Project report for

ET4351 VLSI Systems on Chip

Student: Papadopoulos Charalampos
Student Number: 4192141
Project: Image coprocessor in VHDL

Overview

The purpose of this project is to create a SoC that binarizes a grayscale image. The threshold algorithm chosen is Otsu's method. In the current project is partitioned in three pieces, the first is to calculate the histogram of intensity of the input image, the second is the calculation of the threshold given the histogram and the third is the binarization of the input image. The second piece is designed and implemented in hardware while the other two are implemented in software.

Design

The basic parts of the design are the processor Microblaze Lite (aka MBLite), a wishbone slave which implements part of Otsu's method, a wishbone master for interfacing that slave and an external memory that holds the input image. An abstract schematic is presented in figure 1.

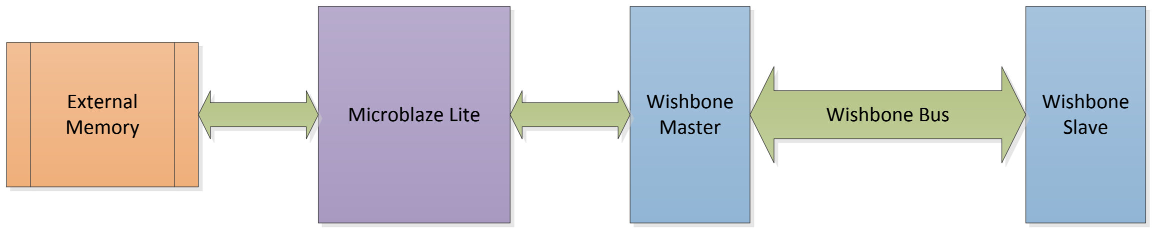


Figure 1: General schematic

With the aid of figure 1, a coarse grain explanation of the systems dataflow follows. We assume that the input image is preloaded in the external memory component. The dataflow begins with the MBLite reading the image from the external memory and calculating the histogram, which is passed to the Otsu slave via the wishbone bus. When Otsu produces the threshold value, MBLite reads it through the wishbone and finally MBLite starts binarizing the input image in place, overwriting the non binarized input image.

In more detail description, the system consists also of an address decoder which decodes the addresses issued by MBLite and selects the appropriate component and feed the data to the MBLite's data bus, a system controller which produces internal clocks and reset signals needed by the components, a clock divider for deriving internal clocks by the master (external) clock and a debouncer for the reset button. Also the MBLite uses imem and dmem local memories. The first is the instruction memory and the second the local data memory. A more detailed schematic, including the aforementioned components, is presented in figure 2.

