Final project

FPGA based Digital Design

Omri Raz – 318671120

Eyal Rothschild – 318793882

Introduction:

Aim of the Assignment:

• Design, synthesis, and analysis of a simple (single cycle architecture) MIPS CPU core with Memory Mapped I/O, interrupt capability, basic timer and FIR filter.

• Understanding of CPU vs. MCU concepts, and FPGA embedded memory structures.

The system:

In this project we were asked to build MCU that contains a Single Cycle MIPS with data memory DTCM and program memory ITCM for hosting the program data and code segments. The MIPS will handle interrupts.
In addition the MCU contains peripheral components such as: switches , LEDS , FIR filter etc.

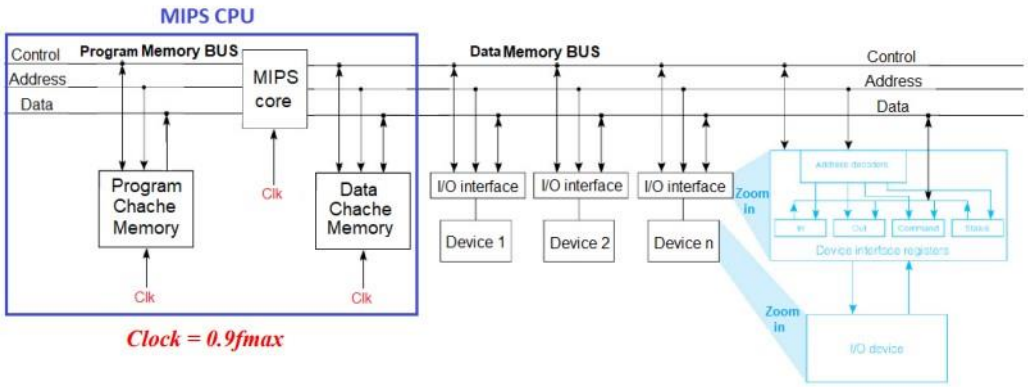


Figure 1 – MCU architecture

GPIO:

• The GPIO (General Purpose I/O) is a simple decoder with buffer registers mapped to data address (Higher than data memory) as given in the assembly code examples that enables the CPU to output data to GPIO devices as LEDs and 7-Segment and to read the Switches array value.

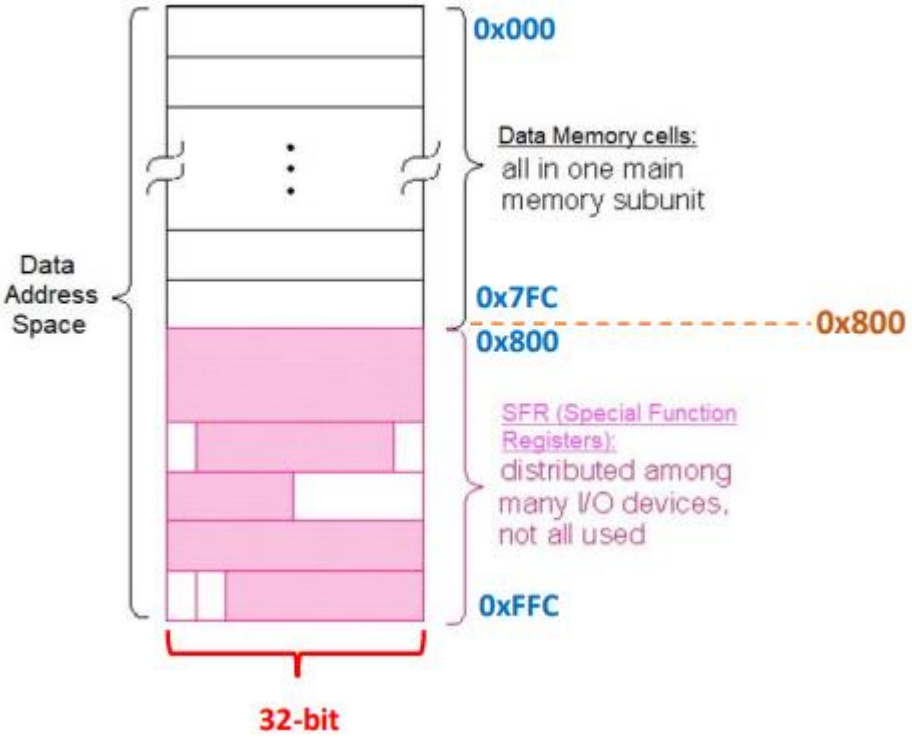
I/O devices and memory mapped I/O:

• Board eight switches (SW7-SW0) and push four debounced pushbuttons (KEY3-KEY0) will be used as Input interface.

• Board 8 red LEDs (LEDR7-LEDR0) and six 7-segment displays (HEX5-HEX0) used as Output interface.

• Connections between the 2x20 GPIO Expansion Header and Cyclone V SoC FPGA

In the next figure we can see how the data address space.



The Data Address Space is 32-bit WORD aligned where the physical address space it is the lowest 12-bit $\boldsymbol{0}$ … $\boldsymbol{0A11}$ … $\boldsymbol{A0}$ with partial mapping.

In the next figure we can see how the I/O devices are connected to the buses.

