Image Co-Processor: Image Thresholding SoC Design using SystemC

ET4351: VLSI Systems on Chip

Delft University of Technology
July 3, 2016

Group Members:

Muneeb Yousaf
4411129
m.m.yousaf@student.tudelft.nl

Christopher McGirr
4415302
c.a.mcgirr-1@student.tudelft.nl

Abstract

The following project demonstrates the successful implementation of a binary image thresholding using entropy for a System on Chip (SoC) design using the hardware modelling language known as SystemC. With analysis of the algorithm, a fixed point implementation was created to accelerate the key parts of the algorithm. The result is a system which uses both a MBLite RISC style processor and a co-processor to implement an image processing algorithm. Although, the fixed point implementation was thought to be the more area and performance efficient than the floating precision the opposite was shown. As the only benefit in the fixed point precision was the reduced memory needed.

1 Introduction

The move towards integrated Systems on Chips (SoCs) can be seen more and more as the demands of applications and developers focus on cost, area and power. SoCs are a perfect example of combining all the facets of electronics into one convenient chip including the digital and analog processing.

However designing such a large and intricate system requires expertise and experience for most designers. That is why there have been movements to move the design of such systems to a higher level. This includes the abstraction of hardware in terms of a Hardware Description Language such as VHDL or SystemC.

In this Project, we will be tackling the design of a simple SoC using the SystemC hardware modelling language. The application which will be implemented on a Co-Processor is a image thresholding application which converts a gray scale image into a binary image using a dynamic threshold.

The following sections will focus on the analysis of the application, the general architecture of the solution including the software and hardware and final the results of synthesis.

2 Application Analysis

The image application which we have chosen for this project is the Binary Thresholding using entropy. The algorithm is described below.

```
int i,j;
double Hn, Ps, Hs;
double psi, psiMax;

for (i = 0, Hn = 0.0; i < NHIST; i++){
        if (prob[i] != 0.0){
                Hn -= prob[i] * log(prob[i]);
        }
}
for (i = 1, psiMax = 0.0; i < NHIST; i++) {
        for (j = 0, Ps = Hs = 0.0; j < i; j++) {
                Ps += prob[j];
                if (prob[j] > 0.0){
                        Hs -= prob[j] * log(prob[j]);
                }
        }
        if (Ps > 0.0 && Ps < 1.0)
                psi = log(Ps - Ps * Ps) + Hs / Ps + (Hn - Hs) / (1.0 - Ps);
        if (psi > psiMax) {
                psiMax = psi;
                (*thresh) = i;
        }
}
```

Figure 1: Binary Thresholding using Entropy Method

The above thresholding algorithm uses the concept of entropy to measure the separation of two classes of data. In this case the background and foreground. The separation is then done for all intensities of the histogram. For each intensity level of entropies are summed and the maximum of among the levels are considered the optimum.

Now the first step in converting this algorithm to an hardware implementation is to identify the areas of the algorithm that utilize the most resources. This was done by using the Valgrind tool. Though the profiler is not suited for hardware implementations, it can help give us an idea of how much time is spent in each part of the application.