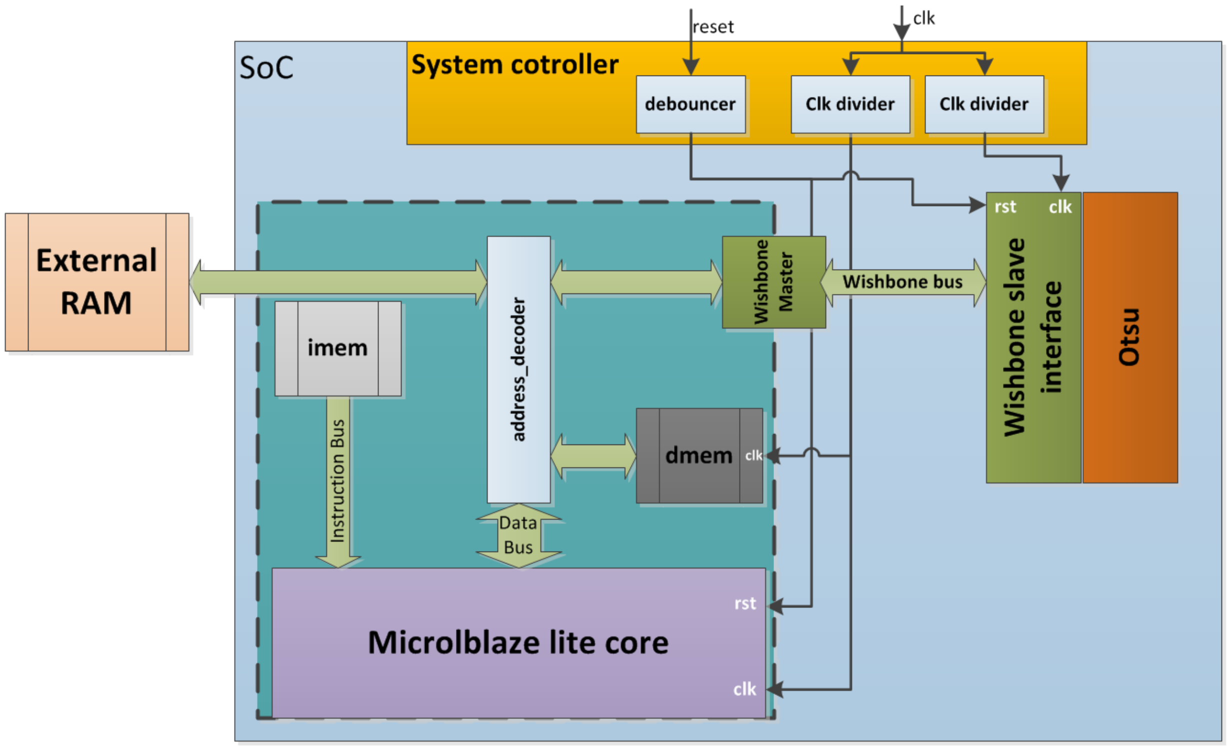


Figure 2: Detailed schematic

Otsu Algorithm

Otsu algorithm is presented with javascript code below (source: Wikipedia).

```
function otsu(histogram, total) {
   var sum = 0;
   for (var i = 1; i < 256; ++i)
      sum += i * histogram[i];
   var sumB = 0;
   var wB = 0;
   var wF = 0;
   var mB;
   var mF;
   var max = 0.0;
   var between = 0.0;
   var threshold1 = 0.0;
   var threshold2 = 0.0;
   for (var i = 0; i < 256; ++i) {
      wB += histogram[i];
      if (wB == 0)
         continue;
```

```
        wF = total - wB;
        if (wF == 0)
           break;
        sumB += i * histogram[i];
        mB = sumB / wB;
        mF = (sum - sumB) / wF;
        between = wB * wF * Math.pow(mB - mF, 2);
        if ( between >= max ) {
           threshold1 = i;
           if ( between > max ) {
              threshold2 = i;
           }
           max = between;
        }
     }
     return ( threshold1 + threshold2 ) / 2.0;
  }
```

The above algorithm takes as input the image histogram and the total number of pixels of the image. Since the image size is know from the start, the total parameter is omitted in the implementation. The output of the algorithm is a threshold value. The histogram calculation in is shown below in C (with known information that the image size is 160x240 pixels) :

```
void histogram(uint8_t *image[38400], uint32_t *histogram[256])
{
         for (i=0; i< 256; i++)
                        histogram [i]=0;

                  for (i=0; i< 38400; i++)
                        histogram[image[i]]++;
}
```

It takes an input image of size 38400 pixels, as 8bit an array and outputs the histogram of the image.

The histogram calculation is implemented in software while the rest of the algorithm is implemented in hardware.

The histogram consists of counts of the times of appearance of every color. Because every element it's a sum, the maximum value will be the total number of pixels. In our case it's 160x240 = 38400 → 2^{15}, so 15 bits are enough for the histogram elements. For **convenience and only**, in VHDL implementation, the 32 bit slave is used, and thus every register that interfaces with otsu will be 32bit.

Hardware

The system's input is an 8-bit grayscale image, thus the image data are stored as 8-bit unsigned values. From this image a histogram of the frequencies of appearance of every color must be produced. The histogram consists of counts of the times of appearance of every color. Because every element it's a count, the maximum value will be the total number of pixels. In our case it's 160x240 = 38400 → 2^{15}, so 15 bits are enough. For convenience in VHDL implementation, the 32 bit slave is used, and thus every register that interfaces with otsu will be 32bit. Having this histogram we can calculate a threshold using otsu's algorithm and finally using the threshold value we can binarize the grayscale image.

For the system's hardware, we had to design and implement the external RAM, the wishbone slave with otsu's logic and to adjust some parameters of the other components. Also we needed to create the imem and dmem memories using the given toolkits.

The external RAM is implemented as a byte array. The size was chosen to be exactly the size of the input image. The RAM is designed to hold bytes because the image is read as a byte stream, with each byte corresponding to the grayscale value of each pixel. Some glue logic was needed for the 32bit MBLite to cooperate with the 8bit RAM. The first one is during the write of a byte from MBLite to RAM. MBLite outputs a 32bit value to the data bus, which contains the byte to be written, but the position of the byte in the 4 byte word changes with respect to the target address, specifically the two least significant bits of the address. The appropriate logic was introduced to choose which byte of the word is going to RAM. The other part of the glue logic is needed when a byte is read from RAM. Here, MBLite expects the input byte to be positioned inside the 4byte word, according again with the least significant bits of the source address. This is easier to solve by cloning the value of the byte to the other three byte positions. The RAM is part of the test bench, located at tb_msim/tb_wb_soc.vhd.

The imem and dmem components are produced with the given toolkit, when the software of the system is compiled. The size of these memories is configured at 16Kbyte which is enough for the purpose of the system.

For the otsu slave, two components were implemented. The first is the slave's interface with the wishbone bus, including the interface registers and the second is the implementation of the Otsu's method. The slave, as mentioned before, accepts as input the histogram of the intensities of the given image, and outputs the calculated threshold. The input (from the slave's side) is chosen to be 16 registers of 32bit. Each register holds the summed values of an area of the histogram. The size of 32bits ensures that an image with 2^{32} pixels can be processed, because at the worst case, all the pixels will have the same intensity, and must be enumerated by one register. For the output threshold value, an output 32bit register exists. A control register exist, which has software reset command and an enable bit that is used to command the Otsu component to start the calculation. Both of these commands are automatically reseted, so there is no need from the software to do so. Finally a status register exists, with has The software reset is the second and the enable command is the third least significant bit respectively. The corresponding file is the wb_slave_ex32b.vhd.