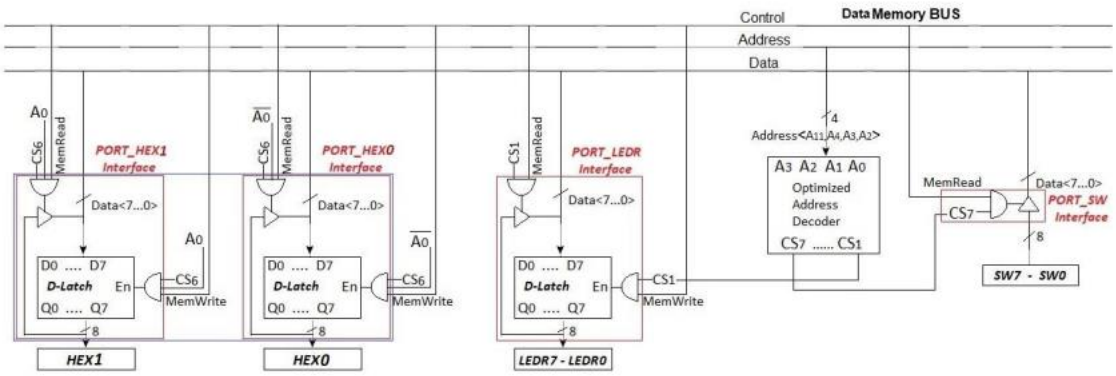


Figure 3: Primitive GPIO peripheral connection using Memory Mapped I/O approach

We were asked to implement the GPIO peripherals.
In the next figures we will see examples of the implementation:

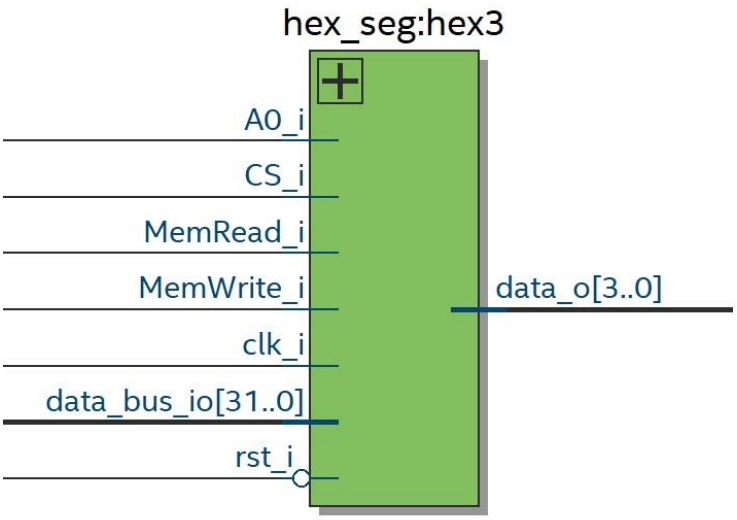


Figure 4- HEX seg component

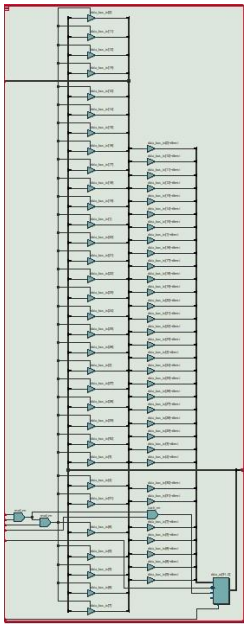


Figure 5 -RTL HEX seg

in figure 4 and 5 you can see how we have implemented the connection of one of our I/O devices to the busses.

Each component got the relevant Chip Select, Adress, MemRead_i, MemWrite_i bits to decide on the read and write from the component in order from avoid from collision of data.
Another example is of the switches:

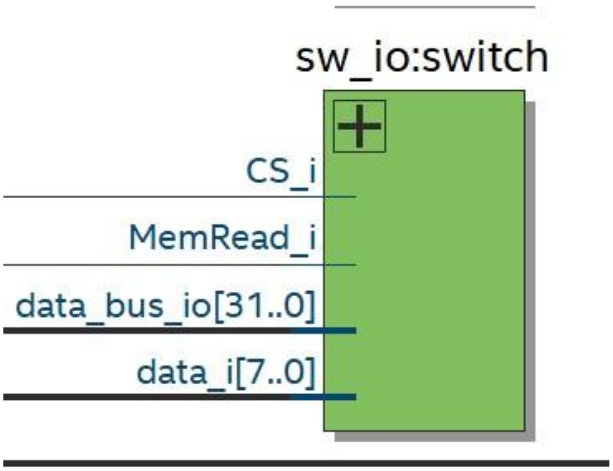


Figure 6 -Switches in_out component

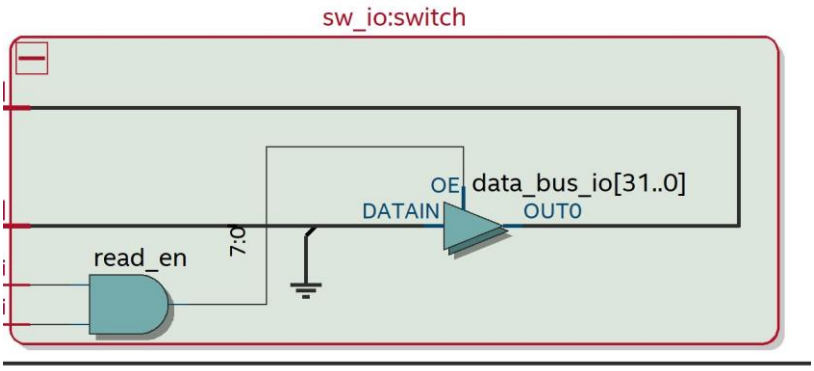


Figure 7- RTL SW

MIPS:

This is the CPU of our system. We have decided to use a single cycle architecture.

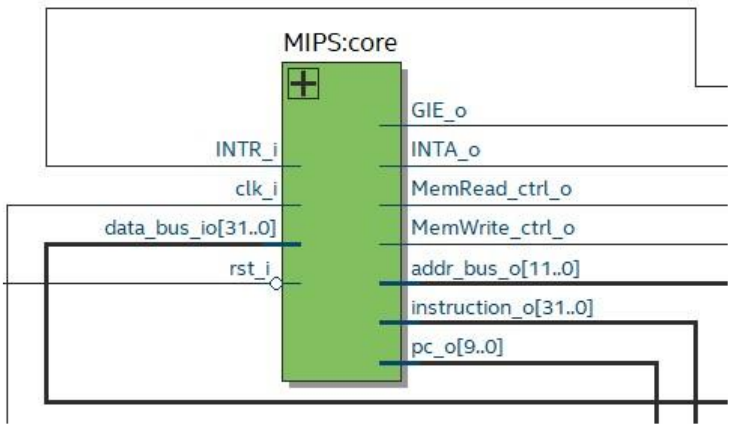


Figure 8 -MIPS component

Here we can see all the signals that enter and exit the MIPS component.

