

Final Project 2015

CPU Architecture 2015

Index

Background -----	2
1. Top level block diagram design -----	3
2. Deep review for main blocks in top level design -----	4 - 13
3. Critical path and Maximum clock frequency -----	13 - 16
4. I2C Communication -----	16
5. Interrupt Mechanism I2C-----	17
6. SignalTap Analysis -----	18 – 20
7. Interrupt Mechanism MIPS -----	22 – 23



Background

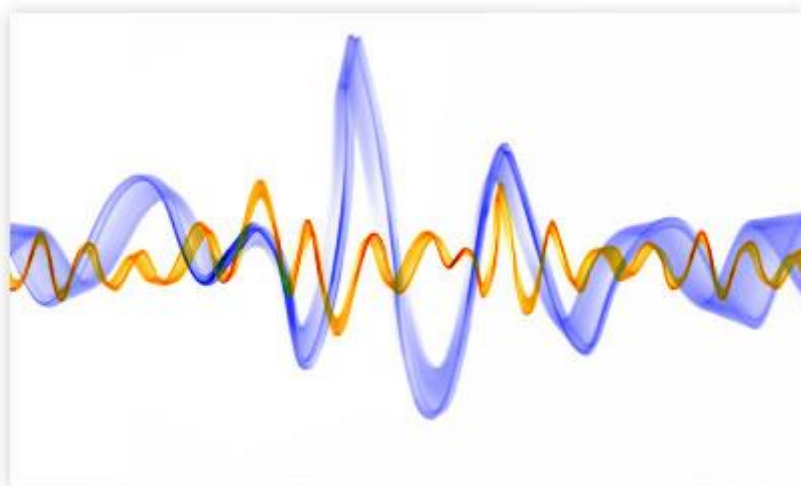
The aim of this project is to design LPF, Median, Variance and Mean filters in purpose of reducing noise from signals received from the tested ADXL345 accelerometer.

The communication between our core system (MIPS, FPGA memory and I/O interface) and the ADXL is implemented through I2C protocol, in which the core system serves as the master module and the ADXL as the slave module.

Our system is fully designed on interrupt mechanism on both sides. More on the interrupt mechanism will be describe later.

Our MIPS processor is a 5 stages pipelined structured which was design in the previous lab and got some enhancements in this projects, mainly on the interrupts aspect.

All the required filtered signals are displayed on a monitor screen, using the VGA protocol and the FPGA memory which we have implemented.



