Specification and Modeling of a Canny Edge Detector for System-on-Chip Design

Yuan-Hung Lin

06/14/2017

EECS222 EMBEDDED SYS MODLNG
Department of Electrical Engineering and Computer Science
University of California, Irvine

Abstract

This report contains case study about an edge detect program and how to modelized the source code into a system step by step. It also includes some analysis about the performance of the system by using SoC environment and how to utilize the evaluation to improve the design of the system. The goal of this project is to build a real-time edge detect system from video camera.

1. Instruction

Since that Systems are becoming more complex people need a higher level of abstraction language. That why we need to learn specC and systemC. They contain structure hierarchy, concurrency, synchronization, timing and composite data type. The System is built with computation and communication. We want to replace the function into blocks and channel and then connected blocks with channels. The advantage of modulization is that when some blocks need to be modified, user can modify the blocks without affecting other function. The system is more flexible.

2. Case Study

2.1 Obtaining and studying the Canny application

We start from downloading the source code which is written in c. The source code is runnable and it can generate the edge of the golf cart. We read the source code to see what functions are in the code and what is their purpose.

In class, there is a discussion about the array that store the picture. It is better to use 1D array or 2D array. I think 1D is easier to code and it will be faster than 2D.

After 2 weeks of practice, we have some experience about SpecC and SystemC. Now it is time to get back to modeling of Canny. The first task is to fix bugs in the source code. The program omits a column of pixel in the right and a row of pixel at the bottom. To be honest, I might not notice this mistake if the professor did not tell me. I make a mistake in this step. I compare the picture which download from the server with the picture on the slice of lecture3. However, they are not the same.


2.2 Creating a simulatable model in SpecC/SystemC SLDL

After that, we need to make the code warning-free. As the professor said, you don't want to use a code with any warning. It will be hard to debug and test. Most of the warning is that some variables are not used. It is easy to eliminate those variable because compiler indicates the lines of the code.

Now, we need to hard code the parameter which fixed constants in the system. They still can be modified in the software. That is one of the advantages of embedded system. We know the purpose of the chip so we can customize the system. This can save resource or increase the efficiency. In this program, we know the size of the input. That is to say, we can fix the size of the array which stores the picture. No need to reallocate memory for the input. We clean out all of the memory allocation and memory free.

2.3.1 Creating structural hierarchy with test bench

Now, we have a program without bugs and the parameter is also fixed. Next step, we are going to modelized the code, turn it into a top-level structure hierocracy with a test bench. First, let's talk about the test bench. In the previous homework, we use a single image as input. We now want to convert the application into real-time video handling.

Here comes the thing, the overall structure hierarchy. The outer block which is the Main behavior contains three blocks: stimulus, platform and monitor. They are connected by two channels. (Fig. 1)

Stimulus read the image from input and sent it to the platform through channel_queue. Platform receiver the image and process the image. Inside platform, there three blocks: DataIn, DUT and DataOut. When dataIn receives image, it will send it to DUT through a channel. DataIn has an infinity loop, it will keep checking if there is any input image.

Dut is the most important block in this system. Most of the computation happens in this block. DUT contain a function call Canny. This is the source code that we studied before. We copy the source code and paste it in this block. After the process, DUT will send the result image to DataOut. DataOut is also an infinity loop. It will send the image to monitor when it receives edge image from DUT. The last block is monitor, its job is to output the edge image as a file.