Finally, Otsu's method was implemented with behavioral VHDL. The input of the component is a vector of 16 32bit values, which constitutes the histogram, and the output the threshold value. Also the control signals enable and ready exist. Care must be taken because the threshold value is calculated with the reduced histogram of 16 values. When binarizing, this value must be "decoded", multiplying it by 16 in this case.

Memory map and Slave's Registers List

The memory map of the system consists of an area for the external RAM and an area for the memory mapped slave registers. Figure 3 shows the memory map, and the memory mapped registers. Also the control and status register bits are shown, with "XX..X" notation for don't care bits range.

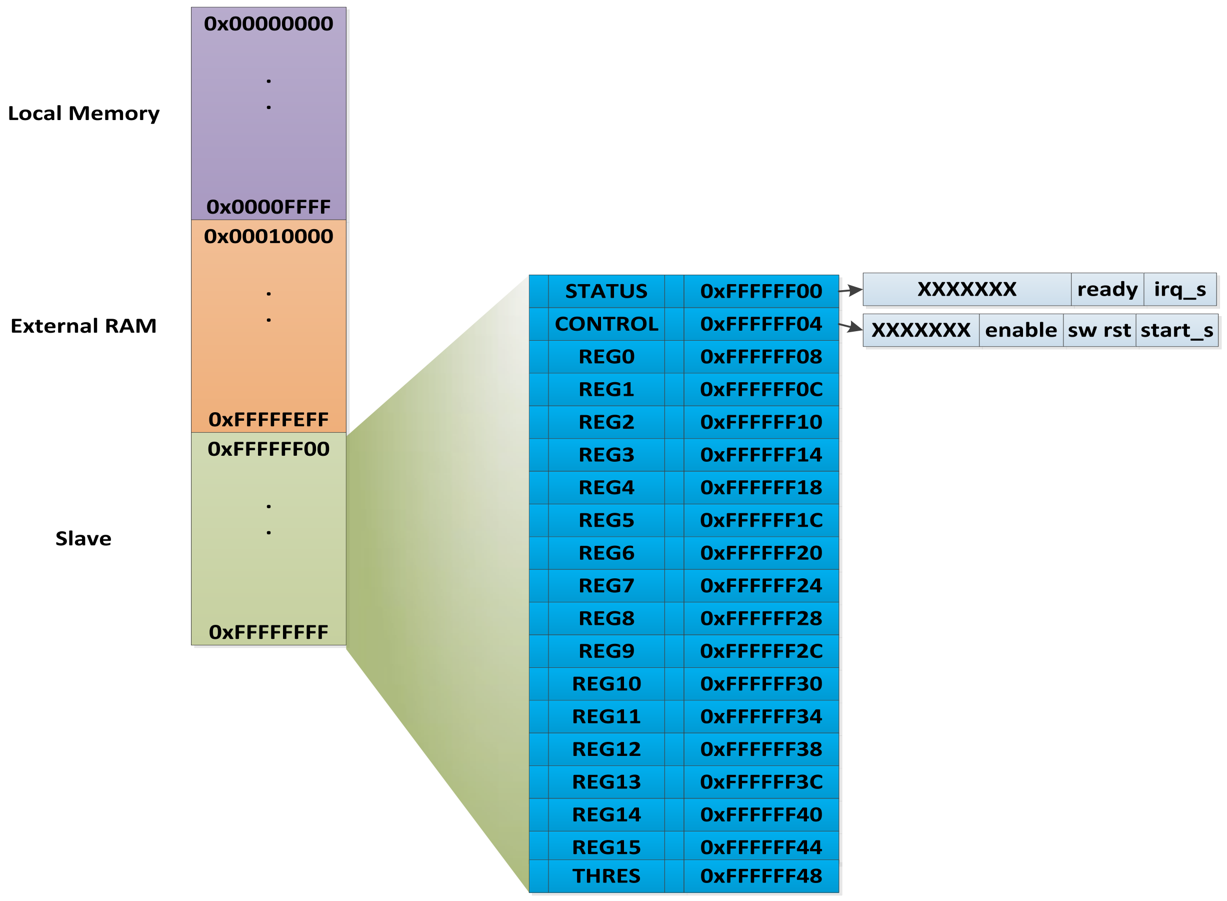


Figure 3: Memory Map

Software

The software of the system has two responsibilities. First to produce the histogram and feed it to the peripheral and second to binarize the image in RAM with the, calculated by the peripheral, threshold.

For the first part, the image is read byte-byte and the position at the histogram, corresponding to the value of each byte is augmented by one. After that, the histogram is reduced from an array of 256 to 16 rows, by adding the values of 16 groups of 16 consequent histogram values. Finally the reduced histogram is given to the slaves registers and the enable signal is given to the control register for the calculation to start.

For the second part, the threshold value calculated is read from the threshold register, then it is multiplied by 16 to take the correct threshold value and finally, the image's pixels are compared with the threshold, and are changed to black (0x00) or white (0xFF) when the pixel's value is less or greater respectively, than the value of the threshold. The c code is located at sw/otsu.c and sw/otsu.h files.

