Binarization Image Processing synthesized SystemC

Dirk Vos 4391470
Michiel van der Vlag 4369327

June 4, 2017

1 Introduction

The project goal is to implement a well known algorithm for gray-scale image manipulations. We will investigate a dynamic image threshold algorithm (binarization of a gray-scale image). The algorithm implemented is called Otsu's moment preservation.

The goal is to load an image in memory, after which the co-processor performs the algorithm for binarization and communicates back the resulting image.

The intended co-processor works in an environment where the Wishbone bus connects all components of the SoC. This algorithm is for gray-scale images, and targeted to fixed-point arithmetic. We have found a suitable hardware architecture for such a co-processor and have implemented it using high-level SystemC.

We have succeeded in starting the loading of the image in the hardware controlled from software. This image is then communicated to the co-processor which performs the gray-scaling. After which the image is returned and written to a file.

2 Binarization algorithms

Binarization is the process of converting a pixel image to a binary image. All pixels of an image are compared to a certain threshold, after which they are either black or white. Binarization is often an important step to extract information from images. As simple is its principle, as hard is its execution. In greyscaled pictures there is often a lot of intensities due to shading or noise.
It is important to choose the right threshold value. A value to low or to high can result in loss of information of the image. There is no single optimally suited thresholding method and it is often found due to experimentation.

The method for finding an optimal threshold that was implemented on hardware for this project belongs to the category of global techniques. Such techniques use the intensity histogram to identify a threshold between foreground and background intensity values. These methods perform better for images with uniform backgrounds because threshold selection is based on a larger data set of pixels.

Otsu's moment preservation method performs binarization with respect to an automatically determined intensity threshold; the input gray-level image is converted to a binary output image.

3 Detailed project design

3.1 The architecture

As is visible in figure 1, the test bench consists of a (lite) MicroBlaze (MB) processor, a FileHandler, a memory block and a Wishbone bridge.

The MB processor holds the c code that sends the in.bmp file over a TLM bus to the wishbone bridge, which will send it over a wishbone connection to the co-processor we designed. When the software is started the user can select a threshold calculation method through a console menu in order to calculate the threshold value for binarization. After which the image is send to the co-processor. The next step is starting the binarization. After the binarization the MB processor can send a read command in order to retrieve the processed image from the coprocessor and write it to out.bmp. The MB uses the TLM bus interface.

The FileHandler reads an .bmp image from in.bmp and stores in its local storage memory in an array format containing all the pixels which consists of 32bit per pixel: 8bit for the alpha component, 8bit red component, 8bit blue component, 8bit green component. Our designed co-processor will expect gray-scaled images which means the red, blue and green component all have the same value. The FileHandler writes its local stored image to out.bmp. It uses the TLM interface.

The memory block is not used since we fully utilize the storage space of the FileHandler and the storage place of the co-processor. It uses a TLM bus interface.

The wishbone bridge will act as a translator that sends messages that were sent over the TLM bus to messages over the wishbone bridge to connected slaves. Which in our case is only the co-processor programmed for the binarization process. The bridge will use both TLM and wishbone interface.