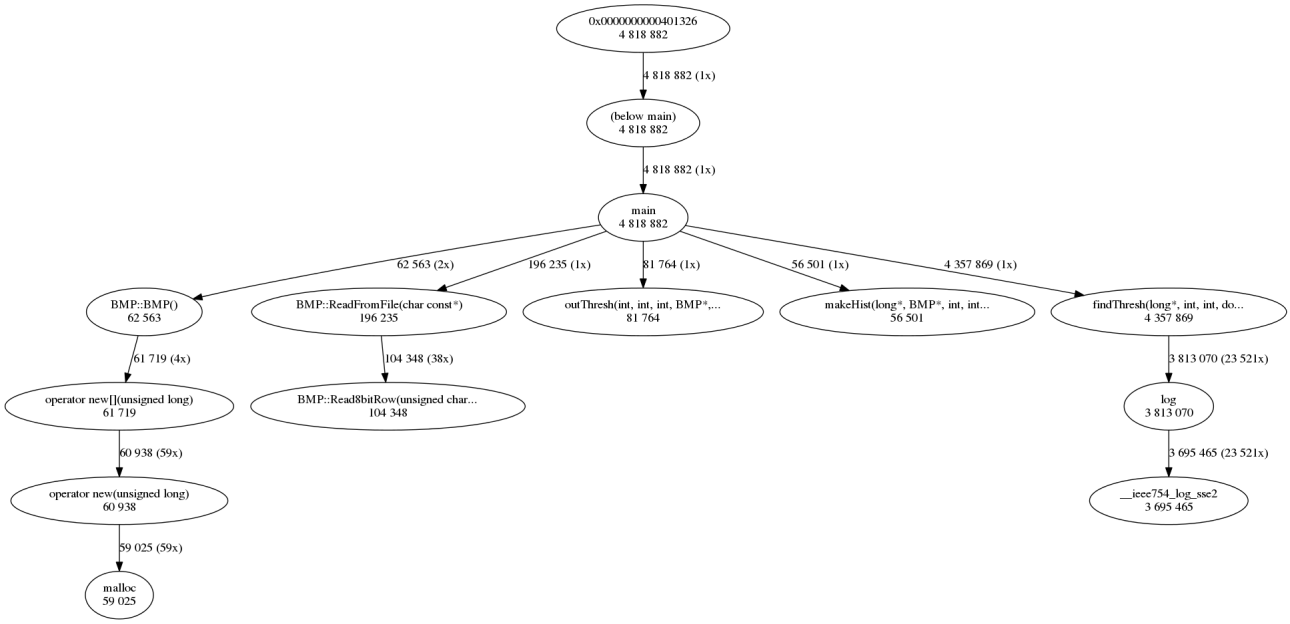

Figure 2: Program Profile using Valgrind Profiler

Figure 2 demonstrates the amount of clock cycles used in each function. As it shows most of the time is spent calculating the natural logarithm. Therefore, to achieve the most performance the whole algorithm of Figure 1 and the calculation of the natural logarithm must be implemented in hardware. The calculation of the image histogram and the converting of the grayscale image to a binary image will all then be done in software on the MB-Lite Core.

The second part of the analysis was to look at the maximum and minimum values of the variables in Figure 1 so that an idea of what widths could be used for a fixed point implementation. The advantages of the fixed point implementation are the performance and area usage as the hardware logic required to implement when compared to the floating point precision is much less.

To give an idea of the range of values for each of the variables involved, a variety of images where used to estimate the range.

Figure 3: Image Penguin

Figure 4: Image Lena

Figure 5: Image Cameraman

Figure 6: Three images used to test the implementation

Penguin		
Variable	Min	Max
prob[i]	0	0.31
Hn	6.95e-310	3.26
Ps	1.21e-312	0.87
Hs	0.247	3.00
psi	9.88e-324	5.79
psiMax	1.21e-312	5.79

Lena		
Variable	Min	Max
prob[i]	0	0.018
Hn	6.95e-310	4.74
Ps	1.21e-312	1
Hs	0.0037	4.74
psi	9.88e-324	8.086
psiMax	2.09e-317	8.07

Penguin		
Variable	Min	Max
prob[i]	0	0.028
Hn	6.95e-310	4.78
Ps	1.21e-312	1
Hs	0.0037	4.78
psi	9.88e-324	8.23
psiMax	2.09e-317	8.23

Table 1: Max and Min Values for all variables

Table 1 demonstrates the range of values for the variables used in the algorithm. As we can see the maximum number for a small set of images is around 8. With a minimum value of very close to 0. Therefore a fixed point implementation that uses an integer width of 5bits should be sufficient. For the fractional part a length of 16bits will be used to approximate as close as possible these small values. Note we have restricted ourselves to images sizes of 128x128 as with increased number of pixels means that the number of fractional bits required to represent a number increases. This is due to the fact that in order to represent the probability of an intensity level which one pixel requires at least a fixed point representation that can hold $1/n$ where n is the total number of pixels.