Simulation

For simulation and verification purposes, a test bench was used. This test bench produces the external clock and reset signals for the SoC. Also it initializes the external memory with the image file data, parsed by modelsim using the BinIO package. At the end, the binarized image is written back to a file to verify the results.

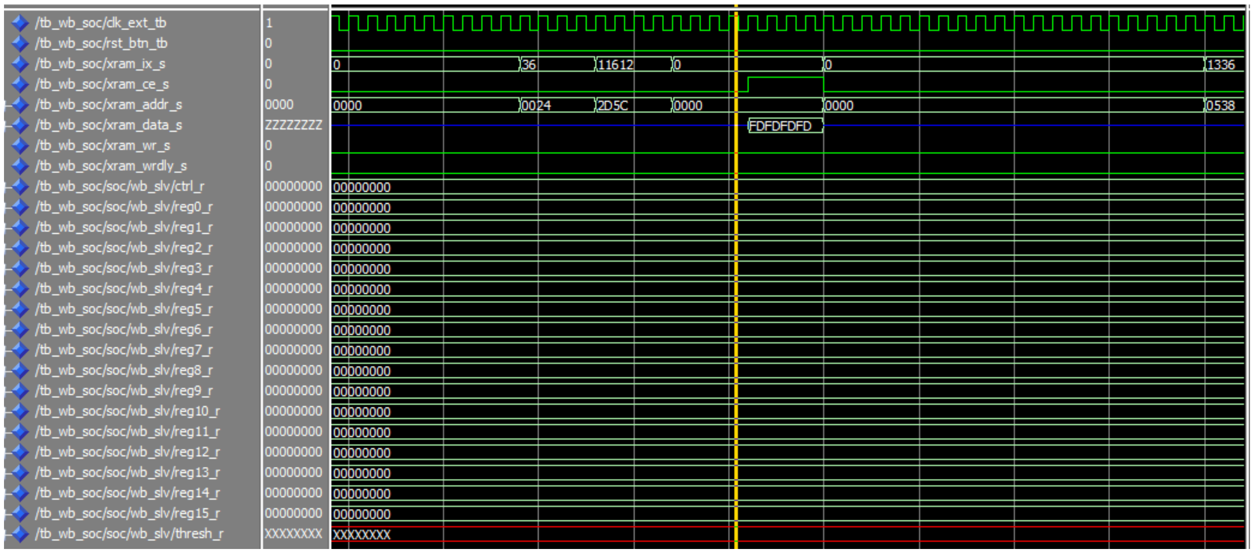
In specific, the clock is set with 50% duty cycle and 10ns period. Reset is issued at 10ns and zeroed at 150ns. At the rising edge of the reset signal, the BinIO package is used to populate the external memory, resulting with a preloaded image in external ram before the system begins. At the end of the simulation (approximately at 77129040ns) the binarized image is written to the hard drive.

The signals shown in the wave are the system clock and reset signals, the external RAM signals and the co-processors registers. More specifically the signals are:

- clk_ext_tb : The external clock
- rst_btn_tb : The external reset button
- xram_ix_s: The position that is being addressed in the external RAM (integer)
- xram_ce_s : External RAM chip select
- xram_addr_s : External RAM address bus (hexadecimal)
- xram_data_s : External RAM data bus (hexadecimal)
- xram_wr_s : External RAM Read/Write signal
- wb_slv/ctrl_r : Otsu slave control register (hexadecimal)
- wb_slv/reg0_r - reg15_r : Otsu slave input registers (histogram values, hexadecimals)
- wb_slv/thresh_r : Otsu slave output register (threshold value, hexadecimal)
- wb_slv/stat_r : Otsu slave status register (hexadecimal)

We can separate the simulation wave at regions.