We have added INTR and INTA signals to handling interrupts.

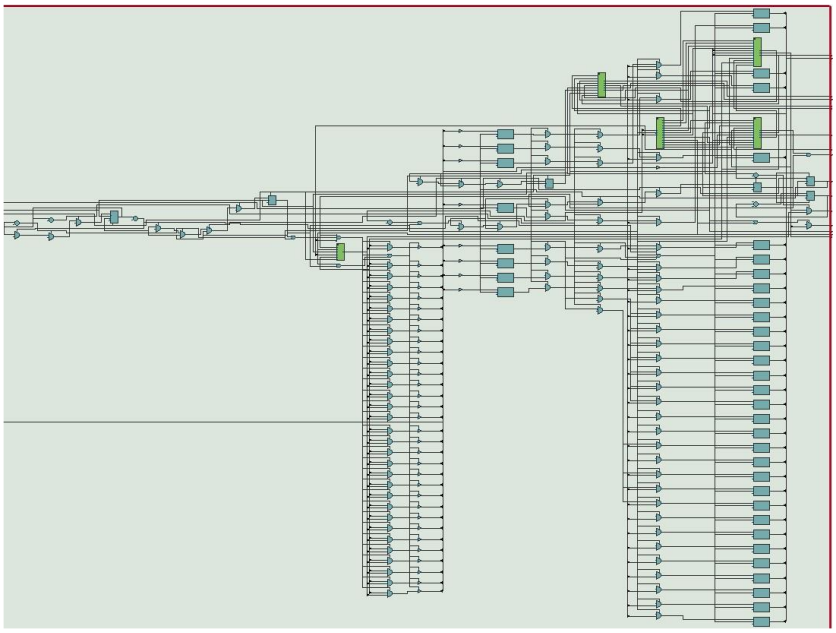


Figure 9 -RTL MIPS

And here we can impress from our RTL of the MIPS.

Basic Timer:

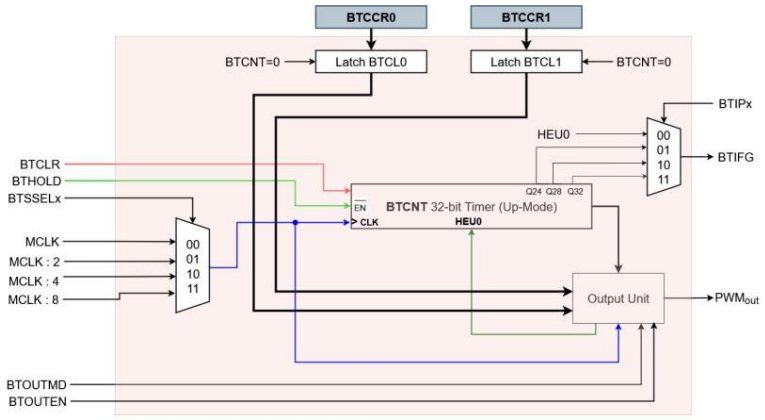


Figure 10 – Basic Timer

This component is a basic timer, it has the timer counter, the output unit (for creating PWM output), 4 clocks that are chosen by the BTSSELx , 2 registers – BTCCR0 and BTCCR1 who are memory mapped I/O and BTIFG signal that create an interrupt.

In the next figures, we will see how we have implemented the Basic Timer.

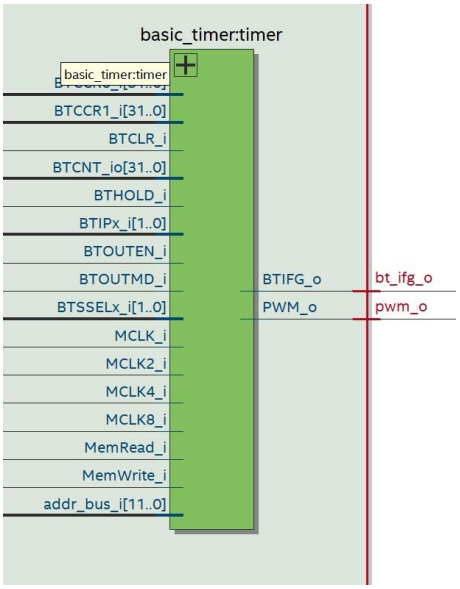


Figure 11 - Basic timer component

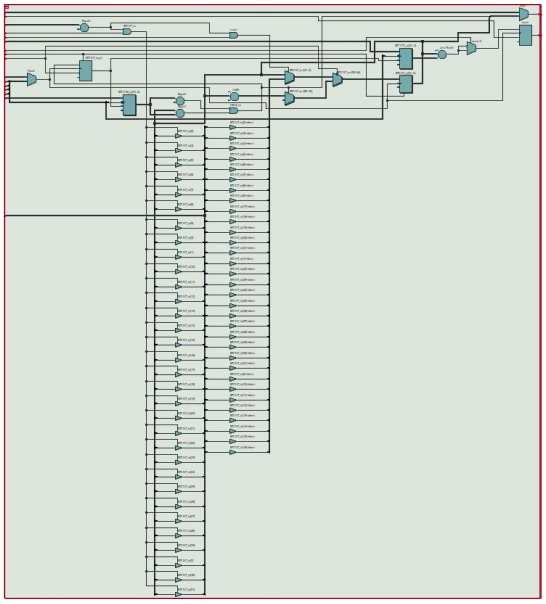


Figure 12 -RTL – basic timer

FIR:

we have been asked to implement a HW- Accelerator - FIR:

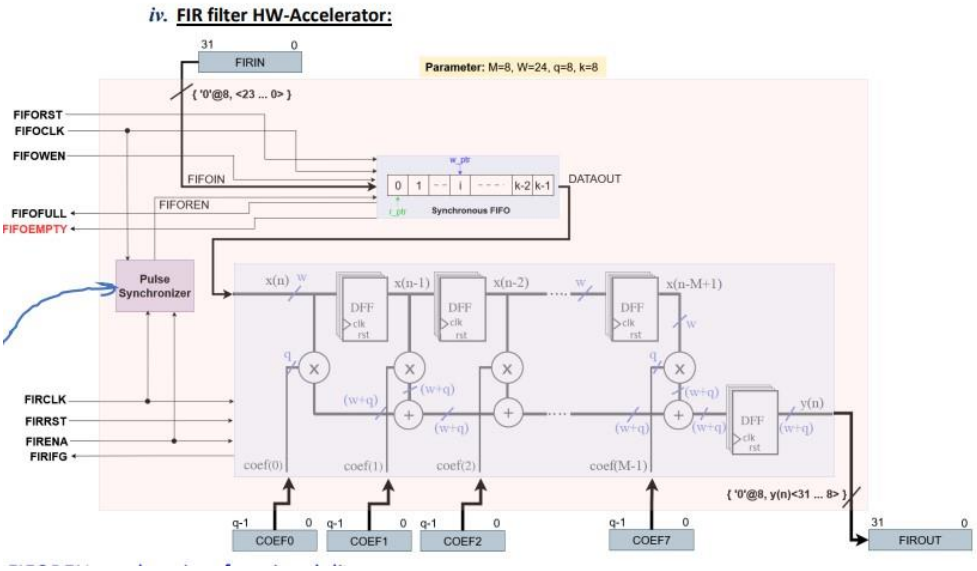


Figure 13 – FIR filter

we have implemented this component with FIR top and FIR sub-component:

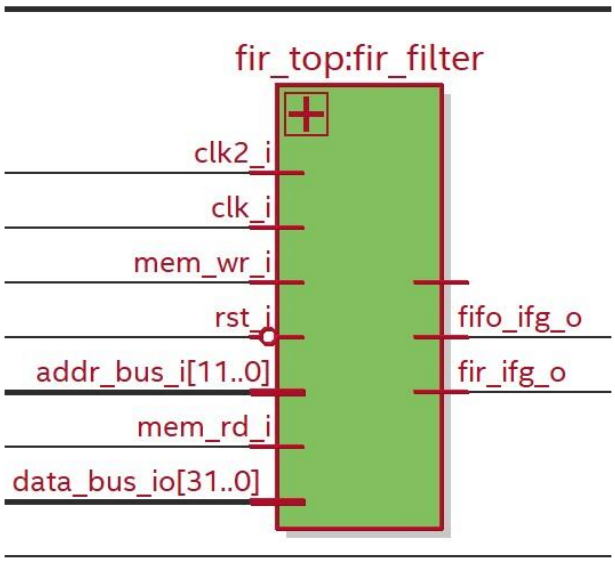


Figure 14 -FIR top component

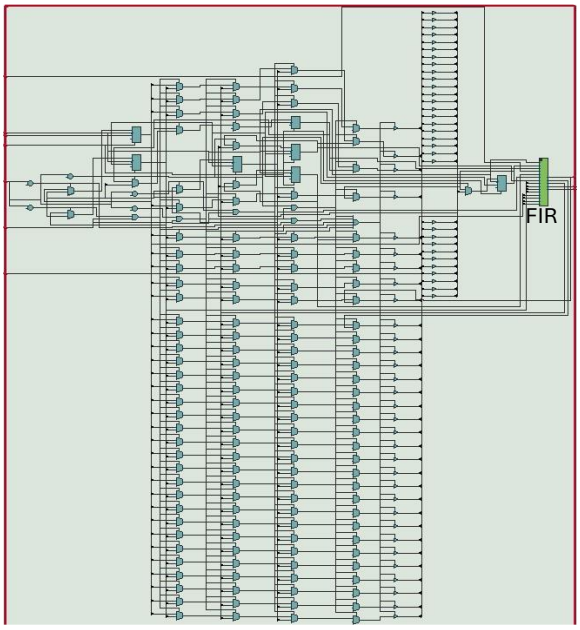


Figure 15 -RTL – FIR top

Than we create a FIR component that contains several subcomponents :

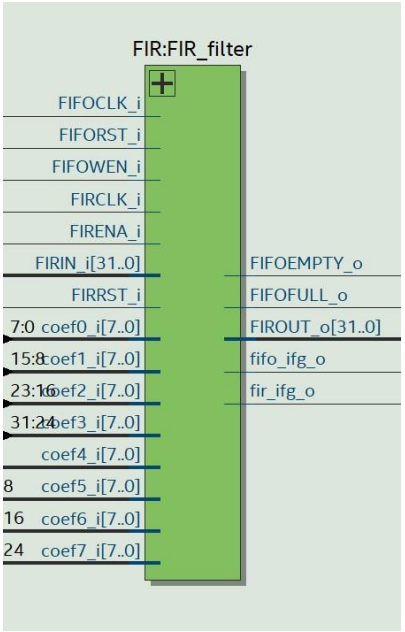


Figure 16 -FIR component

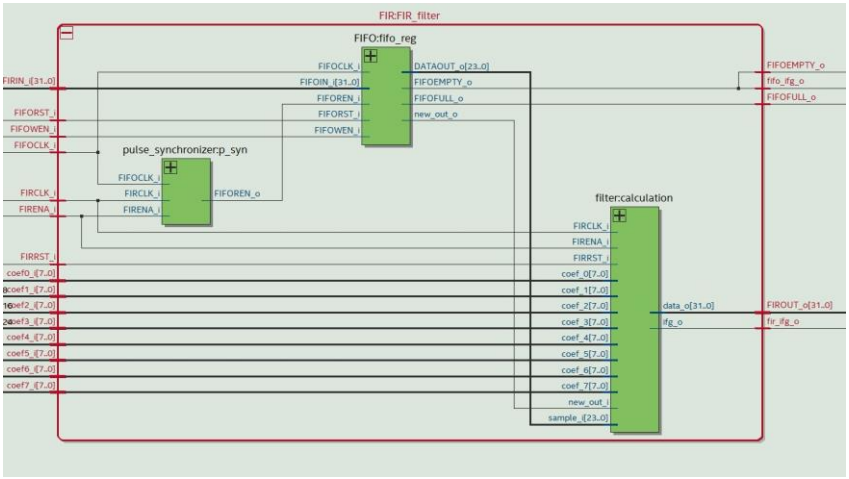


Figure 17 - RTL FIR

Our subcomponents are the pulse synchronizer, the FIFO register and FIR calculation:

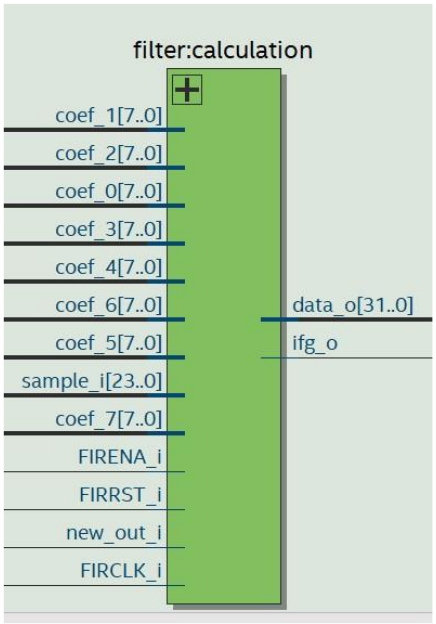
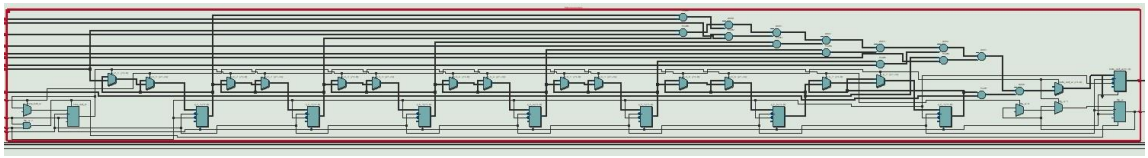


Figure 18 -Calculation component in the FIR



RTL – calculation component in the FIR

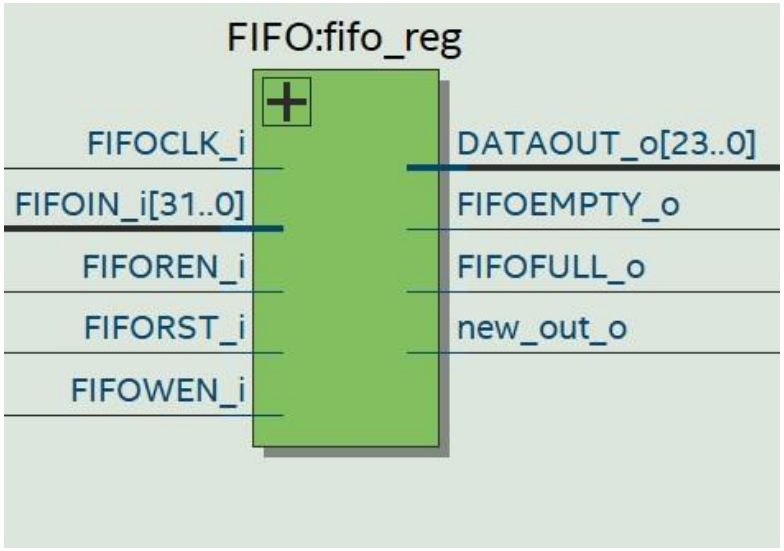
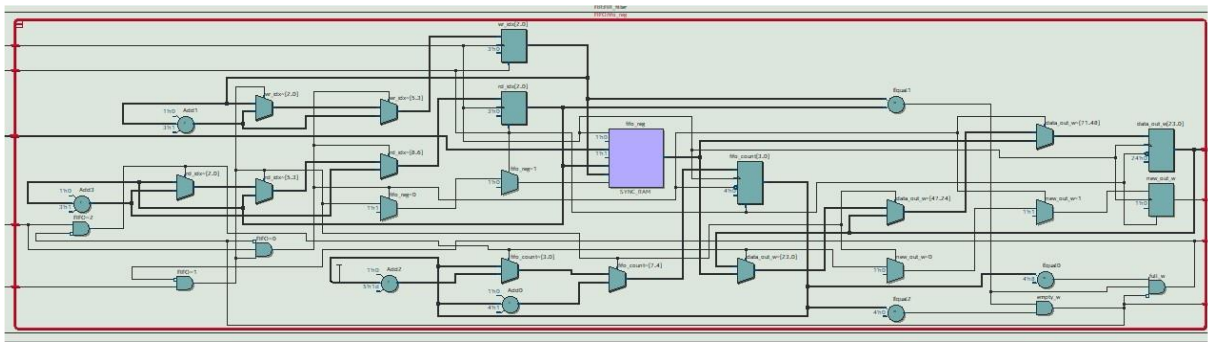