1. Top level block diagram review

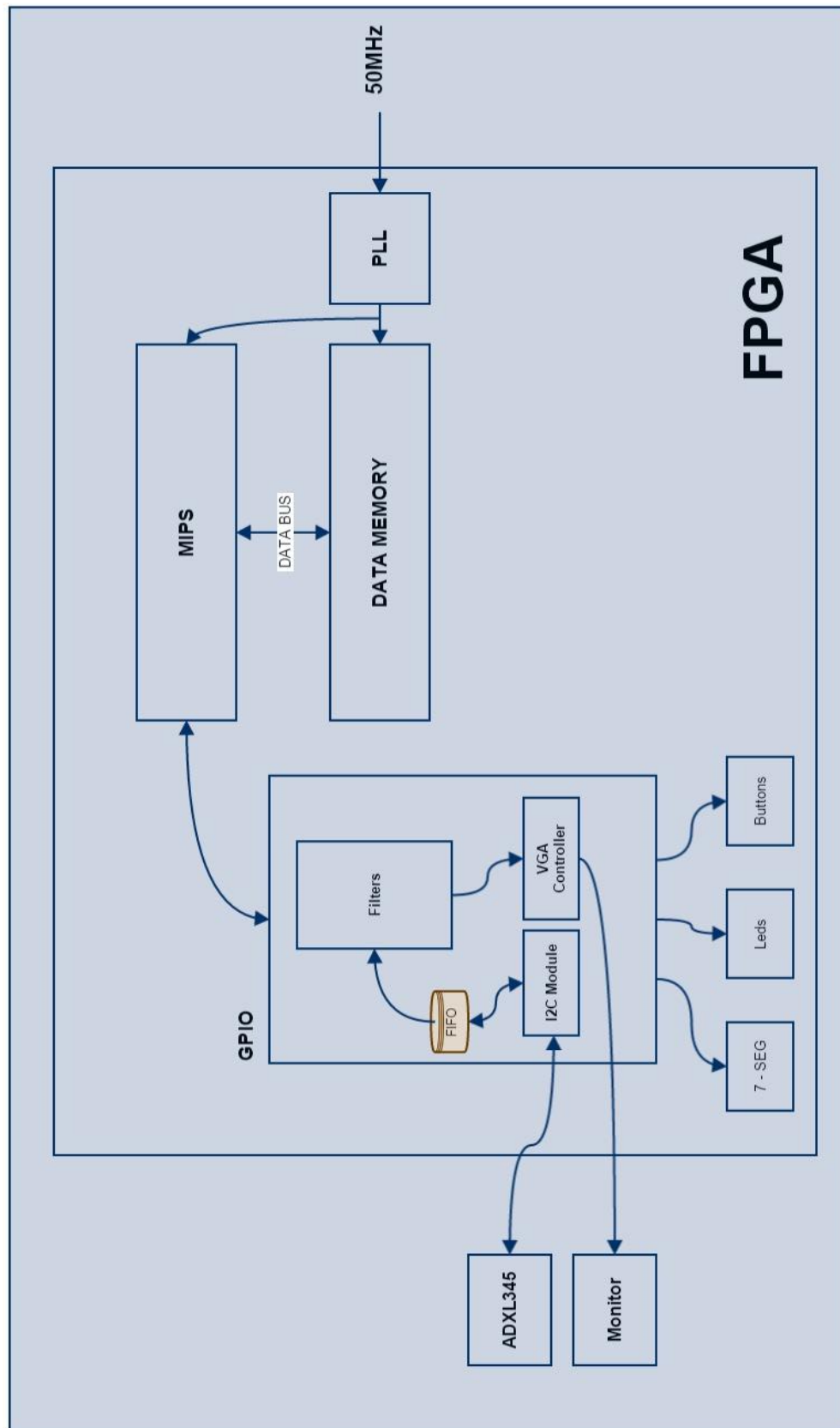


Figure 1

2. Deep review for blocks in top design

2.1 GPIO

The GPIO module is an intermediate stage between I/O's and the MIPS architecture. GPIO module is functioned by addressing specific locations. The GPIO also includes the Filter modules.

RTL Viewer



Figure 2

Port Table

Port	Direction	Size (Bit)	Functionality
Clock	IN	1	
InterruptADXL	IN	1	When ADXL has new data
MemRead	IN	1	
MemWrite	IN	1	
reset	IN	1	
Address	IN	32	Choosing the required IO
Write Data	IN	32	
I2C Read Interrupt	OUT	1	
I2C Write Interrupt	OUT	1	
SCL	INOUT	1	Communication
SDA	INOUT	1	Communication
Read Data	OUT	32	
Segment0	OUT	7	
Segment1	OUT	7	
Segment2	OUT	7	
Segment3	OUT	7	

Table 1

The GPIO addressing code in GPIO.vhd.

```

if (Memwrite = '1') then

    case address is
        -- 7-Segments control
        when X"00000404" =>
            data_to_7_seg <= write_data;
        -- I2C Master control - write to slave
        when X"00000408" =>
            fifo1_data_in <= write_data(7 downto 0);
            fifo1_write_enable <= '1';
        when X"0000040C" => LPS_Coef0 <= write_data;

        when X"00000410" => LPS_Coef1 <= write_data;

        when X"00000414" => LPS_Coef2 <= write_data;

        when X"00000418" => LPS_Coef3 <= write_data;

        when X"0000041C" => LPS_Coef4 <= write_data;

        when X"00000420" => LPS_Coef5 <= write_data;

        when X"00000424" => LPS_Coef6 <= write_data;

        -- FIFO2 control
        when X"00000430" =>
            fifo2_data_in <= write_data(15 downto 0);
            fifo2_read_enable <= '1';
            fifo2_write_enable <= '1';
        when others => null;

    end case;