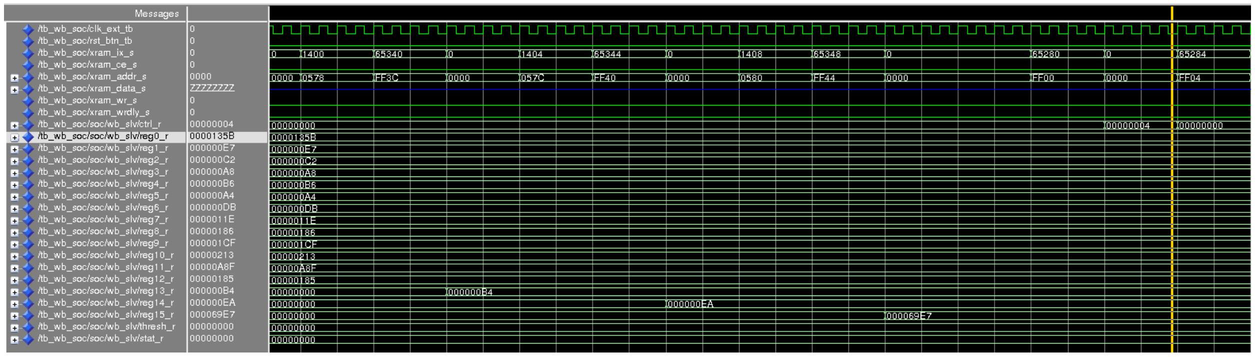
- The first region is until about 181130ns, at which the initialization of the software is done.
- In the next region, the compilation of the histogram takes place. This region ends at approximately 37387120ns.
- In the region that follows, the operation of the Otsu coprocessor is performed, until 3738160ns.
- Finally, at the last region, the image is being binarized.



```
/tb_wb_soc/clk_ext_tb              1
/tb_wb_soc/rst_btn_tb              0
/tb_wb_soc/xram_ix_s               0
/tb_wb_soc/xram_ce_s               0
/tb_wb_soc/xram_addr_s             0000
/tb_wb_soc/xram_data_s             ZZZZZZZZ
/tb_wb_soc/xram_wr_s               0
/tb_wb_soc/xram_wrdly_s            0
/tb_wb_soc/soc/wb_slv/ctrl_r       00000000
/tb_wb_soc/soc/wb_slv/reg0_r       00000000
/tb_wb_soc/soc/wb_slv/reg1_r       00000000
/tb_wb_soc/soc/wb_slv/reg2_r       00000000
/tb_wb_soc/soc/wb_slv/reg3_r       00000000
/tb_wb_soc/soc/wb_slv/reg4_r       00000000
/tb_wb_soc/soc/wb_slv/reg5_r       00000000
/tb_wb_soc/soc/wb_slv/reg6_r       00000000
/tb_wb_soc/soc/wb_slv/reg7_r       00000000
/tb_wb_soc/soc/wb_slv/reg8_r       00000000
/tb_wb_soc/soc/wb_slv/reg9_r       00000000
/tb_wb_soc/soc/wb_slv/reg10_r      00000000
/tb_wb_soc/soc/wb_slv/reg11_r      00000000
/tb_wb_soc/soc/wb_slv/reg12_r      00000000
/tb_wb_soc/soc/wb_slv/reg13_r      00000000
/tb_wb_soc/soc/wb_slv/reg14_r      00000000
/tb_wb_soc/soc/wb_slv/reg15_r      00000000
/tb_wb_soc/soc/wb_slv/thresh_r     XXXXXXXX
```

Figure 4: Start histogram compilation

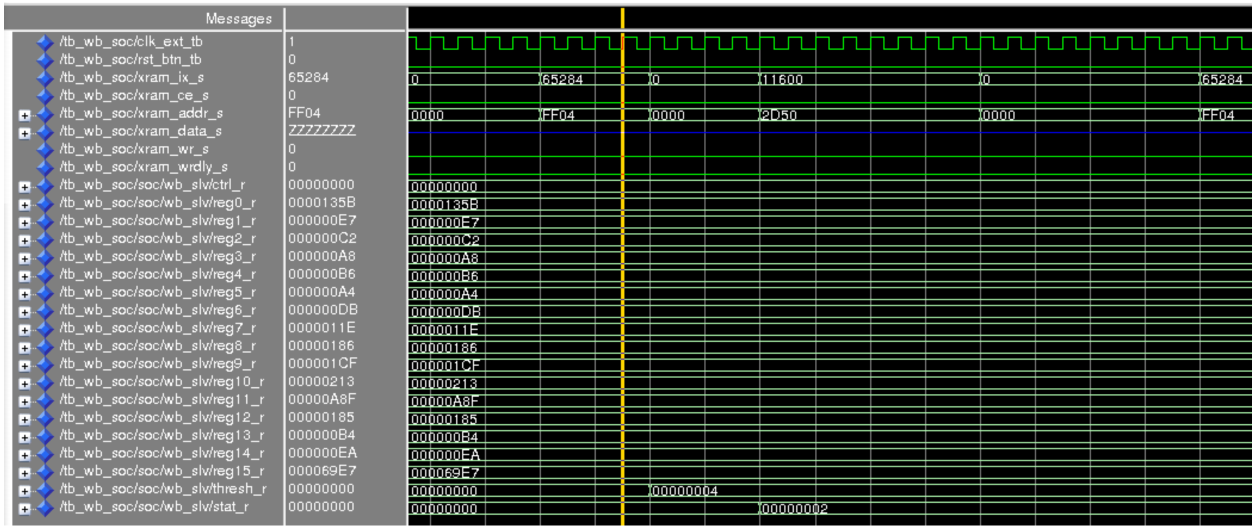
Figure 5 shows the start of the second region. There you can see the first read of the external RAM. This region consists of repeatedly reads to external RAM followed by some MBLite assembly execution.



```
Messages
/tb_wb_soc/clk_ext_tb              0
/tb_wb_soc/rst_btn_tb              0
/tb_wb_soc/xram_ix_s               0
/tb_wb_soc/xram_ce_s               0
/tb_wb_soc/xram_addr_s             0000
/tb_wb_soc/xram_data_s             ZZZZZZZZ
/tb_wb_soc/xram_wr_s               0
/tb_wb_soc/xram_wrdly_s            0
/tb_wb_soc/soc/wb_slv/ctrl_r       00000004
/tb_wb_soc/soc/wb_slv/reg0_r       0000135B
/tb_wb_soc/soc/wb_slv/reg1_r       000000E7
/tb_wb_soc/soc/wb_slv/reg2_r       000000C2
/tb_wb_soc/soc/wb_slv/reg3_r       000000A8
/tb_wb_soc/soc/wb_slv/reg4_r       000000B6
/tb_wb_soc/soc/wb_slv/reg5_r       000000A4
/tb_wb_soc/soc/wb_slv/reg6_r       000000DB
/tb_wb_soc/soc/wb_slv/reg7_r       0000011E
/tb_wb_soc/soc/wb_slv/reg8_r       00000186
/tb_wb_soc/soc/wb_slv/reg9_r       000001CF
/tb_wb_soc/soc/wb_slv/reg10_r      00000213
/tb_wb_soc/soc/wb_slv/reg11_r      00000A8F
/tb_wb_soc/soc/wb_slv/reg12_r      00000185
/tb_wb_soc/soc/wb_slv/reg13_r      000000B4
/tb_wb_soc/soc/wb_slv/reg14_r      000000EA
/tb_wb_soc/soc/wb_slv/reg15_r      000069E7
/tb_wb_soc/soc/wb_slv/thresh_r     00000000
/tb_wb_soc/soc/wb_slv/stat_r       00000000
```