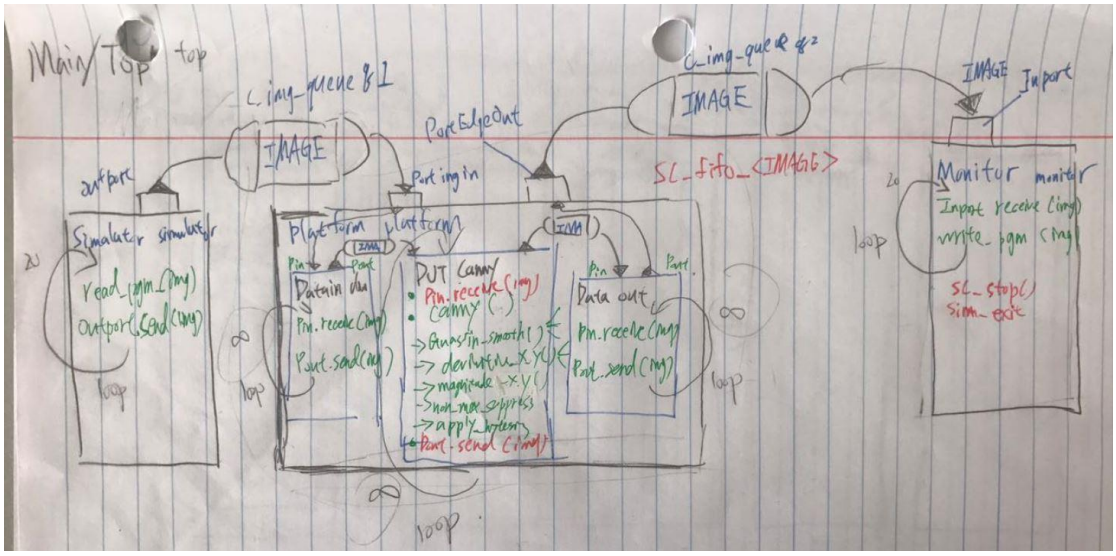


Fig. 1

I get a deduction in this homework for using global value. The reason is that I do not know change the input value during the program, so I store them in a global array like this:

```
string infilename[20]={"video/EngPlaza001.pgm","video/EngPlaza002.pgm","…….
string outfilename[20]={"video/EngPlaza001.pgm","video/EngPlaza002.pgm","…….
```

Professor use an amazing way to deal with this problem:

```
#define IMG_IN       "video/" VIDEONAME "%03d.pgm"
sprintf(infilename, IMG_IN, n+1);
```

To me, this is something new. It solves the problem that the name of the input is changing during the process.

Another problem is stack overflow:

Program received signal SIGSEGV, Segmentation fault. 0x0000000000403a42 in sc_main (argc=Cannot access memory at address 0x7fffff04f27c

This is because the default stack size is 1024k bytes. When the platform's constructor instantiates three instances, they will share the same stack.

```
SC_CTOR(Platform):din("din"),dut("dut"),dout("dout")
{
…..
}
```

However, every model is allocated with a large stack size since canny is an image processing program. It will lead to the problem of stack overflow.

```
set_stack_size(128*1024*1024)
```

There are two solutions:

1. Increase the stack size by command
   limit stacksize 128 megabytes
2. Use pointer.
   Instantiate a pointer instead of an object, use this pointer to point the object that is outside the model.

2.3.2

We have a big picture for the whole system, now we can try to refine the structure of DUT. Most of the function are in DUT, so we can add an additional level in this block. There are five functions in this block: gaussian_smooth, derivative_x_y, magnitude_x_y, non_max_supp and apply_hysteresis. It is similar to the previous homework. (see Fig. 2)

Make those function into behaviors. Create some channels and connect the blocks with correct ports. There are two kinds of channels here. One is IMAGE, it is for unsigned char. Another one is SIMAGE which is for short int. Connect to the wrong type of channel will lead to serious problem. The program will produce a blank image. Channel in SystemC is sc_fifo, that is pipeline channel. However, I use specC this time. Channel in specC is sequential.

The task is not over yet. We still need to add another level in gaussian_smooth.

There are 2 functions and two for loops in gaussian_smooth: Receive_Image, Gaussian_Kernel, blurX and blurY. This job is trickier because the connection between functions is variable rather than channel.

For example, make_gaussian_kernel is a void function, it does not have a return value. We must read the code carefully and find the variable. Those variables will become output of the function. We also need to wrap up the two for loop and turn the two loops into two behaviors. Blur_x receives the data from Gaussian_Kernel which are image, kernel and center. Blur_y receives the result from blur_x and receive kernel and center from previous function. Finally, blur_y will send the result to the output port.