```

Code 1

Graphical Description

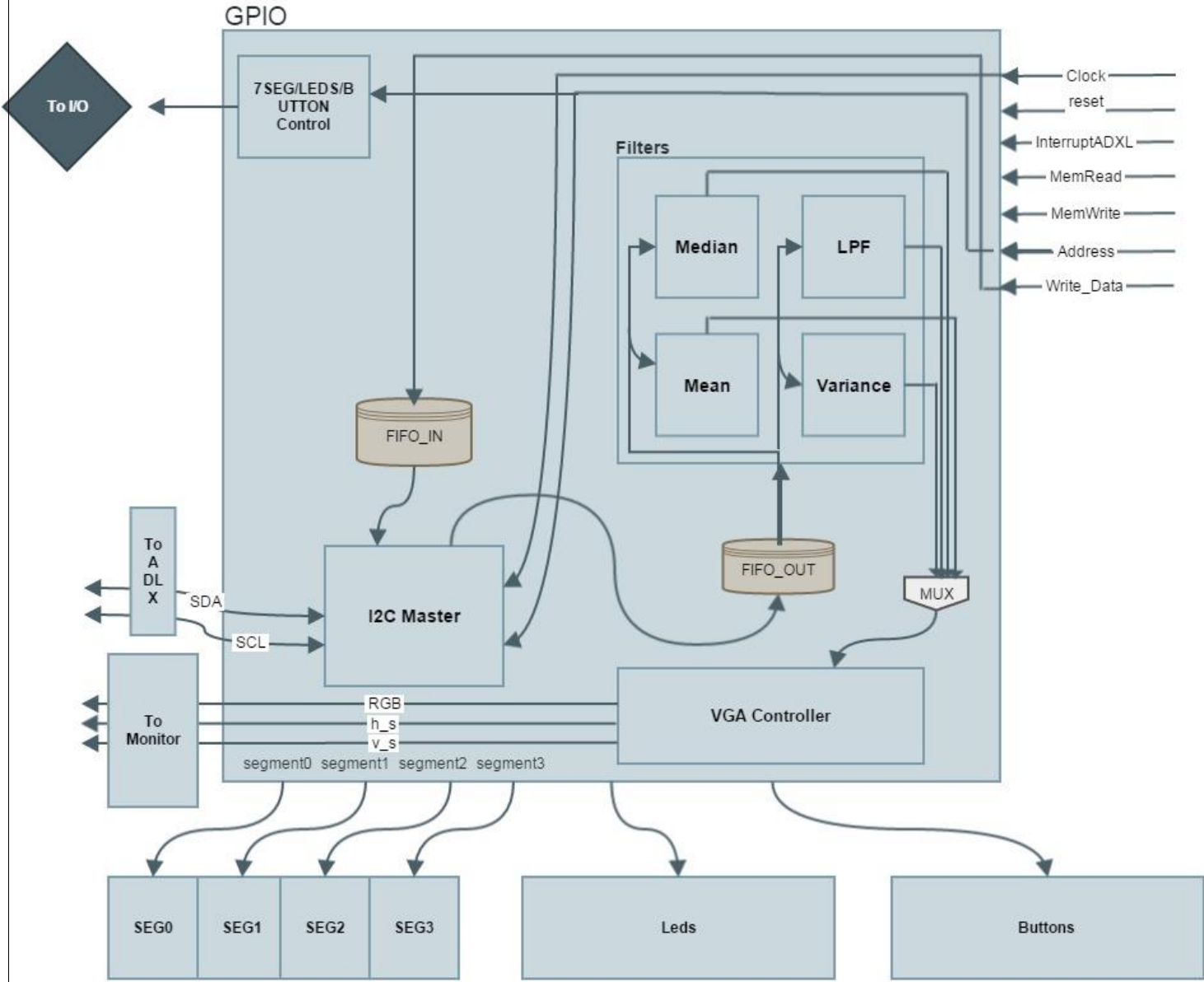


Figure 3

2.2 MIPS

5 stages Pipelined MIPS CPU architecture.

