Participants:

Jacob Munkholm Hansen, 20140479

Jens Jakob Mikkelsen, 201506215

Yang Hanshou, 201902791

**Embedded Real Time Systems – Assignment 2**

For the exercises, the following design will be used:

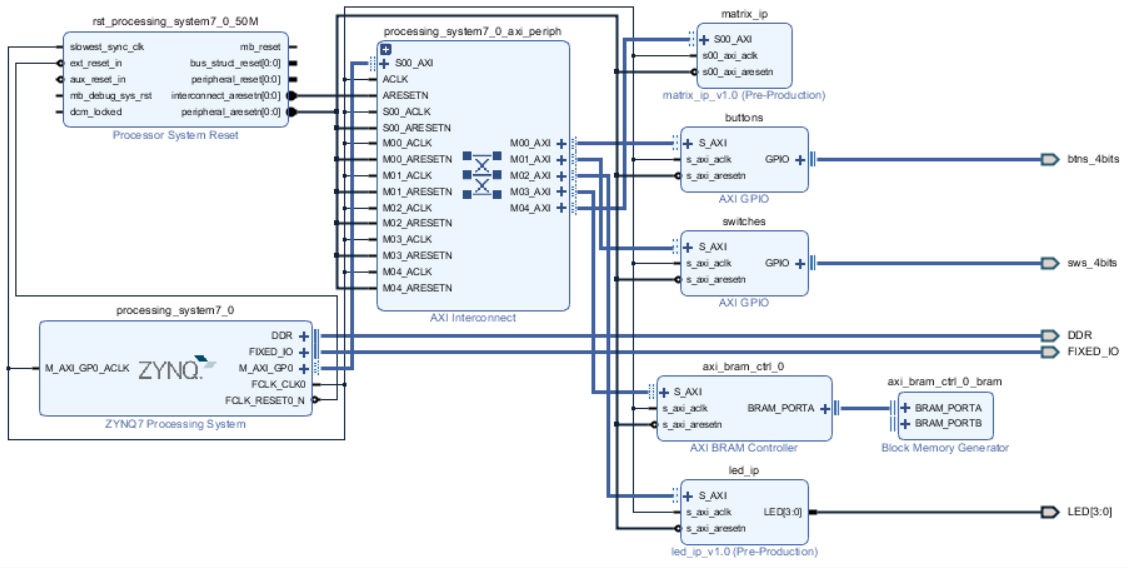


Figure 1 - System building blocks

The buttons and switches will be used in exercise 3 and 4, and the matrix\_ip will be used in exercise 5. The BRAM and led\_ip persists in the design created in the pre-exercises.

3) Create a Console Application

The main-program looks like this:

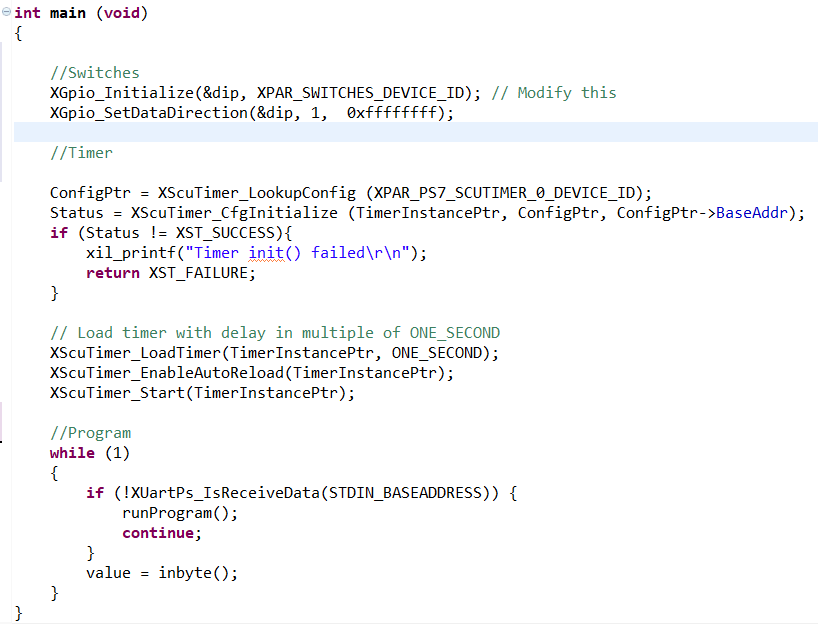


Figure 2 - main program

The main function can be split into two parts; initialization of hardware components (Led’s and timer) and an always-running loop. Every iteration of the loop checks whether data has been received from the uart. If a new value is received, the ‘value’ will be set to it, allowing for the user to switch between the different parts of the program. This functionality is created in the runProgram() function:

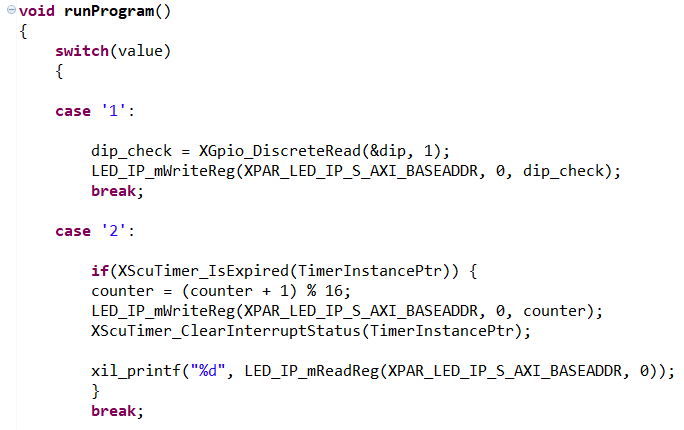


Figure 3 - runProgram() function

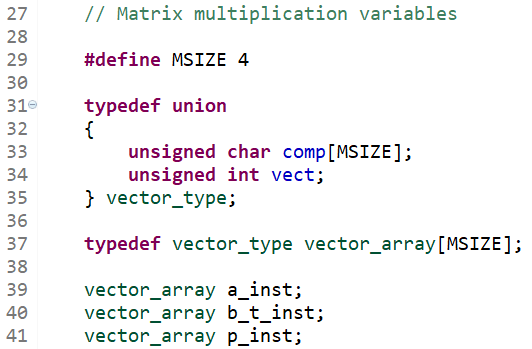
So every time the user writes from the UART, the program will check whether the value corresponds to the case values, and switch between them.

If the value is 1, the switches will be read, and the LED’s will be written to based on what switches are turned on/off.

If the value is 2, every time the timer expires the counter will increase up to 15 and then reset. The LED registers will then be written to, based on the value of the counter. As seen in figure 1, the timer is initialized with a value of one second (325000000 cycles), so it counts every second.

# 4) Create a Matrix Multiplication

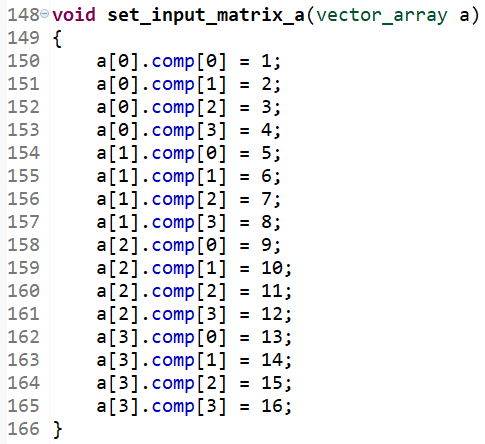
The matrix variables are defined and declared as seen below. The row/column size is restricted to be 4, and the matrices a\_inst, b\_t\_inst, and p\_inst are thus defined two-dimentional arrays of size 4x4:



Figur 3 - Matrix multiplication variables

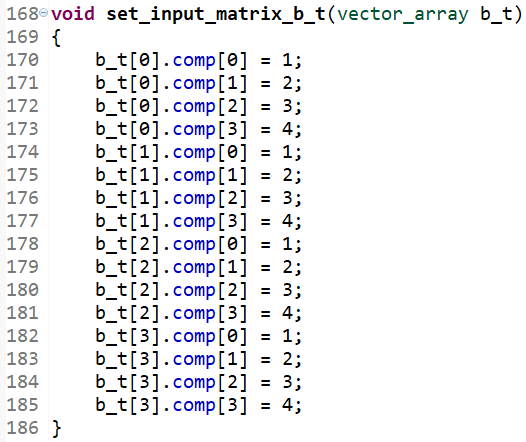
“set\_input\_matrices(…)” has been split into two separate functions “set\_input\_matrix\_a(…)” and “set\_input\_matrix\_b\_t(…)”.

Matrix a is set so that it reflects the matrix in the guidelines:



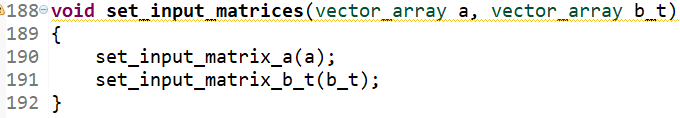
Figur 4 - set\_input\_matrix\_a(...) function

The same is done for matrix b\_t. Notice, however, that since b\_t is b transposed, the rows from the matrix in the guidelines corresponds to the columns of b\_t, so that all values of comp[0] are set to 1, comp[1] to 2, etc.:



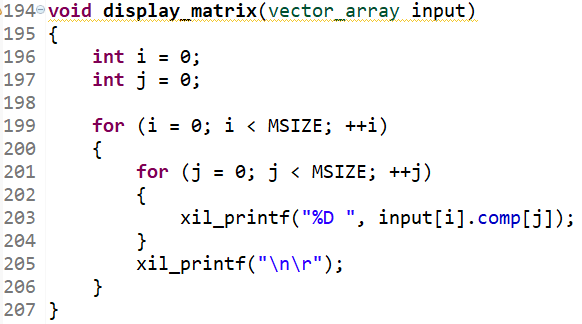
Figur 5 - set\_input\_matrix\_b\_t(...) function

The two functions are then called inside “set\_input\_matrices(…)”:



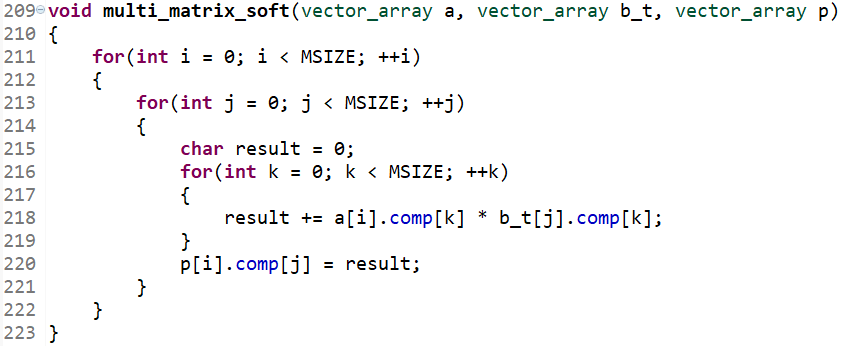
Figur 6 - set\_input\_matrices(...) function

To display a matrix, the function “display\_matrix(…)” has been implemented to loop through each row “i” and each row “j” and printing the value of each row-column element to the UART by calling “xil\_printf(…)”:



Figur 7 - display\_matrix(...) function

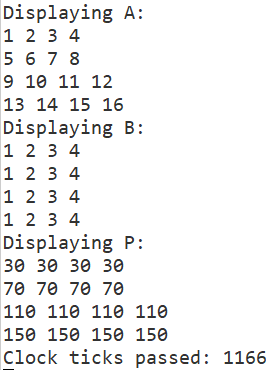
“multi\_matrix\_soft(…)” is implemented with two nested for-loops. “i” is the row number of matrices a and p; “j” is the row number of matrix b\_t and the column number of matrix p; “k” is the column number of matrices a and b\_t. The elements of matrix p are calculated by accumulating the sums of the row elements of matrices a and b\_t. This is because b\_t is matrix b transposed, otherwise it would be the column elements of matrix b that would be incorporated as a factor.