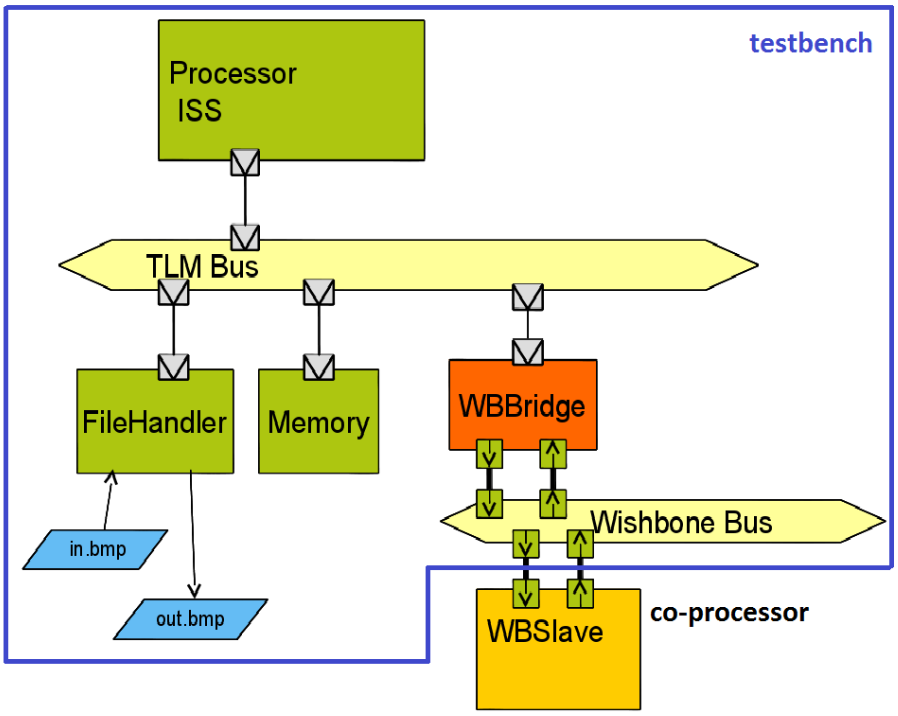


Figure 1: Top level architecture of testbench and co-processor

Communicating over the TLM bus works with memory mapping: as you can see from figure 3.

The wishbone bridge is connected to the TLM bus at address 0xC0000000, with all of its registers starting from that address up to 0xC0040010. Including 1 word $S1\_CTRL\_REG$ at 0x0000, 1 word $S1\_STAT\_REG$ at 0x0004, 1 word $S1\_DELAY$ at 0x0008, 1 word $S1\_LENGTH$ at 0x000C and all data bytes for the data starting at 0x0010 up to 0x40010 (65k words/pixels).

The CTRL register holds all commands that the MB can send: $RESET$ (clears all registers starting from LENGTH and clear local storage memory for the processed image), $FIXED\_PROCESSING$(binarizes the image using a fixed threshold), $OTSU\_PROCESSING$(binarizes the image using a threshold calculated with the otsu algorithm), $MOMENTS\_PROCESSING$(binarizes the image using a threshold calculated with the preservation of moments algorithm), $ENTROPY\_PROCESSING$ (binarizes the image using a threshold calculated with the maximum entropy algorithm), $KITTLER\_PROCESSING$ (binarizes the image using a threshold calculated with the Kittler algorithm) and $WRITE$ (sends back the image to the MB processor). Writing to these addresses will send the data over the TLM bus to that specific address. The wishbone bridge will then translate that address to an address on the wishbone slave: the 32bit address is first normalized to the wishbone bridge base address on the TLM bus (translating 0xC0000000 into 0x00000000), then the address is divided into a slave address and an address on that slave. The top 8 bits are slave addressing and the lower 24 bits are the addresses on that slave (see figure 2, here you can see a maximum of 256 slaves are supported on the wishbone bridge each with an address space of $\approx$ 16Mbytes).

The STAT register holds the handshaking commands, if the co-processor is ready with its task it will indicate with the $S1\_READY\_STATE$ message.

The DELAY register holds the delay the wishbone bridge uses before sending messages through the wishbone interface to the slaves.

The LENGTH register is used to store the amount of pixels of the image stored at the data registers.

The DATA registers hold all the pixels of the image, we choose to put the complete image in the registers because that way we do not need to use software handshaking using the STAT register every time, this way we fully utilize the hardware handshaking of the wishbone bus and are able to send all the pixels one after another over the wishbone connection.

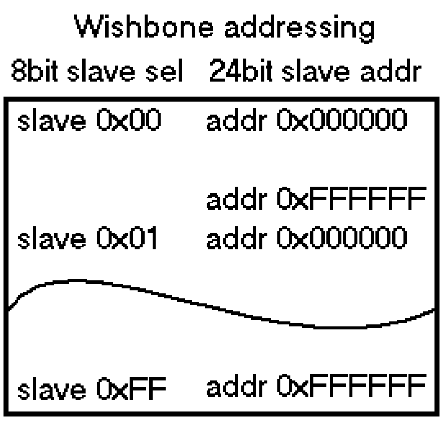


Figure 2: address translation of the wishbone bridge

The FileHandler is connected at address 0xC1000000 on the TLM bus, is has a double mapping, so the read registers start at the base address of 0xC1000000 but also the write registers start at that position. It will discriminate between the two by looking at the access type (read or write).