Figure 20 -FIFO register component



RTL – FIFO reg

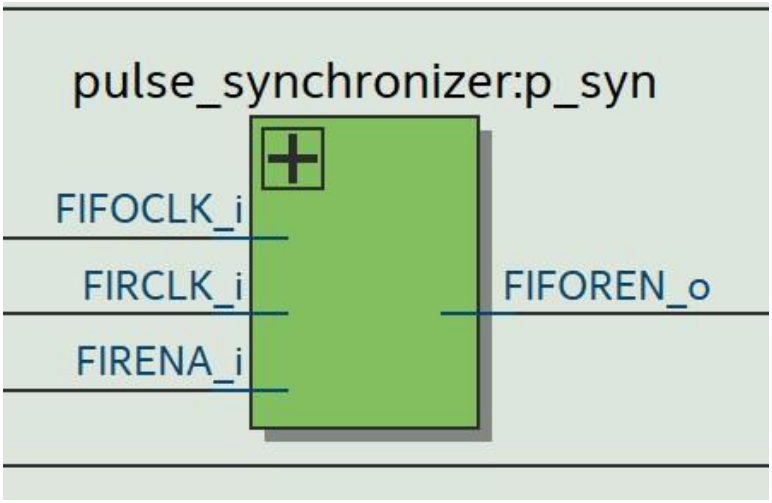
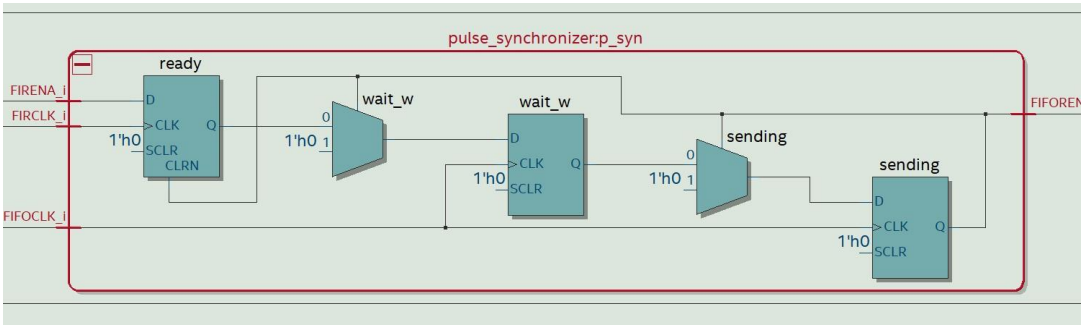


Figure 22 -Pulse Synchronizer component



RTL Pulse Synchronizer

We tried to implement as described in the mission (with some changes)

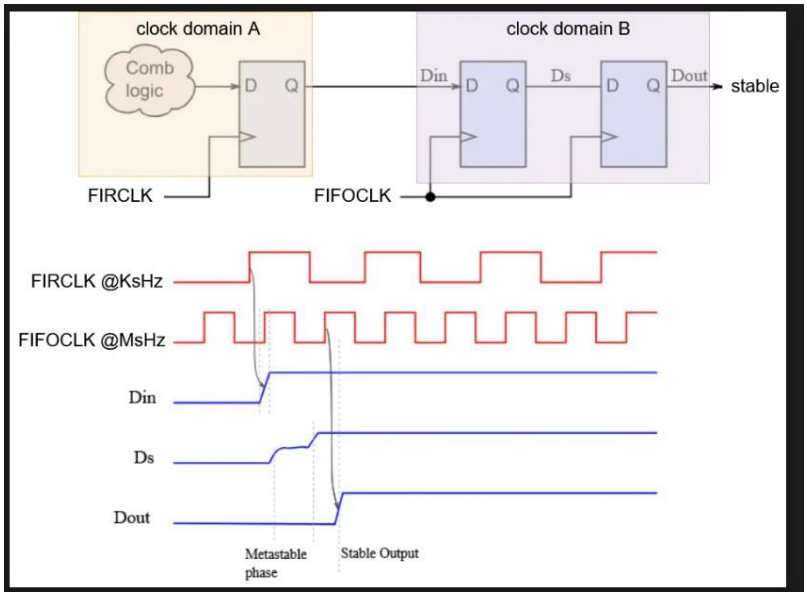
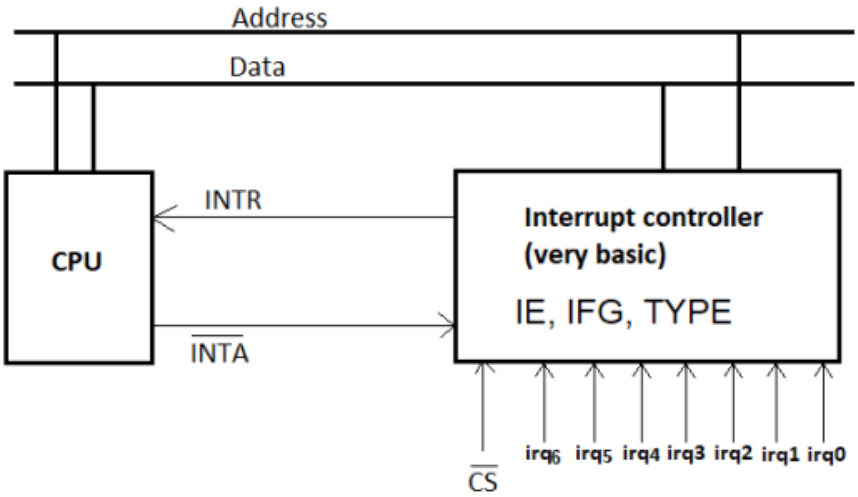


Figure 24 – pulse synchronizer “algorithm”

Interrupt controller:

vi. **Interrupt controller:**



When there is an interrupt and the user allows it (GIE = '1') the interrupt controller will send INTR = '1' when an interrupt have been captured (depending on IE value).

Then if the MIPS is available to handle the interrupt (currently not handling another interrupt) it will send INTA = '0' (acknolage).

The next step is that the interrupt controller will send the type of the interrupt that have been chosen by the highest priority algorithm.

In addition, the interrupt controller will send the enable register and the flag register according to the requirement (user read from its memory address).