With the amount of fixed point precision determined and which part of the algorithm to implement, we can now move ahead to the SoC implementation.

3 Architecture

Using the SoC approach of design gives us the ability to utilize both software and hardware of the system. In the following sections a description of the software and hardware architectures will be given.

3.1 Software Architecture

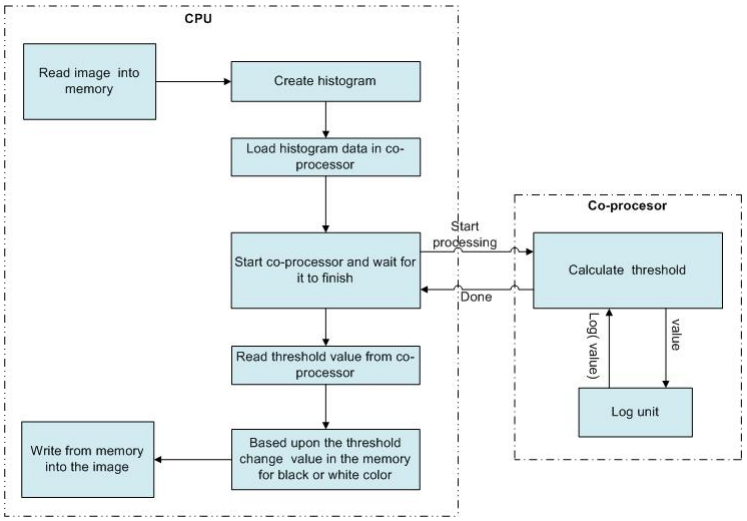

Figure 7: Software architecture

In Figure 7 we can see the flow of execution of the algorithm. On the MBLite Processor we calculate the histogram of the image. This histogram represents the number of pixels corresponding to a certain intensity. Where the intensities vary from 0-255 as we are dealing with an 8-bit grayscale image.

The software then simply passes the histogram to the buffer of the Wishbone bus at which point it then executes the co-processor with a simply set of flags to indicate the histogram data is ready to be grabbed and the threshold to be calculated.

The co-processor then returns a done flag when the execution is complete and the MBLite processor can then read the threshold value and begin transforming the grayscale image to a binary one.

Register	**Description**
S1_REG0	Used to pass the Wishbone Base address
S1_REG1	Used to pass the number of pixels in the image
S1_CTRL_REG	Used to start and reset the co-processor. *S1_START_CMD* for starting the co-processor. *S1_SW_RESET_CMD and S1_CLEAR_CMD* to clear and reset the co-processor.
S1_STAT_REG	Used to check for the done flag of *S1_READY_STATE*

Table 2: Registers used to execute the co-processor

In Table 2 the registers used to control the co-processor are shown. The MBLite processor simply passes the number of pixels and the Wishbone Base address and sets the start flag to let the co-processor run. While it waits for the ready flag to indicate that the co-processor has completed its task.

3.2 Hardware Architecture

For this project, the full system is provided with a simple implemented wishbone-slave that does not have any processing capability. In addition, to provide the system the ability to read *.bmp files a file handler is connected to the SoC. The processor of the SoC is modified version of the MB-Lite processor, which in turn is an open source alternative to Xilinx's MicroBlaze soft core.

The MB-Lite processor has a 32 bit RISC design and can execute almost all instructions in one clock cycle. The soft core used in this project is the MBL1c, a version of the MB-Lite where all instructions are executed in one clock cycle. The MBL1c processor is connected to a TLM bus for communication with the memory and peripherals. Memory mapped I/O is used to communicate with peripherals. The peripherals in this SoC are memories, file handler and wishbone bridge. These peripherals are all connected to TLM bus to communicate with processor. Wishbone bridge translates the signals from TLM bus signals to wishbone compatible signals. Therefore in order to meet the requirements of this assignment, in this design of SoC, image co-processor is attached behind the wishbone bridge. The full SoC design is shown in the Figure 8.