A wrapper has been made for the read registers: getFileWidth() will access the first register and will return the file width, getFileHeight() will access the second register at 0x04 and will return the file height, getFilelength() accesses the third register and will return the total amount of pixels of the image. The function readFile(address) will access image pixels, the parameter address is normalized so 0x00 will actually access register 0x0C.

The write register also has a wrapper build around it for easy access:
writeToFile(address, word) will write the parameter "word" to the address indicated by the parameter "address" (also normalized just like the readFile function) to the local image stored by filehandler; after which startFileWrite() can be used to initiate a write from the FileHandlers local image storage to the out.bmp file.

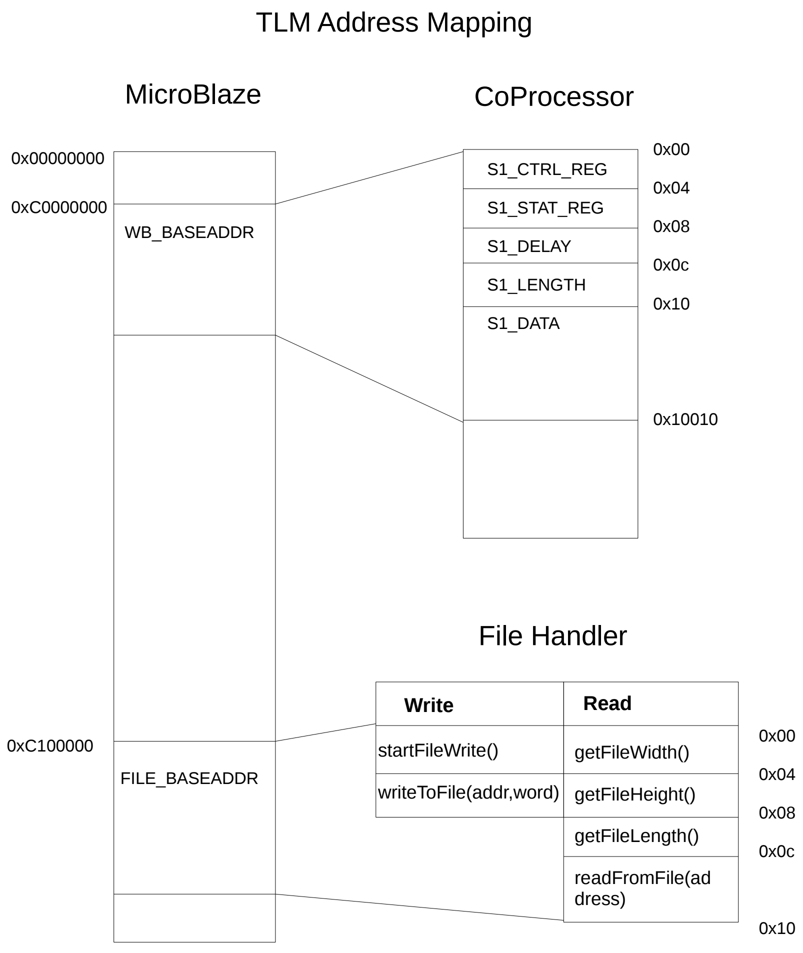


Figure 3: Memory map TLM to Co-Processor and FileHandler.

3.2 Microblaze

The Microblaze (MB) controls the execution of the co-processor. When the execution starts, the program will ask the user which threshold calculation algorithm it should use on the co-processor.

When the choice has been made, the MB will first send a reset command to the co-processor which will clear the data registers (this command could be omitted, but it is a nice feature for possible future users of this co-processor to have a reset option). Sending a reset command to the co-processor is much faster as clearing all the data registers piece by piece using the MB.