RTL Viewer

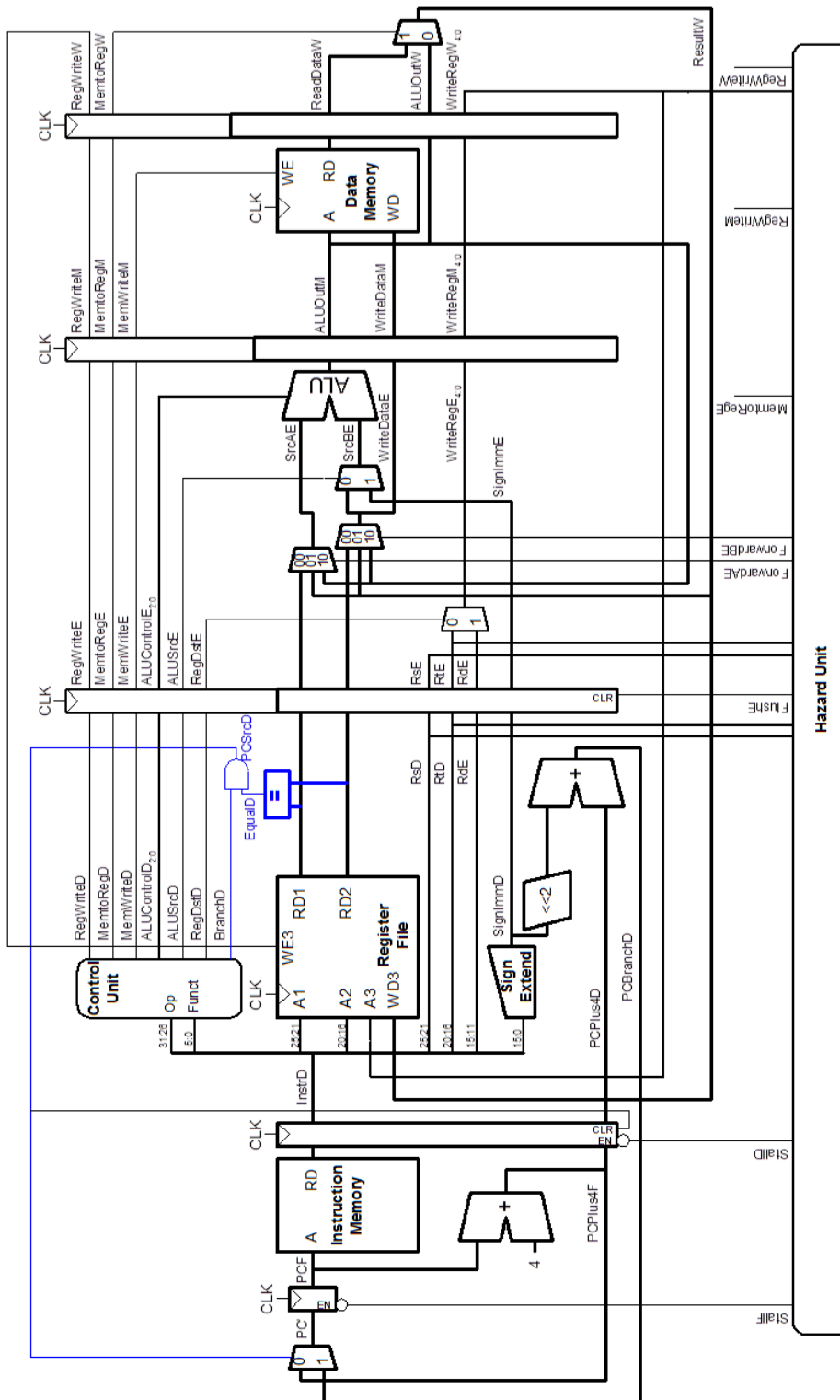


Figure 4

Port Table

Port	Direction	Size (Bit)	Functionality
Clock	IN	1	System Clock
Reset	IN	1	System Reset