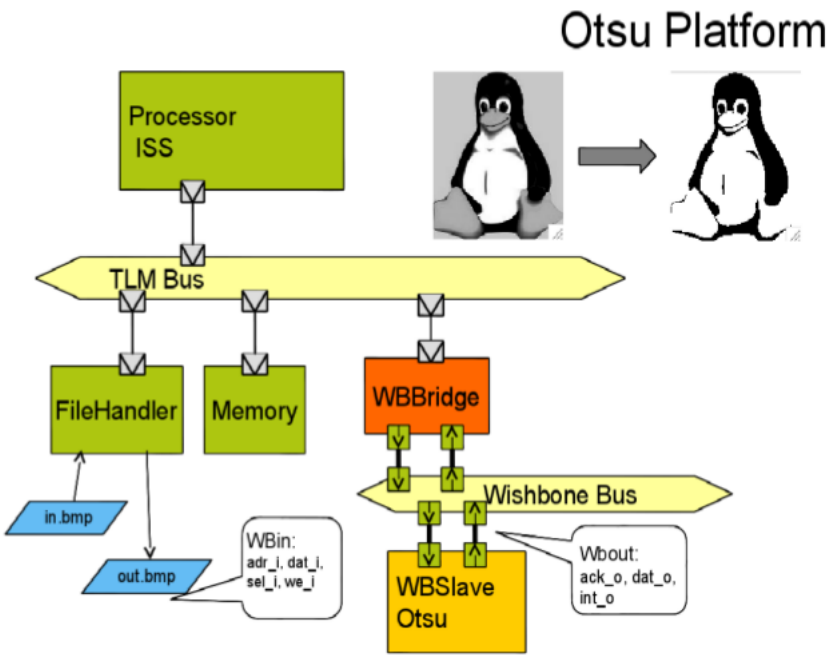

Figure 8: hardware Architecture

3.3 Memory map

All peripherals are addressed by using memory mapped I/O. The provided system has two memories(memory and external memory), a console , interrupt handler, a wish bone bridge and a file handler. The memory map of the SoC is shown in the Table 3.

Device	**Memory base – Size**
Memory	0x0000000 – 0x4000000
Interrupt handler	0x17000000 –
Console base	0x40000000 –
Co-processor/Wishbone base	0xC0000000 –
External Memory(XRAM)	0xC1000000 – 0x20000
File Handler	0xC1020000 –

Table 3: Memory Map

4 Co-Processor

The co-processor design implements the bulk of the image processing algorithm. Once the MBLite processor has sent the start flag the co-processor will begin reading from the wishbone buffer the 256 intensity levels of the image. At which point a probability histogram will be created using the number of total pixels in the image and the number of pixels corresponding to that intensity level.

At which point the rest of the algorithm will execute according to the code shown in Figure 1.

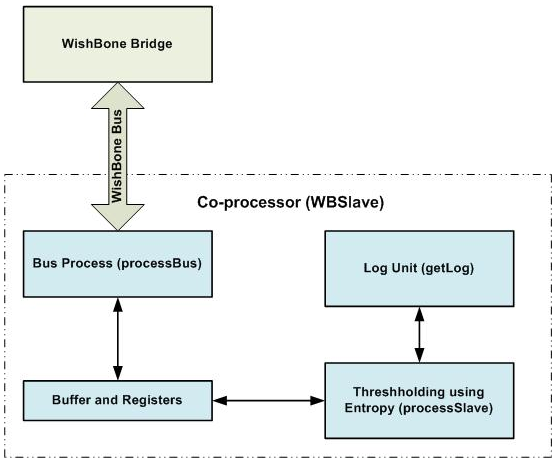

Figure 9: System level view of the co-processor

As Figure 9 demonstrates the Co-Processor is made up of four main components. The processBus which is an independent component that reads the data from the wishbone bus and places the data in the slave buffer. At which point the processSlave reads data from the buffer and the control registers to start execution. While the getLog unit is the independent component responsible for calculating the natural logarithm.

the processSlave and getLog components interact by send and reading flags to determine when the logarithmic unit should start executing and when the execution is finished and the output is ready.