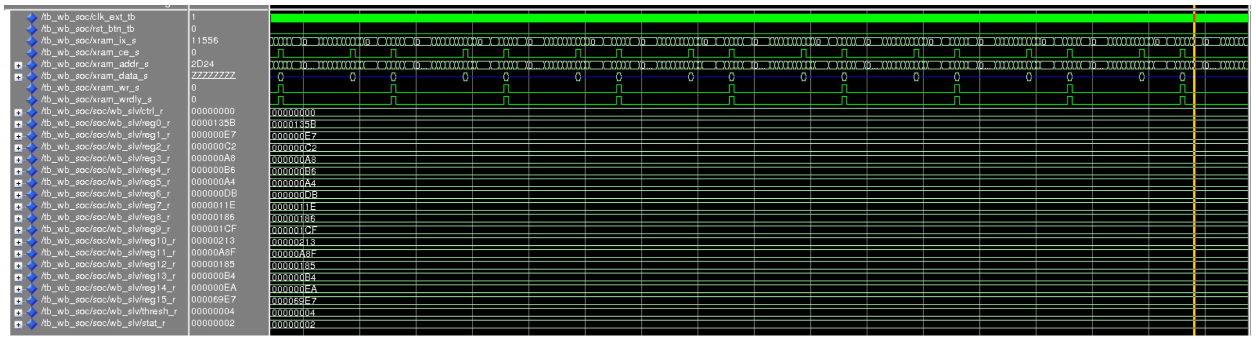
Figure 5: End of histogram compilation. Histogram is being written to slave registers

At figure 5 the simulation completed the histogram compilation and the histogram values are written at the input registers of the Otsu slave. Figure 6 shows the end of the Otsu processing with the threshold value populating the threshold register. In figure 7 the image is being binarized. The operations included are the comparison of each pixel of the image with the threshold followed by a write to the image pixel for all the pixels.



```
Messages
/tb_wb_soc/clk_ext_tb               1
/tb_wb_soc/rst_btn_tb               0
/tb_wb_soc/xram_ix_s                65284
/tb_wb_soc/xram_ce_s                0
/tb_wb_soc/xram_addr_s              FF04
/tb_wb_soc/xram_data_s              ZZZZZZZZ
/tb_wb_soc/xram_wr_s                0
/tb_wb_soc/xram_wrdly_s             0
/tb_wb_soc/soc/wb_slv/ctrl_r        00000000
/tb_wb_soc/soc/wb_slv/reg0_r        0000135B
/tb_wb_soc/soc/wb_slv/reg1_r        000000E7
/tb_wb_soc/soc/wb_slv/reg2_r        000000C2
/tb_wb_soc/soc/wb_slv/reg3_r        000000A8
/tb_wb_soc/soc/wb_slv/reg4_r        000000B6
/tb_wb_soc/soc/wb_slv/reg5_r        000000A4
/tb_wb_soc/soc/wb_slv/reg6_r        000000DB
/tb_wb_soc/soc/wb_slv/reg7_r        0000011E
/tb_wb_soc/soc/wb_slv/reg8_r        00000186
/tb_wb_soc/soc/wb_slv/reg9_r        000001CF
/tb_wb_soc/soc/wb_slv/reg10_r       00000213
/tb_wb_soc/soc/wb_slv/reg11_r       00000A8F
/tb_wb_soc/soc/wb_slv/reg12_r       00000185
/tb_wb_soc/soc/wb_slv/reg13_r       000000B4
/tb_wb_soc/soc/wb_slv/reg14_r       000000EA
/tb_wb_soc/soc/wb_slv/reg15_r       000069E7
/tb_wb_soc/soc/wb_slv/thresh_r      00000000
/tb_wb_soc/soc/wb_slv/stat_r        00000000
```

