

CMPE-660 Laboratory Exercise 5
MicroBlaze & Embedded System Design

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

Chris Abajian
Performed 11/09/2020
Submitted 11/23/2020

Section 01S2
Professor: Marcin Lukowiak
TA: Connor Henley
Joshua Willson

Design Methodology

This laboratory exercise investigated embedded system design using the MicroBlaze soft microprocessor core. Rather than writing conventional HDL, Xilinx's MicroBlaze platform allows embedded projects written in C/C++ to communicate and execute on an FPGA.

The first part of this exercise was to configure MicroBlaze for the Nexys4 DDR FPGA to create blank projects. Figure 1 shows a block diagram of the final MicroBlaze design used in this exercise.

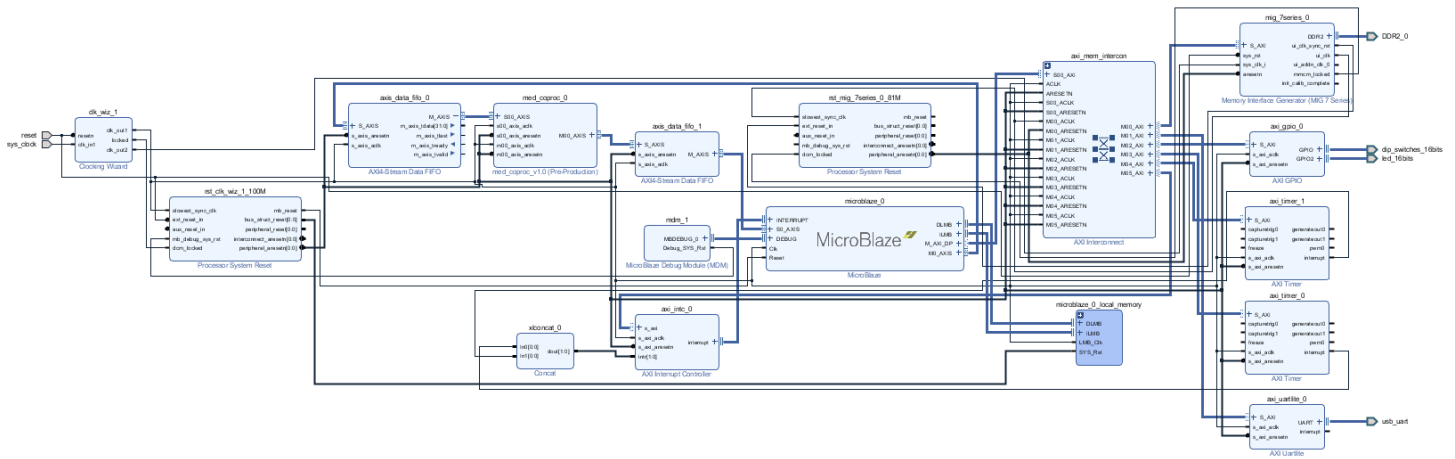


Figure 1: Final MicroBlaze design block diagram.

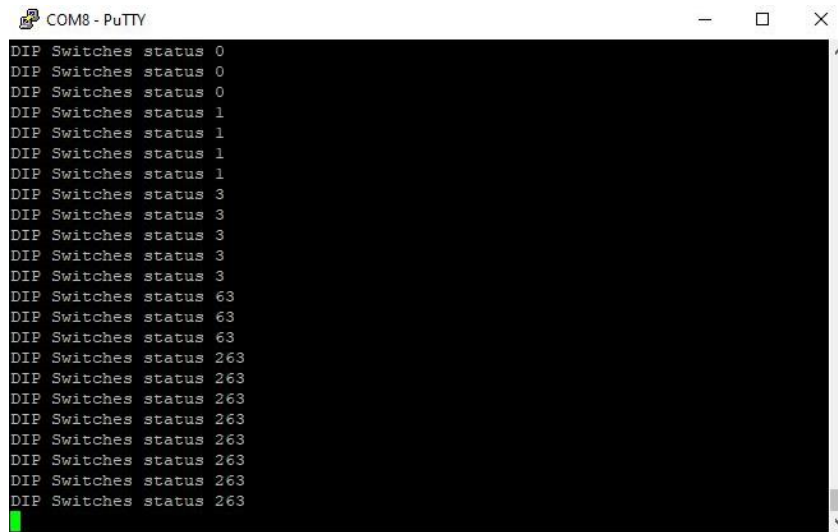
A few sample programs were created to interface with the FPGA's GPIO including onboard DIP switches and LED's.