Flags and I/O	**Description**
log_start	Flag for starting the logarithmic unit
log_done	Flag to indicate the output of the logarithmic unit is ready to be read
log_in	Fixed point value for input to the logarithmic unit.
log_out	Fixed point value for the output of the logarithmic unit.

Table 4: Flags and I/O between processSlave and the Logarithmic Unit

As Table 4 shows there are two flags which are used to control the logarithmic unit. And two fixed point variables to pass the input and output between the two processes. Below is a description of the operation of the logarithmic unit.

1. Place a valid input value in fixed point notation to *log_in*
2. Set the *log_start* to 0x1 to start execution
3. Wait for the *log_done* flag is toggled to 0x1
4. Once the *log_done* is set high, read from the *log_out* variable the result of the logarithmic unit.
5. Set *log_start* back to 0x0 to reset the logarithmic unit.

4.1 Logarithmic Unit

The design for the logarithmic unit is one of the essential components of the algorithm. Due to the non linear behaviour the function can be approximated in a number of ways. In this design we have taken a two prone approach.

The first approach uses the Taylor expansion of $ln(x)$ to give an approximation. Where,

$$ln(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}(x-1)^n, \quad 0 \leq x \leq 2 \tag{1}$$

Naturally, expanding the function to infinity is not possible and the required accuracy is not needed to be exact. Therefore, the design decision was taken to limit the expansion to 16 terms. Expanding the natural logarithm 16 times yields results that are quite accurate for values larger than 0.1. However, closer to 0 the inaccuracies increase and do affect the output.

The second part of the solution was to pre-calculate the logarithmic values for inputs less than 0.1. Using a set of 35 pre-calculated logarithmic values where we can estimate use linear interpolation the value of ln(x) for values less than 0.1 that do no directly equal those located in the table. This then requires four tables to store the data. One for the input values, output values, the line gradient between input values and their offset.

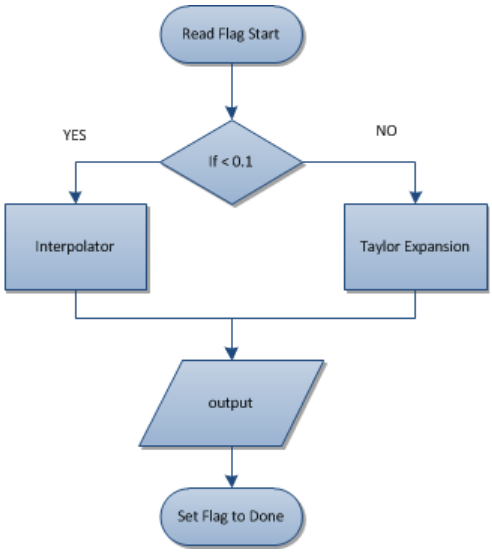

Figure 10: Decision Flow of the Natural Logarithmic Unit

Combining the two methods of calculation we can achieve a logarithmic unit that is both fast and accurate for our image thresholding needs. However, it must be noted that this unit was only designed for input values less than 2. However, as the application analysis shows the values which will be inputted into the unit are on average less than 0.32.

5 Synthesized Results

In order to find performance and area utilization, the design of co-processor is synthesized. The two versions of the co-processor are synthesized to compare the performance. The first design is float version. In this version, no optimization for the float variables is done. The performance and area utilization for float version is give in Tables 5 and 6 respectively.

Float Version Performance Estimates						
Instance	Module	Latency (cycles)		Interval(cycles)		Pipeline
		min	max	min	max	
grp_WBSlave_getLog_fu_216	WBSlave_getLog	?	?	?	?	none
grp_WBSlave_processSlave_fu_280	WBSlave_processSlave	9742	1185288	9742	1185288	none
grp_WBSlave_processBus_fu_336	WBSlave_processBus	6	?	6	?	none

Table 5: Co-processor performance in term of latency and interval for float version

Float Version Utilization Estimates				
Name	BRAM_18K	DSP48A	FF	LUT
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	5	10	4473	7901
Memory	1	-	-	-
Multiplexer	-	-	-	82
Register	-	-	100	-
Total	6	10	4573	7983
Available	268	180	184304	92152
Utilization (%)	2	5	2	8

Table 6: Co-processor area utilization for the float version

In the second version, float variables in the co-processor designs are replaced with the fixed point ones. The performance and area utilization for the fixed point version is given in Tables 7 and 8 respectively.