After the reset of the co-processor the MB will retrieve the image size from the FileHandler with the function getFileLength() which it will write to the co-processor $S1\_LENGTH$ register through the wishbone bridge.

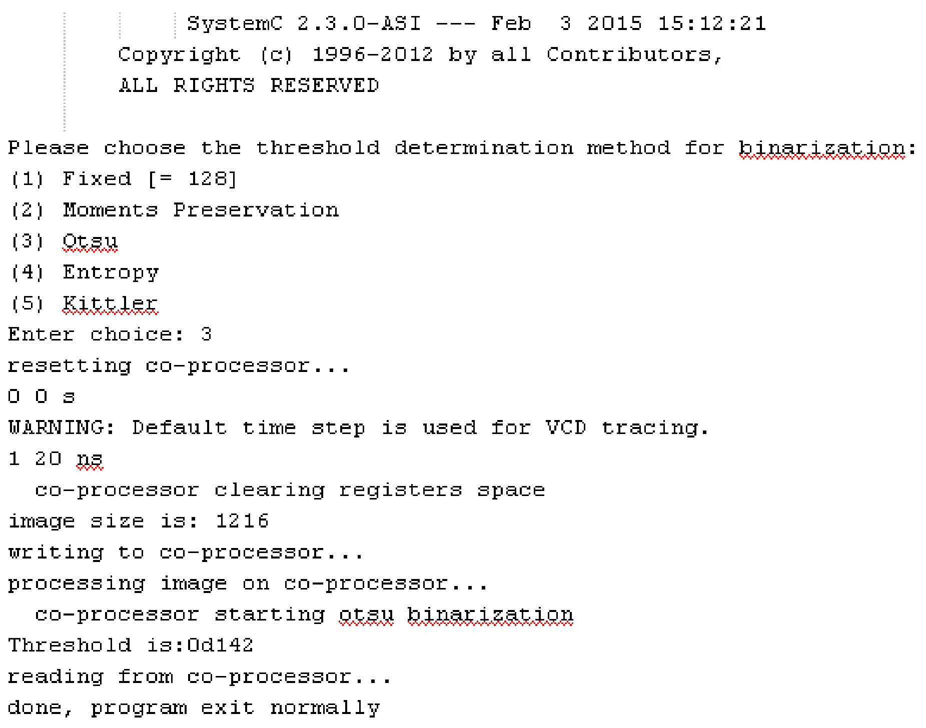
Now the data is read from the local copy in the FileHandler and written to all the data registers of the co-processor using a for loop construction and the $S1\_DATA$ register. This will send all pixels right after one another fully utilizing hardware handshaking of the wishbone bridge without the need of software handshaking through the CTRL and STAT registers of the co-processor.

The image is now on the co-processor so the MB will issue the previously chosen algorithm on the co-processor by writing the appropriate command in the CTRL register. After the co-processor has finished the image processing it will send the ready command by writing to the STAT register the ready status, the MB will detect this and will continue its execution (software handshaking).

The image is now processed and in the data registers of the co-processor, the only thing left to do now is retrieve it with the MB using the $S1\_DATA$ register and transfer it to the FileHandler using the writeToFile() function. The image will be copied to the locally stored image on the FileHandler.

Now we only have to trigger a bmp file write to out.bmp by the FileHandler by issuing a startFileWrite() by the MB.

An example of the output to the console is given in figure 4.



```
             SystemC 2.3.0-ASI --- Feb  3 2015 15:12:21
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Please choose the threshold determination method for binarization:
(1) Fixed [= 128]
(2) Moments Preservation
(3) Otsu
(4) Entropy
(5) Kittler
Enter choice: 3
resetting co-processor...
0 0 s
WARNING: Default time step is used for VCD tracing.
1 20 ns
  co-processor clearing registers space
image size is: 1216
writing to co-processor...
processing image on co-processor...
  co-processor starting otsu binarization
Threshold is:0d142
reading from co-processor...
done, program exit normally
```

Figure 4: console output to the user

3.3 Wave/Signals

This section describes the wave dump of the significant control and data signals of the simulation. We differentiate three important phases in our simulation. Firstly, the reading commands and sending of the picture to the co-processor registers. Secondly, the command to start the binarization process with the thresholding option and thirdly, returning the image to the FileHandler which writes the image directly to its memory and after that to the output file out.bmp.

