our design

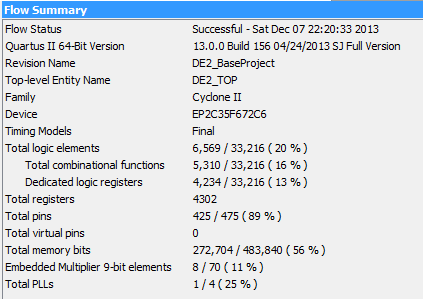


Figure 18 - Flow Summary

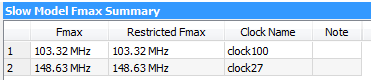


Figure 19 - Slow Model Maximum Frequency

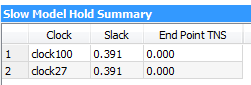


Figure 20 - Slow Model Hold Summary

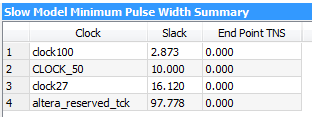


Figure 21 - Slow Model Minimum Pulse Width

C:\Users\Max\vlsi\Screenshot\Compilation Report\Slow Model Recovery.png

Figure 22 - Slow Model Recovery Summary

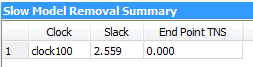


Figure 23 - Slow Model Removal Summary

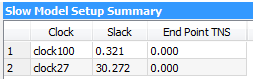


Figure 24 - Slow Model Setup Summary

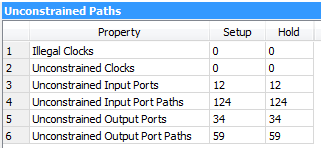


Figure 25 - Unconstrained Paths