The second part of this exercise interfaced with hardware timers and interrupts. These timers can be used to report accurate timings for hardware processes. A software median filter was then implemented where a distorted image could be sent over UART, filtered, and sent back. Timing analysis was added to report the hardware processing time.

Finally, the third exercise implemented a hardware implementation of the median filter using a custom IP coprocessor in Vivado. This block was connected to the MicroBlaze via the AXI Stream protocol. Like previous exercises, two data FIFO's were connected on either end of the coprocessor to prevent data loss.

Results and Analysis

The custom program developed to read onboard DIP switches and write this value to the LED's was tested on the FPGA. Figure 2 shows a terminal capture of the switch values read from the FPGA.

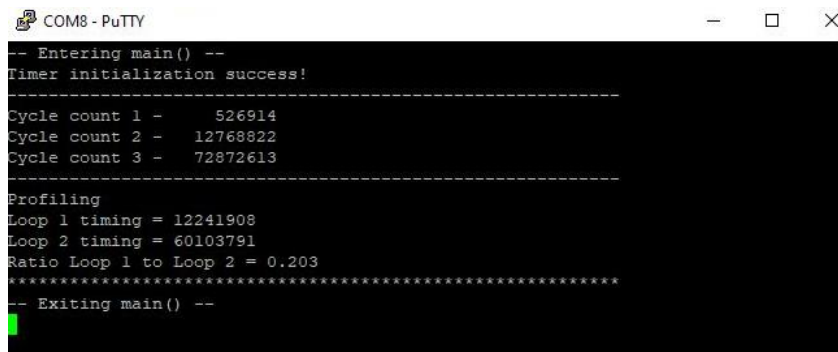
A screenshot of a PuTTY terminal window titled 'COM8 - PuTTY'. The terminal displays a series of 20 lines of text, each reading 'DIP Switches status' followed by a numerical value. The values are: 0, 0, 0, 1, 1, 1, 1, 3, 3, 3, 3, 63, 63, 63, 263, 263, 263, 263, 263, 263, and 263. A green cursor is visible at the end of the last line.

```
COM8 - PuTTY
DIP Switches status 0
DIP Switches status 0
DIP Switches status 0
DIP Switches status 1
DIP Switches status 1
DIP Switches status 1
DIP Switches status 1
DIP Switches status 3
DIP Switches status 3
DIP Switches status 3
DIP Switches status 3
DIP Switches status 63
DIP Switches status 63
DIP Switches status 63
DIP Switches status 263
DIP Switches status 263
DIP Switches status 263
DIP Switches status 263
DIP Switches status 263
DIP Switches status 263
DIP Switches status 263
```

Figure 2: Terminal capture of DIP switch readings from the FPGA using MicroBlaze.

This program was found to correctly function.

Timers were investigated and verified by running a simple program that displays the number of clock cycles between two sections of code. Figure 3 shows the terminal capture of this result.