The wave plots in the following figures contain the most relevant signals to display the functionality of our hardware design and are all input or output signals from the co-processor: ack_o is the acknowledge signal from the handshake. This is set every time the co-processor receives data. The adr_i[31:0] resembles the address of the registers on the co-processor expressed in bytes. The dat_i[31:0] and dat_o[31:0] are respectively the input and the output data. This is the channel on which the image is sent and received.

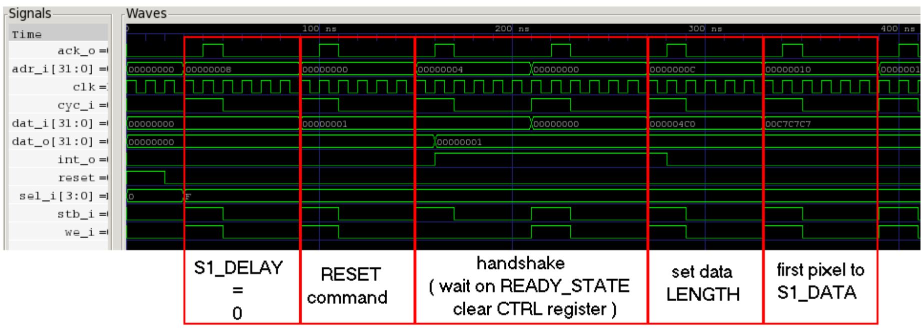


Figure 5: reset and writing through the wishbone interface.

In figure 5 the first action is putting the DELAY register to zero, so messages are send directly through the wishbone bus. Second action is the reset command which makes the co-processor clear all data registers and the LENGTH register. Third action is writing the image size to the LENGTH register on the co-processor. S1_LENGTH is defined on 0x0c, which has the value ox4c0 (= 0d1216) and this resembles the length of the file (33x38). Fourth action is start sending the pixels right after one other on the wishbone bus to the data registers of the co-processor. We see that the first value of the pixel 0x00C7C7C7 is written to the first register of the co-processor 0x10. This is done by the command *(&S1_DATA+i) = readFromFile(i). readFromFile(address) is defined as WBR_MEM32((FILE_BASEADDR + 0x0C + (address<<2))). Then every 4 bytes of the image is transported.

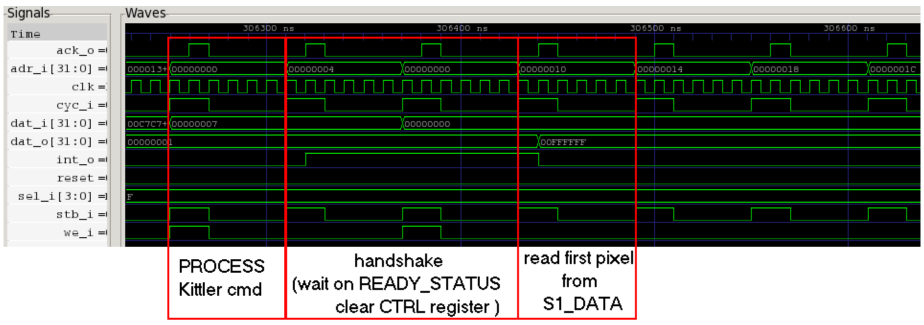


Figure 6: binarization and reading back through the wishbone interface

In figure 6 we can just see the last register for the image is written (0x0000130c). Then the start command for binarization is given. S1_CTRL_REG is set to the choice given by the interface. In this case the option is 7 which is the option for the Kittler algorithm. The algorithm is executed on co-processor until handshake(). The handshake consists of the MB waiting for the STAT register to be set to READY_STATE by the co-processor after which the MB will clear the CTRL register. After the handshake the MB starts reading all the pixels back from the registers of the co-processor and into the local copy of the image on the FileHandler. In the code this shows as writeToFile(i,*(&S1_DATA+i)); command is issued which makes the co-processor start sending the binarized file directly to the FileHandler. The register 0x10 is the first to be read register which has the first - in this case - white pixel (0x00FFFFFF). We can see from the first 4 pixels that binarization worked, since there are all white now and not a gray color somewhere between black and white.