Figure 6: End of Otsu processing

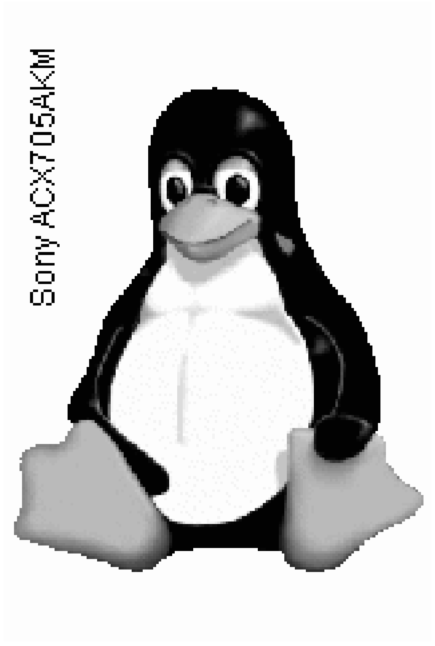


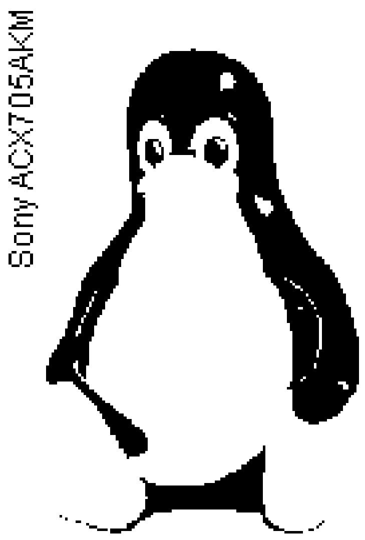
```
/tb_wb_soc/clk_ext_tb               1
/tb_wb_soc/rst_btn_tb               0
/tb_wb_soc/xram_ix_s                11556
/tb_wb_soc/xram_ce_s                0
/tb_wb_soc/xram_addr_s              2D24
/tb_wb_soc/xram_data_s              ZZZZZZZZ
/tb_wb_soc/xram_wr_s                0
/tb_wb_soc/xram_wrdly_s             0
/tb_wb_soc/soc/wb_slv/ctrl_r        00000000
/tb_wb_soc/soc/wb_slv/reg0_r        0000135B
/tb_wb_soc/soc/wb_slv/reg1_r        000000E7
/tb_wb_soc/soc/wb_slv/reg2_r        000000C2
/tb_wb_soc/soc/wb_slv/reg3_r        000000A8
/tb_wb_soc/soc/wb_slv/reg4_r        000000B6
/tb_wb_soc/soc/wb_slv/reg5_r        000000A4
/tb_wb_soc/soc/wb_slv/reg6_r        000000DB
/tb_wb_soc/soc/wb_slv/reg7_r        0000011E
/tb_wb_soc/soc/wb_slv/reg8_r        00000186
/tb_wb_soc/soc/wb_slv/reg9_r        000001CF
/tb_wb_soc/soc/wb_slv/reg10_r       00000213
/tb_wb_soc/soc/wb_slv/reg11_r       00000A8F
/tb_wb_soc/soc/wb_slv/reg12_r       00000185
/tb_wb_soc/soc/wb_slv/reg13_r       000000B4
/tb_wb_soc/soc/wb_slv/reg14_r       000000EA
/tb_wb_soc/soc/wb_slv/reg15_r       000069E7
/tb_wb_soc/soc/wb_slv/thresh_r      00000004
/tb_wb_soc/soc/wb_slv/stat_r        00000002
```

Figure 7: Binarization of the image

For the simulation, a Modelsim script file is included (simulation.do) in the deliverables, which performs the simulation, adding the most useful signals in the waveform. The input image must exist in the Modelsim project folder with the name “sony_penguin.bmp” and must be of the size 160x240 pixels.The binarized image will be produced in the same folder with the name “sony_penguin_binarized.bmp”. For further instructions about the simulation, please refer to README.txt.

The input and binarized images are shown below:





Synthesis

For the synthesis, Sinplify Pro 9.4 was used. The files used for the synthesis are those under the Synthesizable/ folder. This files differ at the BinIO package, which was excluded for the synthesis, because it is only meant for the simulation part. In figure 8 the schematic produced by Sinplify Pro is shown. This schematic shows the top-level of the design.