A screenshot of a PuTTY terminal window titled 'COM8 - PuTTY'. The terminal displays the output of a program. It starts with '-- Entering main() --', followed by 'Timer initialization success!'. Then, three lines show cycle counts: 'Cycle count 1 - 526914', 'Cycle count 2 - 12768822', and 'Cycle count 3 - 72872613'. These are separated by a dashed line. Next, a 'Profiling' section shows 'Loop 1 timing = 12241908', 'Loop 2 timing = 60103791', and 'Ratio Loop 1 to Loop 2 = 0.203'. This is followed by another dashed line and then '-- Exiting main() --'. A green cursor is visible at the end of the last line.

```
COM8 - PuTTY
-- Entering main() --
Timer initialization success!
-----
Cycle count 1 - 526914
Cycle count 2 - 12768822
Cycle count 3 - 72872613
-----
Profiling
Loop 1 timing = 12241908
Loop 2 timing = 60103791
Ratio Loop 1 to Loop 2 = 0.203
-----
-- Exiting main() --
```

Figure 3: Terminal capture of TimersTest program.

From Figure 3, the ratio between Loop 1 and Loop 2 was approximately 0.2. This result aligns with the theoretical ratio of 0.2 as Loop 2 follows a loop consisting of five times the iterations of Loop 1 (5,000,000 versus 1,000,000, respectively). A second program to test timer interrupts was tested. The results are captured in Figure 4.

```

COM8 - PuTTY
Going back to main 0 ()

Interrupt 1 - Count1 = 3
Going back to main 0 ().....

Interrupt 0 - Count0 = 13
Going back to main 0 ().....

Interrupt 0 - Count0 = 14
Going back to main 0 ().....

Interrupt 0 - Count0 = 15
Going back to main 0 ().....

Interrupt 0 - Count0 = 16
Going back to main 0 ()

Interrupt 1 - Count1 = 4
Going back to main 0 ().....

```

Figure 4: Terminal capture of InterruptsTest program.

Interrupt 0 was configured to generate interrupts four times as often as Interrupt 1. This result is verified in Figure 4. Timing analysis was then incorporated into the median filter program to time the duration of each read, processing, and write operation. The resulting filtered image is shown in Figure 5.

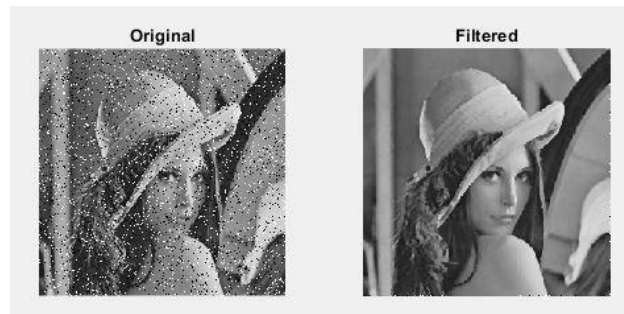


Figure 5: Filtered Lena image using MicroBlaze software algorithm.

Table 1 shows the timing results of these operations.

Table 1: Software Median Filter Implementation Timing Results

Operation	Time
Read Image	5.68 sec
Process Image	2.84 sec
Write Image	5.66 sec

Finally, the third part of this exercise incorporated the filtering program with a hardware coprocessor. This coprocessor handled the median filter operation and was tested using the same image. The final filtered image appeared identical to that shown in Figure 5. Table 2 shows the timing results of the hardware median filter implementation.

Table 2: Hardware Median Filter Implementation Timing Results

Operation	Time
Read Image	5.70 sec
Process Image	0.20 sec
Write Image	5.66 sec

From Table 2, it is observed that the read and write operations are nearly identical between the two implementations since there was no change in implementation for these processes. However, the median filter processing was more than ten times faster on hardware using a coprocessor custom IP than using a software implementation.

Conclusion

This experiment successfully introduced the MicroBlaze soft microprocessor and using it to easily interface with the Nexys4 DDR FPGA. Simple programs verified the configuration of the MicroBlaze environment and showed successful interaction between onboard GPIO and timers. Timing analysis was performed on both software and hardware implemented median filters and showed a significant timing improvement on the hardware enabled approach. The experiment was a success.