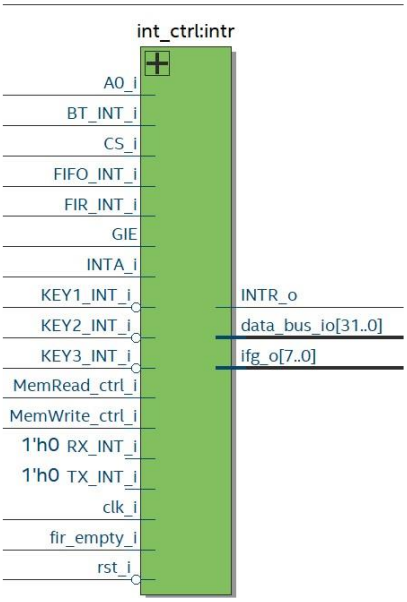
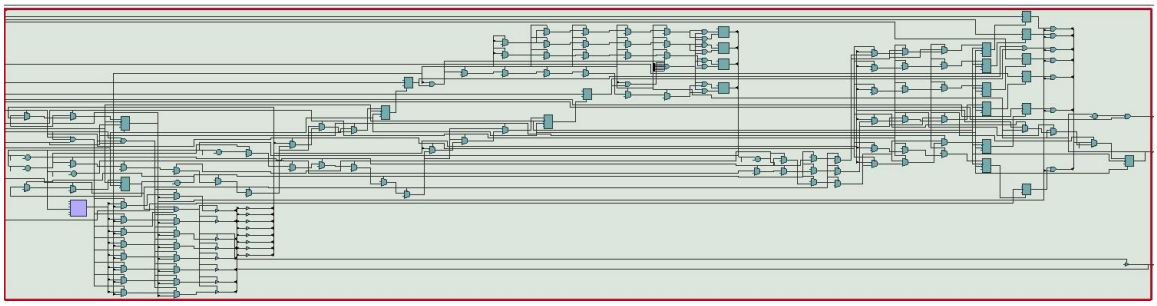


Figure 26 -Interrupt controller component



RTL interrupt controller

RTL of all the MCU:

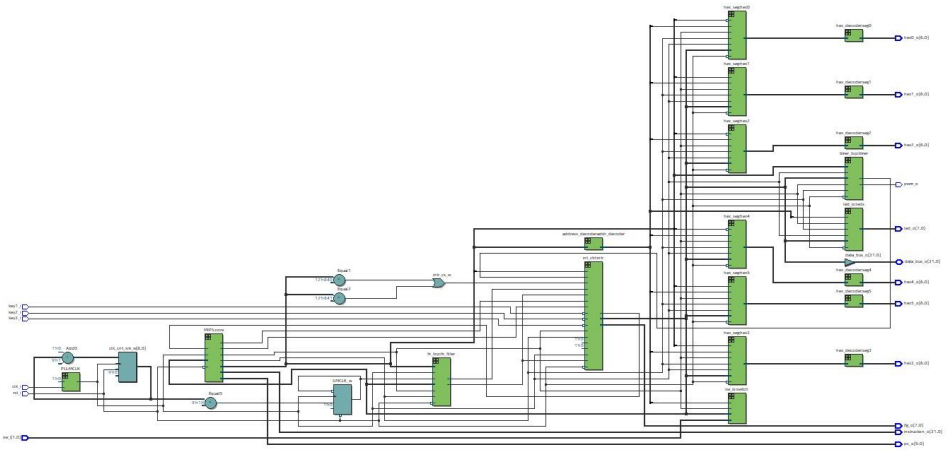
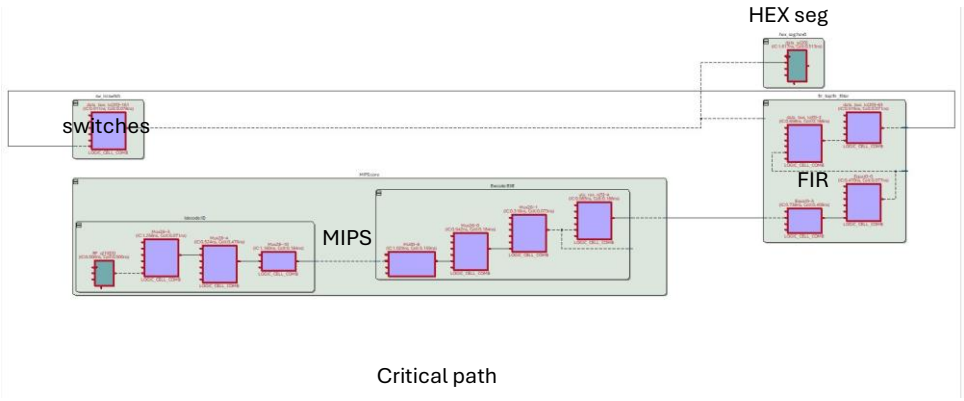


Figure 28 -RTL - MCU

Critical Path:

Our critical path is the longest path for the expansion of the data – its the slowest path of our system.



Critical path

And this is Fmax:

	Fmax	Restricted Fmax	Clock Name	Note
1	27.79 MHz	27.79 MHz	MCLK|altpll_co...COUNTER|divclk	
2	90.42 MHz	90.42 MHz	altera_reserved_tck	

Figure 30 -Fmax of the system

Test our system:

This is an example of test 1 :