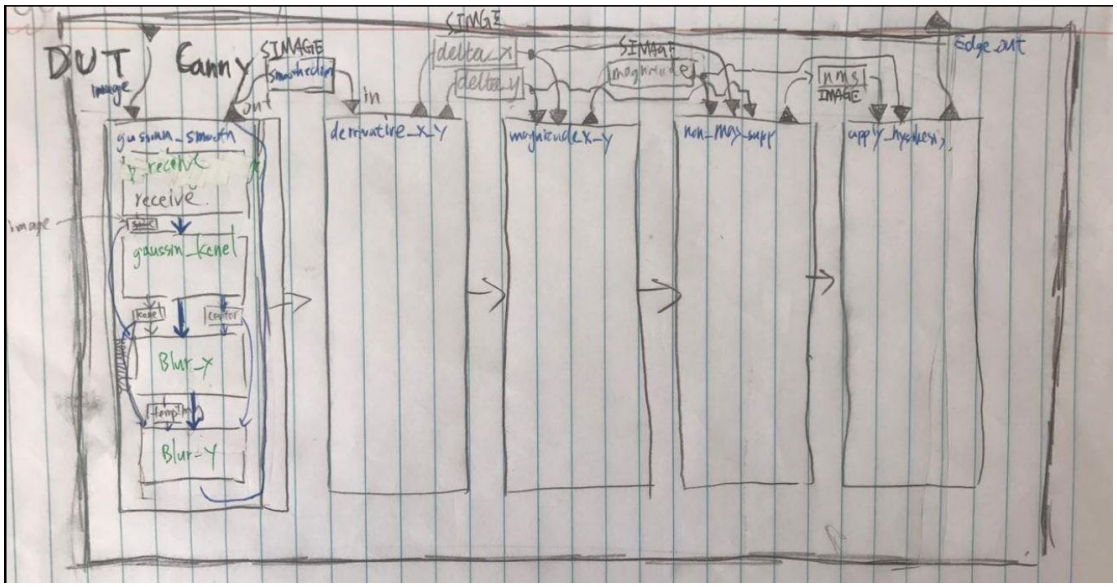


Fig. 2

After we finish the hierarchy structure in gaussian_smooth, now we can use the profiler integrated into the System-on-Chip Environment. It is a convenient GUI that can help user to profile the system. In addition, it can show the result with chart.

Here is the result I get:

Gaussian_Smooth ... 1.5231 56%

------- Gaussian_Kernel ...0%

|------ BlurX ...28%

\------ BlurY ...28%

Derivative_X_Y .. 0.2307 8.47%

Magnitude_X_Y ... 0.2 7.34%

Non_Max_Supp ...0.4616 16.95%

Apply_Hysteresis ... 0.3077 11.3%

Now, we know Gaussian_Smooth is the bottleneck of the DUT.

2.4 Pipelining and parallelization

To increase the performance, there are several methods we could try:

1. Add floating point unit
2. Accelerate the clock
3. Parallelize some blocks
4. Pipeline

We decide to pipeline and parallelize the component in DUT to improve the performance. To begin with, we measure the time for processing a frame. Set two timer in stimulus and monitor. Stimulus will record the time when it receives the input, then it will send this time to monitor. Monitor will record the time when it receives the image. Then monitor will subtract its time with stimulus' time. This is the processing time for each frame.

However, frame per second is what we are interested in, so we need to divide 1 by the frame processing time.

Next step, back-annotate the delay into the model according to the simulation in SoC. Use waitfor to add the delay in each component of DUT, so that we can get the simulation time. Now the system has its simulation time, we can start to improve the performance of the system and reduce the simulation time.

Our strategy is pipeline the DUT into several stages. Since that specC has keyword pipe, it is easy to pipeline the DUT. Pipe is similar to for, we can define how many cycles are there in the pipeline. Then add stages into pipe's body.