Figur 8 - multi\_matrix\_soft(...) function

The result of setting and displaying the matrices a and b as well as calculating the matrix b can be seen below. The clock ticks of calculating p are likewise displayed.

Please note that “B” below should be “B transposed” or something equivalent.

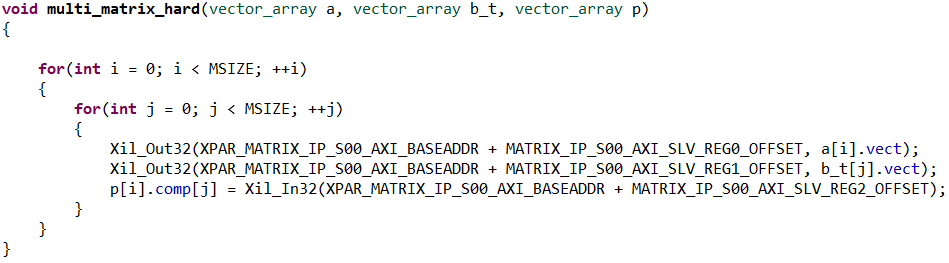


Figur 9 - UART output

# 5) Create a Hardware IP Acceleration

As seen in figure 1, the matrix\_ip is added to the system, allowing the use of the different IP core. As mentioned in the assignment paper, the PL is clocking at 100 MHz, while the PS runs at 650 MHz, so it is expected to be slower.

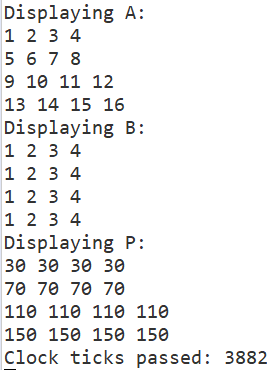
The code implementing the multiplication looks like this:



Figur 10 - multi\_matrix\_hard(...) function

Whereas before each element in the matrix was read and used, now the new IP core allows a simpler implementation that just picks out each column for each row (first two lines in inner loop) and stores the result in the matrix p. The same functions as before is used to showing it.

The following is the result in the terminal:



Figur 11 - UART output

So the same result as before, except now it is 3882/1166 = ~ 3.3 times slower.

# 7) HLS exercise with SystemC

# According to the requirement, the module should include 4 inputs(clk, reset, ctrl, inSwitch) and 1 output (OutLeds), and two threads, one to handle the logic and the other should be able to generate 1 pulse every second, hence the header of the module should be:

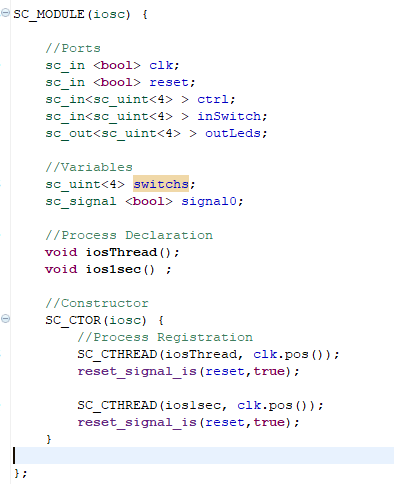


Figure 12 - header file

In the thread handling the logic, the requirement will be met:

When the ctrl value equals to 0x0, the Outleds increments 1 every second, and once the inSwitch is set to 0x0, the outleds should be clear; when the ctrl value equals to other value except 0x0, then the outleds = ctrl and inSwitch. (**#pragma** HLS resource core=AXI4LiteS metadata="-bus\_bundle slv0" variable=ctrl makes the ctrl port and AXI4Lite interface connect with each other)