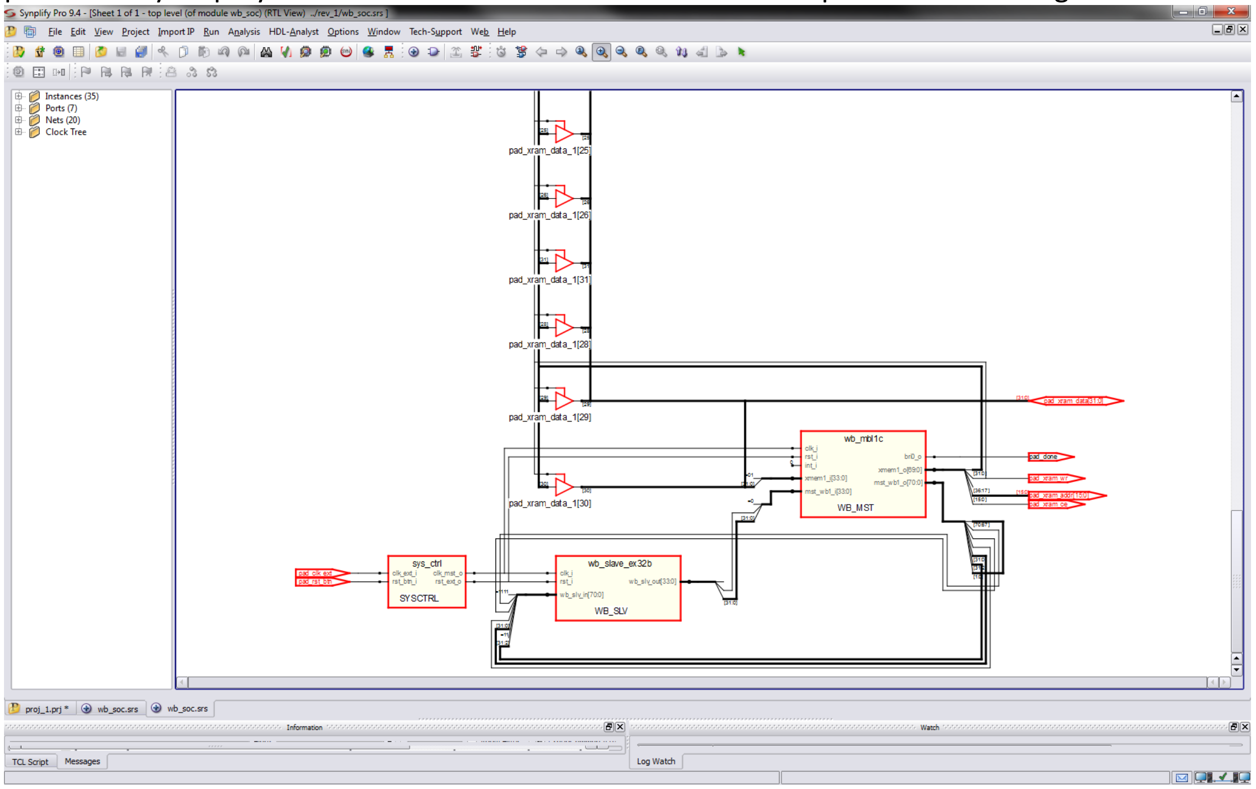


Figure 8: Synplify Pro Schematic

The resources in use by the design according to the Synplify's resource utilization report are:

Total LUTs: 9503
Frequency: 29 MHz

Error

The source of the error in the algorithm's operations is the division. The error produced by the divisions, is propagated and amplified by multiplications and additions/subtractions. Because the algorithm's datapath is complex in the sense that it has many paths, and the divisions are in a loop, I couldn't analyze theoretically the implementation's error. Thus a statistical approach was followed.

I produced 5 test images, run them through the javascript implementation and the SoC implementation, and compared the results. The table of the errors is shown above:

image	**vhd**	**js**	**error**
penguin	32	56	24
puppy	56	32	24
girl	8	40	32
penguin2	24	56	32
girl2	8	48	40
mean			30.4

The mean value of error was found to be 30,4, so the percentage of error is 30,4/256= **11.88%**, approximately.

File/Folder Listing

File	**Description**
SoC/	Folder where the simulation files are saved
address_decoder.vhd	MBLite's address decoder.
BinIO_pkg.vhd	Package to read and write binary files for simulation
clk_div.vhd	Clock divider
debouncer.vhd	Reset debouncer
dmem_init.vhd	MBLite's dmem
ET4351_Pkg.vhd	Projects package
imem_init.vhd	MBLite's imem
mbl1c_core.vhd	MBLite core
mst_wb_adapter.vhd	Master Wishbone adapter for MBLite
otsu2.vhd	Otsu Behavioral Implementation
otsu_Pkg.vhd	Otsu's Package
sys_ctrl.vhd	System controller
wb_mbl1c.vhd	The MBLite. Consists of imem, dmem, address

	decoder and the master wishbone adapter.
wb_slave_ex32b.vhd	Otsu's coprocessor Wishbone interface
wb_slave_Pkg.vhd	Package for wb_slave_ex32b.vhd
wb_soc.vhd	The SoC system. Consists of the MBLite, system controller and the wishbone slave
tb_msim/tb_wb_soc.vhd	Test bench for the SoC. Consists of the SoC, the external RAM and some code for the read /write of input/output image files.
simulation.do	Modelsim commands to reproduce the simulation
sw/otsu.h	System definitions for the software
sw/ otsu.c	The main program. Creates the histogram and passes it to the Otsu slave, reads the calculated threshold value and finally binarizes the image.
sw/Makefile	Makefile used to compile the SoC's software
sw/mem_defs.ld	Memory layout definition used to inform the linker during the build of the SoC's software.
wb_soc.srs	Synmplify Pro schematic file produced by synthesis
sony_penguin.bmp	Test case image
sony_penguin_binarized.bmp	The result binarized image. It will be overwritten when a new simulation starts.
README.txt	Instructions for reproducing the simulation
Synthesizable/	Folder in which the synthesizable vhdl is saved
Scratch.srr	Sinplify srr file