We also need to add piped to the port-mapped. How many stages the variable cross, how many piped is needed for that variable.

Now, we are going to decompose the first stage of the pipeline in order to further improve the performance. Why do we need to do that? Well, let's look at the chart of the computation time for each component. (see Fig. 3) Gaussian_smooth dominates the pipeline significantly. That is to say, we should divide it into more stage to improve the performance of the pipeline.

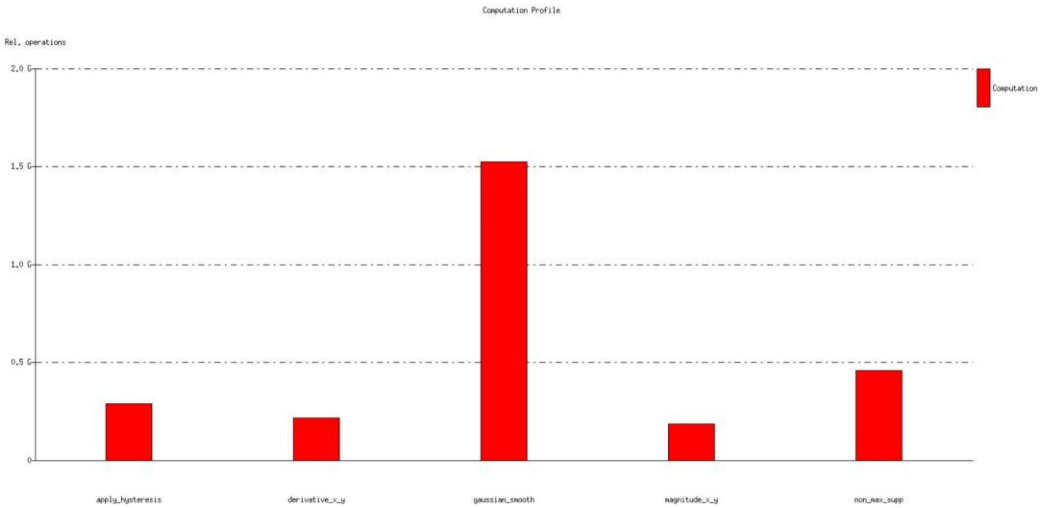


Fig.3

We take out blur_x and blur_y from gaussian_smooth. Add them to the same level as gaussian_smooth. Here is the instance tree:

DUT
|------ Gaussian_Smooth gaussian_smooth
| |------ Receive_Image receive
| \------ Gaussian_Kernel gauss
|------ BlurX blurX
|------ BlurY blurY
|------ Derivative_X_Y derivative_x_y
|------ Magnitude_X_Y magnitude_x_y
|------ Non_Max_Supp non_max_supp
\------ Apply_Hysteresis apply_hysteresis

It takes me a while to do this step because there are a lot of port need to be mapped or reconnect. (see Fig. 4) It is more efficient if you draw a picture before coding. I organize the connection between different components but I did not draw the pipeline.

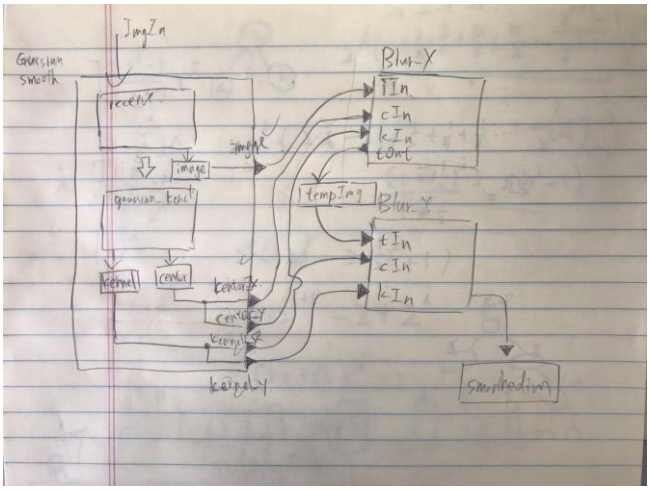


Fig. 4

Last, slice the Blur_x and Blur_y into 8 pieces. Each piece is parallel to each other. Fortunately, I also enroll EECS221 high performance computing in this quarter. I learn how to divide a task into several small tasks and distribute to the threads. After parallel blur_x and blur_y, the through put of the pipeline increase. Here is the result.