Table 2

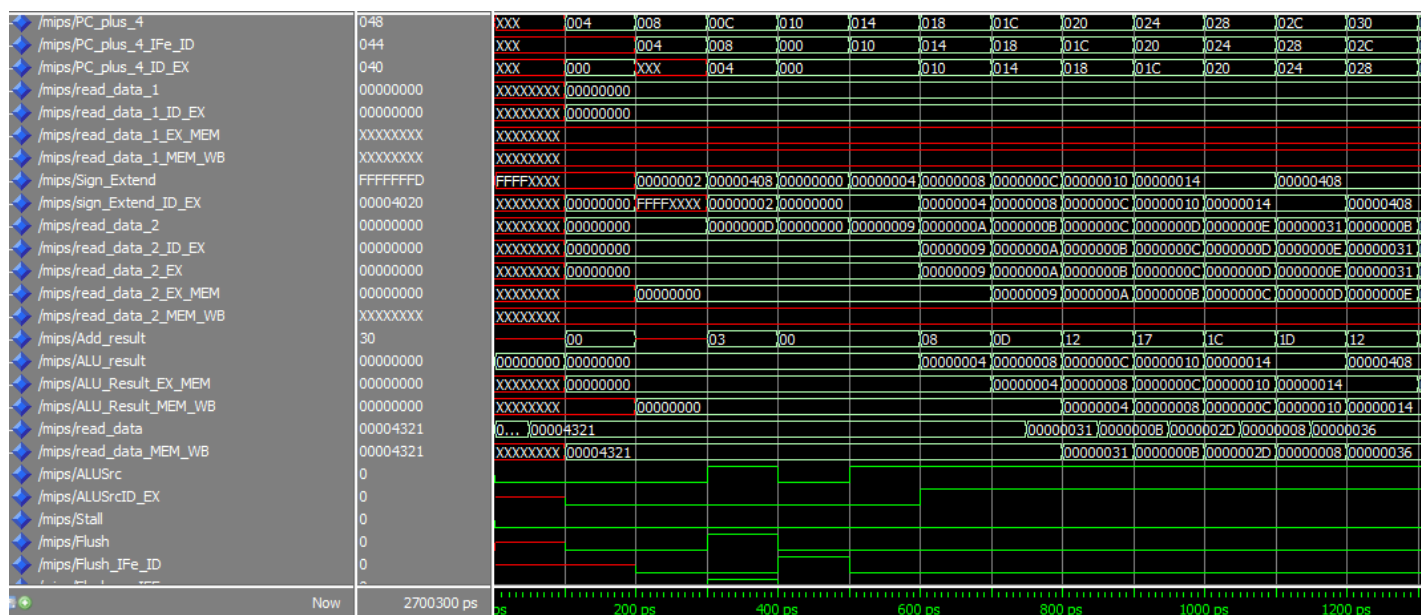


Figure 25

MIPS – Loading instructions, I2C settings and get data from slave

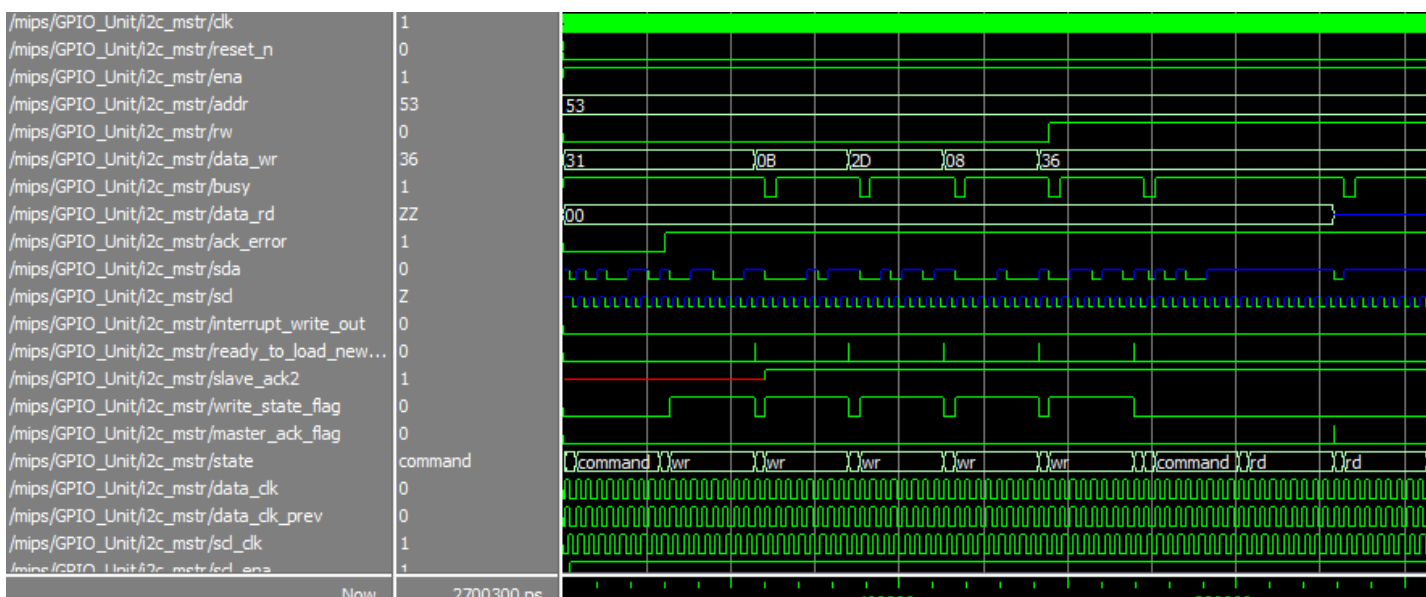


Figure 26

2.3 Mean Filter

Mean Filter collects the last 7 wave samples, add them all together and multiply by a certain factor. In our case the factor is 9/4 (to make calculation and implementation easier).