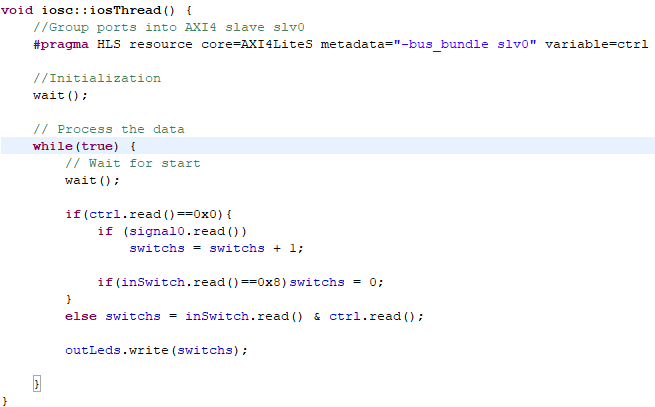


Figure 13 - Implement of the thread handling the logic

In the thread to generate pulse, we have to set a counter to make a signal every 100000000 clock pulses from the board whose frequency is 100Mhz, and sent the signal to the thread handling logic.

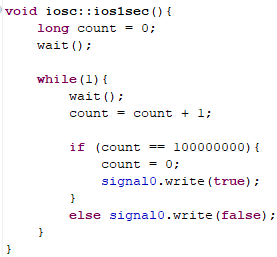


Figure 14 - Implement of the thread generating the pulse every second

The main job of the testbench is to test the logic of the module. Since there are two kinds of output depending on the value of ctrl(ctrl = 0x0 or ctrl = other value), it is necessary to make two tests.

It is necessary to make a new module to get the signal from module mentioned above. The code of the new module is based on the code presented in the BlackBoard as an example, hence here will be only some snippet of the testbench.

Set the ctrl and switch value in the new module:

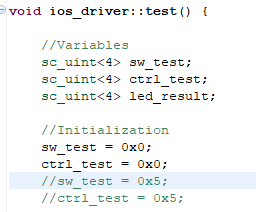


Figure 15 - Snippet of the testbench driver code

To test the logic when value of ctrl equals to 0x0, it is necessary to back to the module to change the thread generating the pulse, otherwise the .vcd file would be very large. In this case, it would generate pulse every 20 ns.

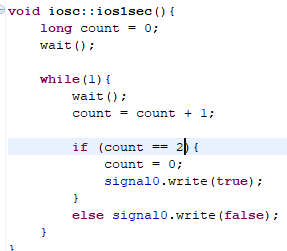


Figure 16 - Snippet of the thread generating the pulse every second

Then, in testbench, make the signals and modules

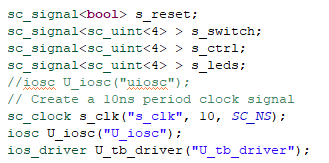


Figure 17 - Snippet A of the testbench

Then connect the ports

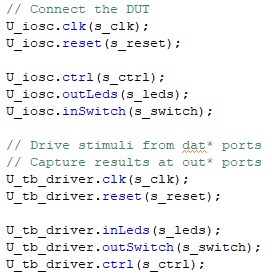


Figure 18 - Snippet B of the testbench

Then simulate 200ns, and generate the .vcd file. In the .vcd file, it shows that the value of leds inrements every 20 ns(every two pulses), which is exactly what the logic describes.

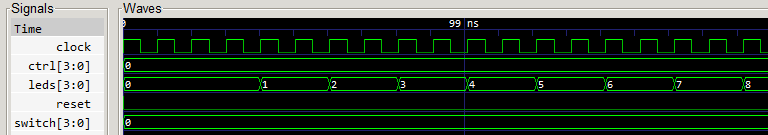


Figure 19 - .Vcd file of ctrl = 0x0

To test the logic when value of ctrl does not equal to 0x0, it is necessary to make a comparison:

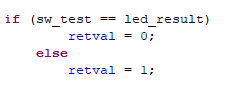


Figure 20 - Snippet logic of the test driver

And only if when the value of inSwitch equals to the vlaue of Outleds, then the test passes.