	Frame Delay	Total simulation time
Canny_step1	0 us	0 us
Canny_step2	39285000 us	130950000 us
Canny_step3	25626000 us	89882500 us
Canny_step4	13122000 us	50286500 us
Canny_step5	8160000 us	28663125 us

In this homework, I met some problems.

1. The channel between stimulus and monitor should not be less then 5ul, otherwise, deadlock happen. I guess it is because there are seven stages in the pipeline so the buffer of the channel should not be too small.
2. When parallelize blur_x and blur_y, I am afraid that it will have race condition if every slice write to a same array. However, it is ok to do that because each piece writes different parts of the array.
   I try to use memcpy at the end of the main function, copy the data sequentially. However, it is hard to do that in blur_y because the data blur_y return is columns rather than rows.

2.5 Performance estimation and throughput optimization

Finally, here we come to the last step. Let's refine the model with more realistic assumption. To do so, we need to allocate processor element for each pipeline stage. After that, we use the SCE profiler to estimate and evaluate the performance. Here is the estimation result in a bar chart. (Fig.5) The load is not balanced. Non_max_supp's computation time dominate the pipeline. Now we need to reduce its computation time to improve the throughput of the pipeline.

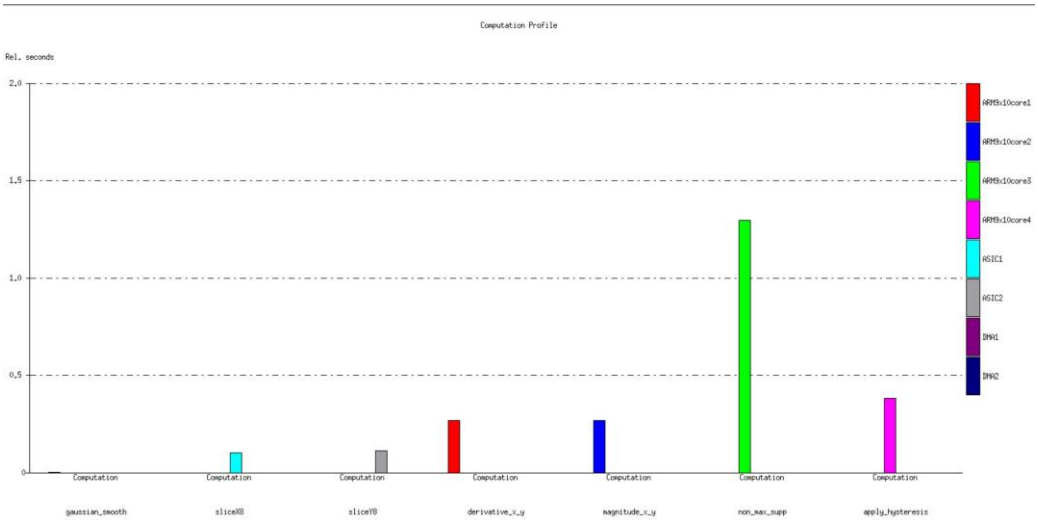


Fig. 5

To trace the highest bar to get a pie chart of the type of the operations performed in non_max_supp, we double click on it. We found the ration of the integer and floating point. (see Fig. 6)

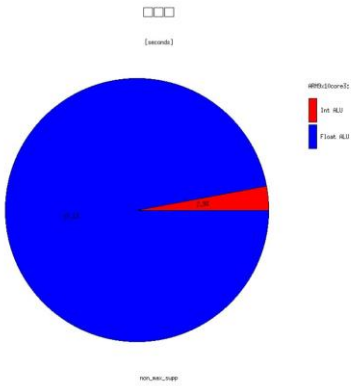


Fig. 6

Since that blur slice blocks are mapped to hardware PEs, their Stage delay will be different. We record the delay and back-annotate them into the model to see the difference. However, the result does not change. The reason is that non_max_supp has dominated the pipeline. Although we improve the blur_x and blur_y by parallelization, they still need to wait for non_max_supp.