RTL View

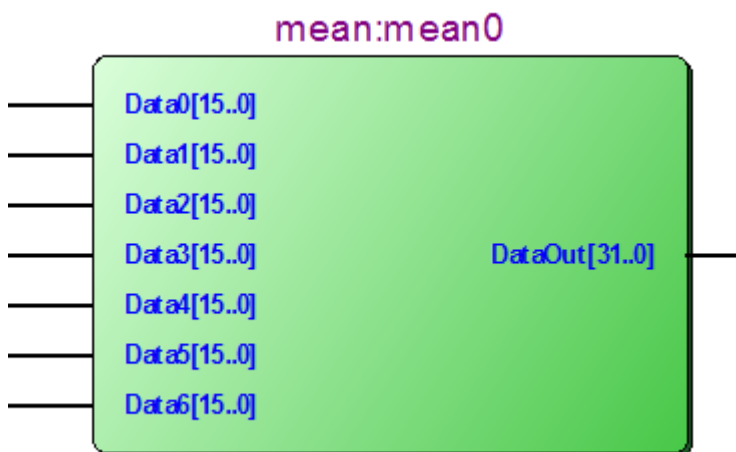


Figure 5

Port table

Port	Direction	Size (Bit)	Functionality
Data0	IN	16	NONE
Data1	IN	16	NONE
Data2	IN	16	NONE
Data3	IN	16	Data sample
Data4	IN	16	NONE
Data5	IN	16	NONE
Data6	IN	16	NONE
DataOut	OUT	32	Data filtered