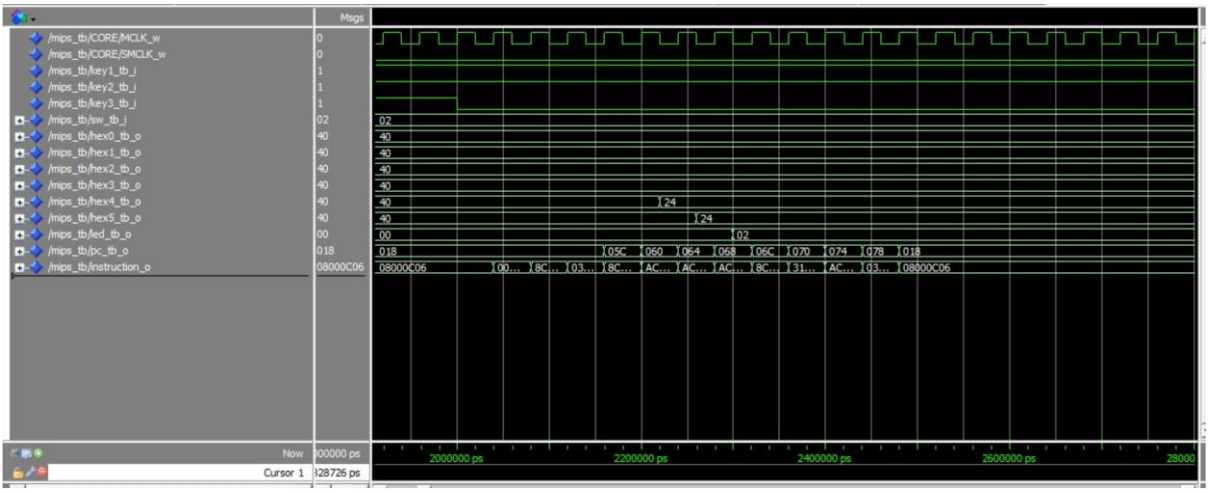


Figure 31 – model sim for test 1

here we can see the interrupt.

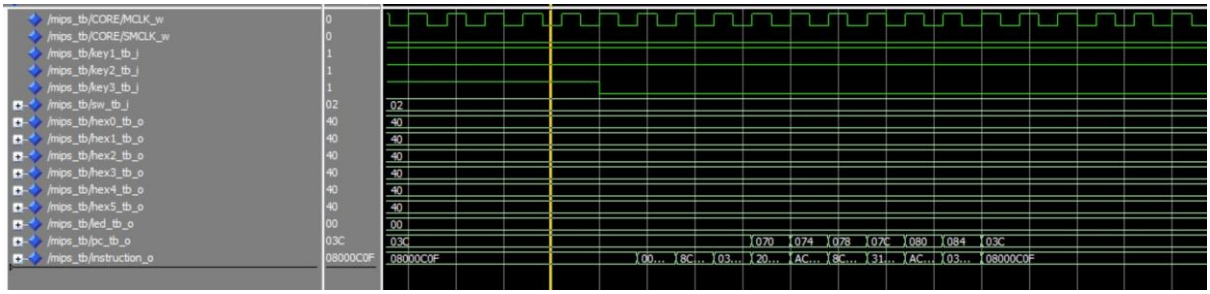


Figure 32 – Model Sim for test 2

Again an example for one of the interrupts.

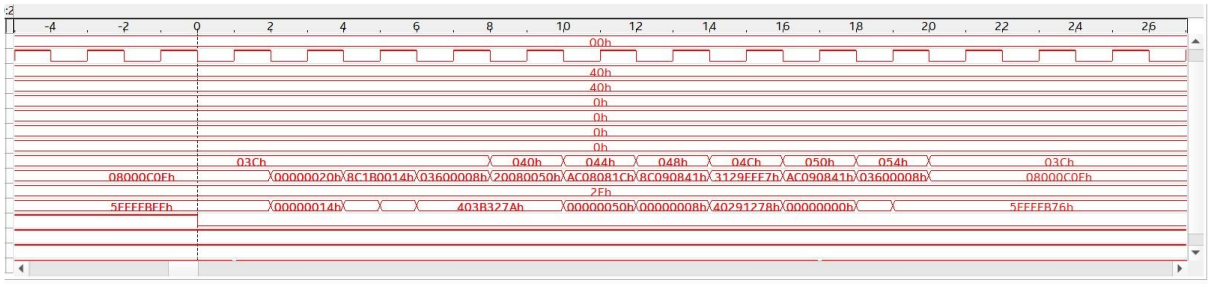


Figure 33 - Test 2 in Quartus

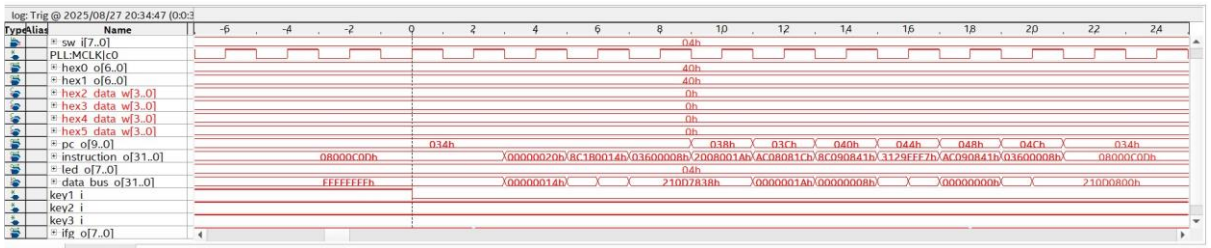


Figure 33 - Test 3 in Quarus

Figure 34 - Memory after running test 4 in the Quartus

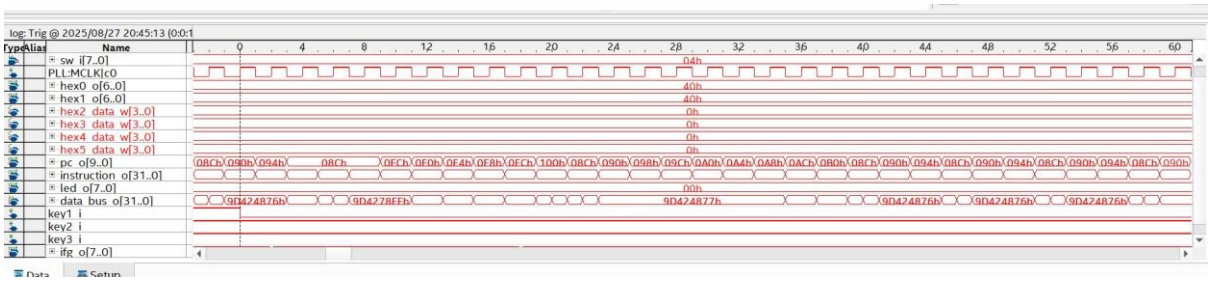


Figure 35 - Interrupt for test 4 in Quartus