In order to improve the throughput of our pipeline in the DUT, we need to deal with the non_max_supp block because it has the longest stage delay. We also know that it contains many floating-point computations which is time-consuming.

Floating-point takes most of the computation time. We know that floating point computation takes more time than integer computation. What we are going to do is to replace the floating-point arithmetic with fixed-point calculations. Use bit manipulation to shift the value and then do the division. It might lose some accuracy but the speed can be improved significantly.

Now, let's see how many accuracies do we sacrifice. Use ImageDiff tool to compare the different pixels between two frames. I use frame002 and the result shows that there are only 3 pixels that are different. I check the output image and zoom in. I did find the three dots in the output. The result shows that the difference is too small that can be ignored.

Last, back annotate the new time for non_max_supp and use SCE to evaluate again. This time, we can see that the job is more balanced. (see Fig. 7)

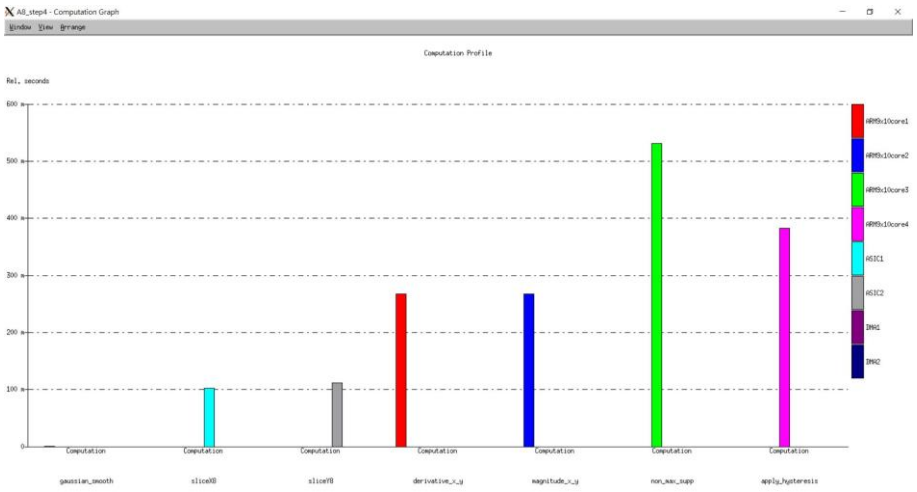


Fig. 7

The new ration between integer and floating-point is in Fig. 8. Basically, there is no floating-point in NMS block.