Figure 3

Graphical View

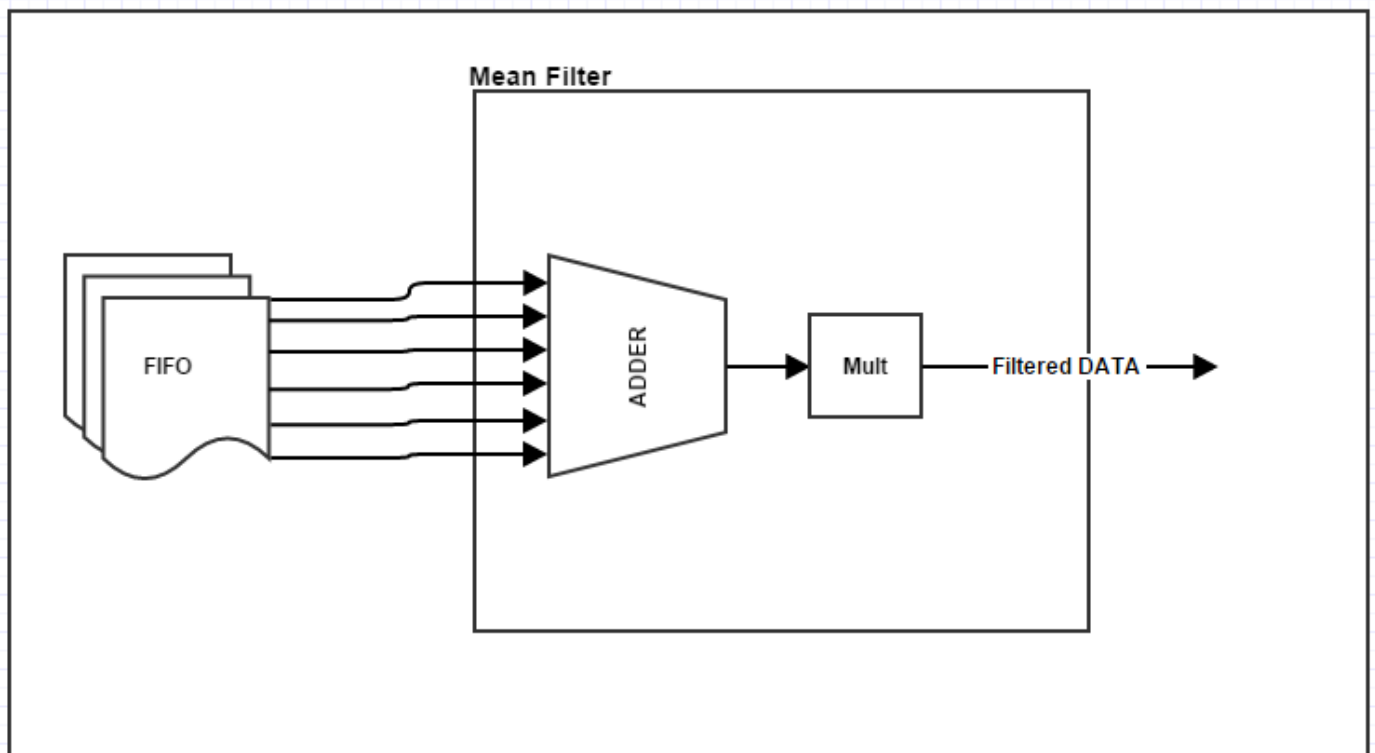


Figure 6

2.4 LPF Filter

A **low-pass filter** is a **filter** that passes signals with a frequency lower than a certain cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. The amount of attenuation for each frequency depends on the **filter** design.

The way we implemented our LPF filter is by calculating the right coefficients for the correspond frequency rate, then, we multiply every sample with its matching coefficient, and finally we summed up all the multiplications. The result is the filtered LPF output.