3.4 Synthesizability

The project folder consists of several subfolders:

- sw, this folder contains the .c and .h files for the MB processor.
- synth, this folder holds the .cpp and .h files that represent the co-processor in SystemC code.
- tb, this folder contains the testbench .cpp and .h files we used to test the co-processor.
- the root holds the in.bmp and out.bmp that are used as the bitmaps which are read and written to.

To construct the MB executable wbs32_test we used the Makefile in de sw folder and invoked it with the make command. To synthesize the co-processor and the testbench into the executable top we used the Makefile in the tb folder and invoked it with the make command. To simulate execution we executed the top file in the tb folder giving it the parameter ../sw/wbs32_test. The console will then ask the user for a threshold algorithm to use for the binarization and will output further execution, the out.bmp in the root of the project folder will now contain the processed image.

4 Multi person project justification

We both started exploring the functionality of the given source files for the project. When we both understood the functionality of the code and how the memory mapping was realized, we decided to split some tasks. Dirk focused on implementing the binarizing algorithms and fixed point calculations on the co-processor. Michiel focused on the communication between the filehandler and the co-processor using the wishbone architecture, gray-scaling algorithms and console menu. When both were finished we merged them and worked on the fixed point implementation. The report was done 50/50.

We both were new to the SystemC language and learned a great deal about this very nice alternative to writing VHDL code by hand. SystemC allows for better portability of algorithms across different platforms and is easy to learn since it uses C++ syntax. For a future hardware project the SystemC language will definitely be considered.

5 File listing

This section lists every file that was written or modified for the project.

Hardware:

1. *top.cpp*: top level file which connects the hardware. Was modified such that file handler was connected and that memory was correctly mapped.
2. *wb_slave.cpp*: slave module which is connected to the MB via wishbone and which contains the gray-scaling algorithm (*co_processor*).
3. *wb_slave.h*: header file of the slave module in which the number of bits for slave addressing was adjusted.

Software:

1. *wb32_test.c*: contains the instructions to be executed by hardware. Loading file; sending file to co-processor, start signal to perform gray-scaling, receiving image and writing it to file.
2. *wb32_test.h*: header file
3. *mbl_wbs32.h*: header file containing information on how to control hardware via addresses.

6 Results

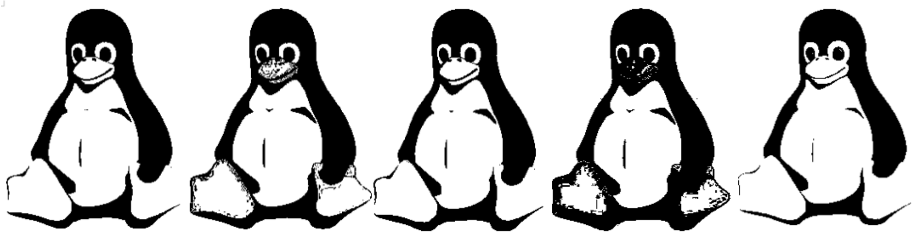


Figure 7: Results for Fixed, Moment, Otsu, Entropy and Kittler thresholding

All the thresholding algorithms were tested for the images of the highest resolution provided. The thresholding values for the algorithms were respectively: 0d128, 0d164, 0d141, 0d171 and 0d88.

From the images we conclude that the fixed or the Otsu algorithm have the best results because they are the most detailed. The thresholding values are close to one other. From the worst results we conclude that algorithms like moments and entropy that 0d164 or higher doesn't improve the quality of the binarized file.
As was stated in the theory of binarization; the best algorithm is found through experimentation.