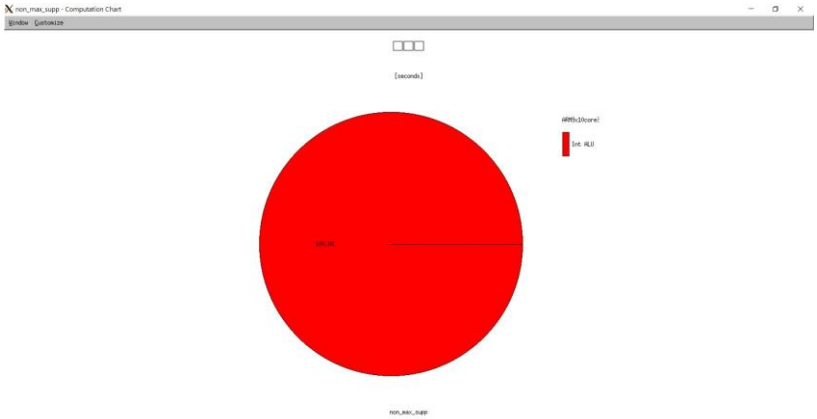
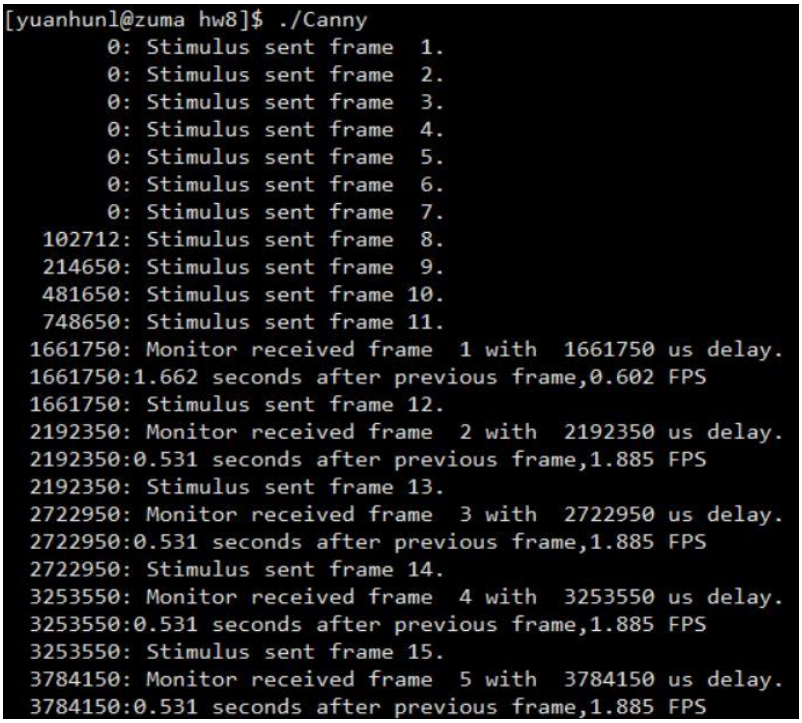


Fig. 8

The final result is in Fig. 9



FPS is 1.885. This does not match the goal of 20. Since that throughput of video camera needs 20 frames per second, 1.885 is not closed to our goal. There are some additional modifications that can further improve the performance:

1. Divide block into more stages. Non_max_supp and Apply_Hysteresis can be divide into more stages since that they dominate the pipeline.
2. Replace the processor with stronger one.
3. The resolution of the image in our program is high. Maybe we could decrease the resolution rate to a certain point that could be accepted by the end user.

3. Summary and Conclusion

In this course, I learn the idea of system level describe language. I try both Spec C and systemC. Their syntax is slightly different so it is not difficult to try both of them. However, when it comes to evaluation, SCE is a convenient GUI for user. It can show the computation time, communication time and other time with charts. The user can also track the specific section by double click the bar. Besides, user can also assign processor elements to the system for simulation. I think it is pity that Spec C cannot become the standard tool in industry.

From the beginning of the course, we start from reading the source code. Then build the structure hierarchy and modelized the program. Use sce tool to evaluate the performance and modify the structure according to the evaluation. We learn how to find the bottleneck and how to deal with those problems.

Last, we get the final result. The source code becomes a system that can build on chips.

I think the most interesting part of the embedded system is that it contains both hardware and software. Engineering of embedded system need to use different aspects to look at the product. It is also an important bridge between hardware and software.

Last, I learn many skills about Linux because we need to run the code on the EECS server. This is an important skill to a CPE student. When I was a child, I always think the hacker in the movie is very cool because they can operate the computer with keyboard. I might not have the chance to learn how to be a hacker, but now I know how to use the Linux OS. Discard the GUI, the performance of Linux is much faster than windows.

I will study some material about Linux as the future work. There are many skills which I need to learn.

4. Reference

Canny Edge Detector Comparison Heath, M., Sarkar, S., Sanocki, T., and Bowyer, K. Computer Vision and Pattern Recognition '96, San Francisco, June 1996.

EECS222 EMBEDDED SYS MODLNG message board