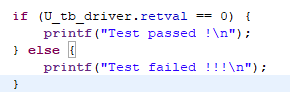


Figure 21 - Snippet C of the testbench

In this case, the initialization should be

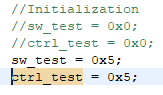


Figure 22 - Change the value to make another test

And then the .vcd file shows the expected result

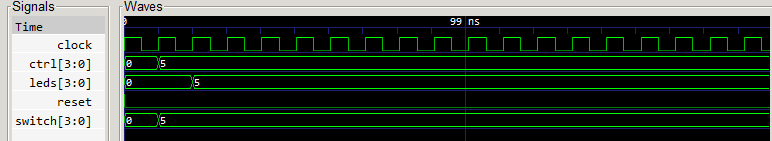


Figure 23 - .vcd file of ctrl = 0x5, inSwitch = 0x5

Since 0101 and 0101 = 0101, hence the test passes. If setting the value of inSwitch to 0xa, then the .vcd file should be:

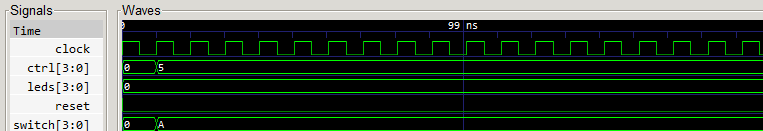


Figure 24 - .vcd file of ctrl = 0x5. inSwitch = 0xA

Since 0101 and 1010 = 0000, hence the value of sw\_test does not equal to the value of the led\_result, the test fails.

After connecting all the modules, the design block looks exactly like the one presented in the requirement.

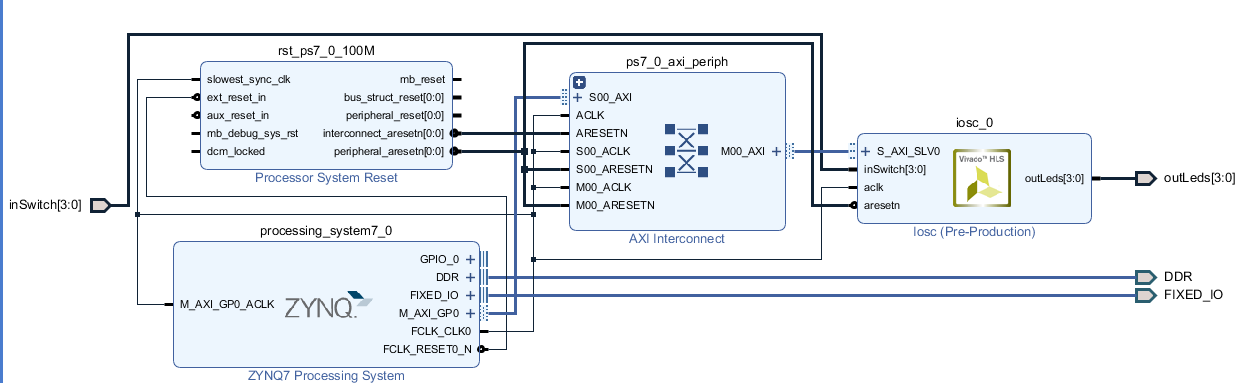


Figure 25 - Design block of the project

After exporting the hardware to the SDK, it is necessary to initialize the driver and set the value of ctrl:

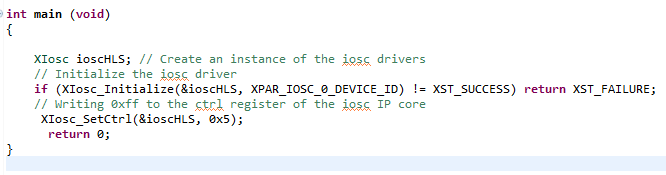


Figure 26 - Initialization of the board

Running the code on the board, when the value equals to 0x5, and value of switches equal to 0xD, the OutLeds equals to 0x5(the second and the fourth lights are on, which means 0101), since 0101 and 1101 equals to 0101.



Figure 27 - Result A of the board

Another successful test with expected result, 0101 and 1001 = 0001.



Figure 28 - Result B of the board

To test another logic, set the ctrl to 0x0, and then run the code on the board. The Outleds would increment every second, and the performance of the board looks like:







Figure 29, 30,31 - Result C, D, E of the board

When the Switch is 1000, all of the leds are off.