Fix Point Performance Estimates						
Instance	Module	Latency		Interval		Pipeline
		min	max	min	max	
grp_WBSlave_getLog_fu_216	WBSlave_getLog	?	?	?	?	none
grp_WBSlave_processSlave_fu_280	WBSlave_processSlave	21251	1339363	21251	1339363	none
grp_WBSlave_processBus_fu_336	WBSlave_processBus	6	?	6	?	none

Table 7: Co-processor performance estimates for the fixed point version

Fixed Point Utilization Estimates				
Name	BRAM_18K	DSP48A	FF	LUT
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	2	26	5280	9186
Memory	1	-	-	-
Multiplexer	-	-	-	87
Register	-	-	78	-
Total	3	26	5358	9273
Available	268	180	184304	92152
Utilization (%)	1	14	2	10

Table 8: Co-processor area utilization for the fixed point version

It is clearly evident from the Tables 5 and 7 that float version performs better than fixed point version in terms of latency and time period. This is because in the float version each variable is assigned 32-bit of memory irrespective of the size of the content of that variable. As wishbone bus is also 32-bit,therefore, there is no need for the select signals from the MBL1c processor in order to transfer data on the wishbone bus. On the other hand, in the fixed point version, as there are 21-bits are being used for the float variables. Therefore, to transfer those variables on the bus there is a need of select signals which can handle one HALFWORD(16-bits) and BYTES(8-bit). Due to the requirement of these additional signals performance of the fixed point version is less float than version. In case of area utilization, fixed point version perform slightly better than float version because fixed point uses as much memory as required by that number to fit in. It can seen from the Tables 6 and 8 .

6 Improvements

The implementation of the image processing algorithm given in this project has some room for improvement. As the development of SoCs allows for parallel execution of logical units it is optimum to find areas of parallelism within an algorithm.

However, for this project the parallel components of the algorithm where not researched and so the execution order of the implementation is very much sequential as given in the code. More time could have been spent in looking at the areas of the code which can be executed independently. As the MBLite Processor is waiting on the co-processor and the co-processor on the logarithmic unit when it is executing. Utilizing the full parallel potential of the algorithm can allow for higher performance of the implementation.

Another improvement can be the utilization of the Wishbone Burst read and write protocol. At the moment the simple single read and write to and from the Wishbone Bus is used. With a burst read and write implementation we can transfer data from the MBLite processor to the co-processor much faster. However, due to the fact that only the histogram is being transferred which consists of 256 values of 32bits a burst write mode may not beneficial. As the whole image is not being transferred.

7 Team Work Division

Christopher focused on the algorithm analysis to help determine the areas of the algorithm which consume the most performance and the fixed point width needed to implement the algorithm correctly. In addition, he focused on the implementation of the logarithmic unit ensuring sufficient accuracy was given in order to implement the algorithm correctly. In short the bulk of the algorithm was ported to SystemC by Christopher.

Muneeb focused on setting up the SystemC development environment. Reading through the documentation and experimenting to ensure that the algorithm was correctly implemented in SystemC. In addition, he enabled the SoC to read images into the memory. Fixed many bugs that appeared during the porting process from software to SystemC implementation. Moreover, planned the memory map and register and how the co-processor interfaced with the MBLite processor. Finally, took the finalized design and synthesized the multiple implementation that were created.

8 Conclusion

The design given implements the image processing algorithm, binary thresholding using entropy, in a compact and elegant design. The majority of the algorithm was ported to a co-processor which took as input a histogram of 256 intensity levels read from the wishbone slave to determine an optimum threshold value to convert a gray scale image to binary.

The results where that a fixed point implementation, though thought to be less complex than a floating point precision implementation yield worse performance when compared. However, the only noteworthy gain of the fixed point implementation is the amount of memory used is less.