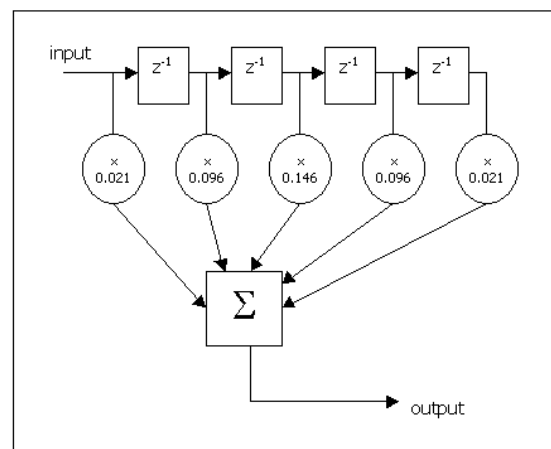


Figure 7

RTL View

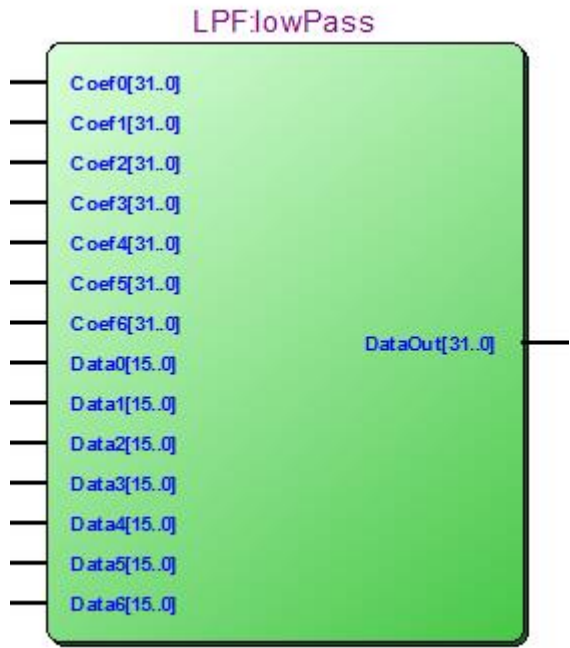


Figure 8

Graphical View

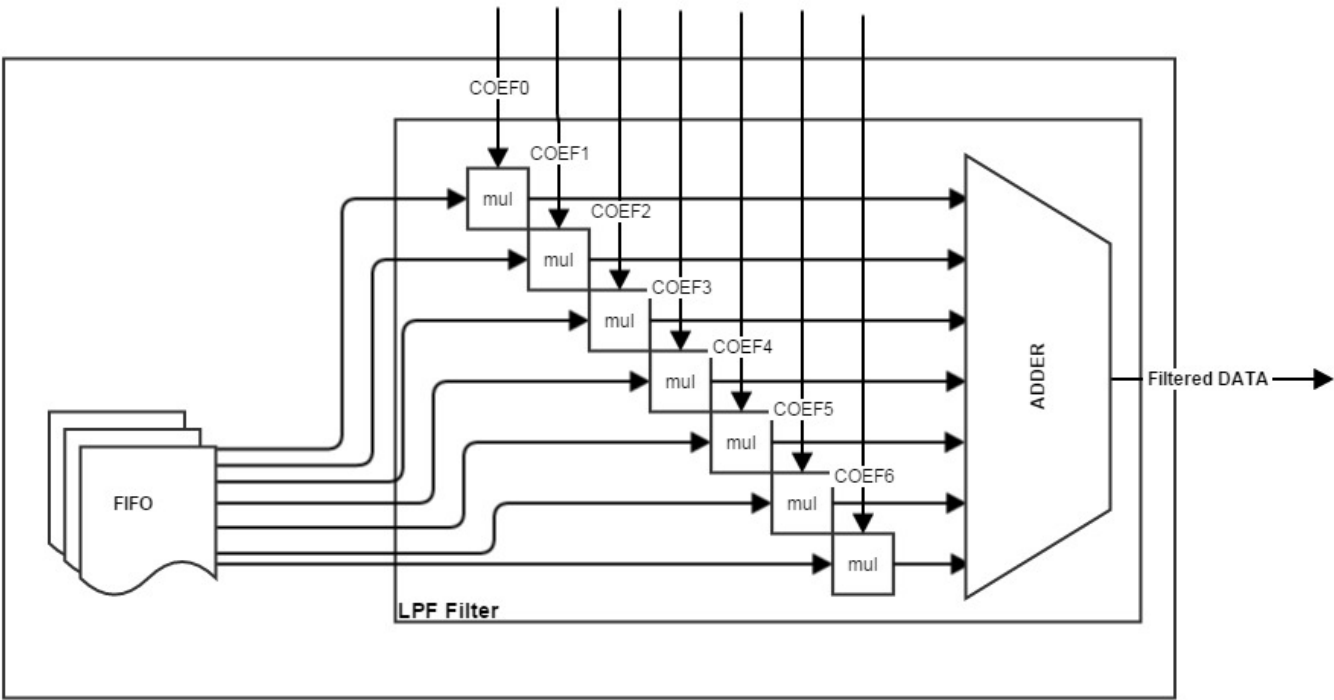


Figure 9

Port table

Port	Direction	Size (Bit)	Functionality
Data0	IN	16	FIFO Sample
Data1	IN	16	FIFO Sample
Data2	IN	16	FIFO Sample
Data3	IN	16	FIFO Sample
Data4	IN	16	FIFO Sample
Data5	IN	16	FIFO Sample
Data6	IN	16	FIFO Sample
Coef0	IN	32	Filter Coefficient
Coef1	IN	32	Filter Coefficient
Coef2	IN	32	Filter Coefficient
Coef3	IN	32	Filter Coefficient
Coef4	IN	32	Filter Coefficient
Coef5	IN	32	Filter Coefficient
Coef6	IN	32	Filter Coefficient
DataOut	OUT	32	Filtered DATA

Table 5

2.5 Median Filter

The **median filter** is a nonlinear digital **filtering** technique, often used to remove noise. Such noise reduction is a typical pre-processing step to improve the results of later processing

RTL View

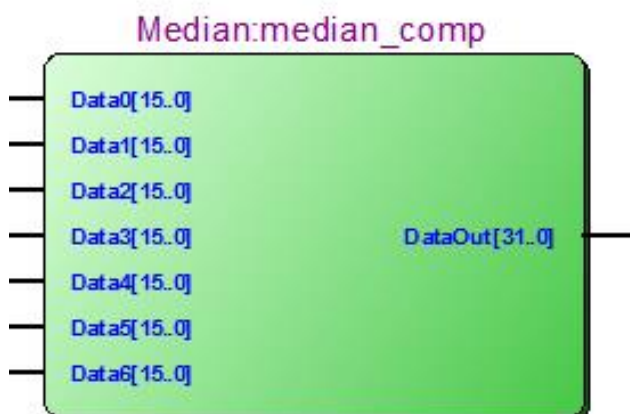


Figure 10

Graphical View

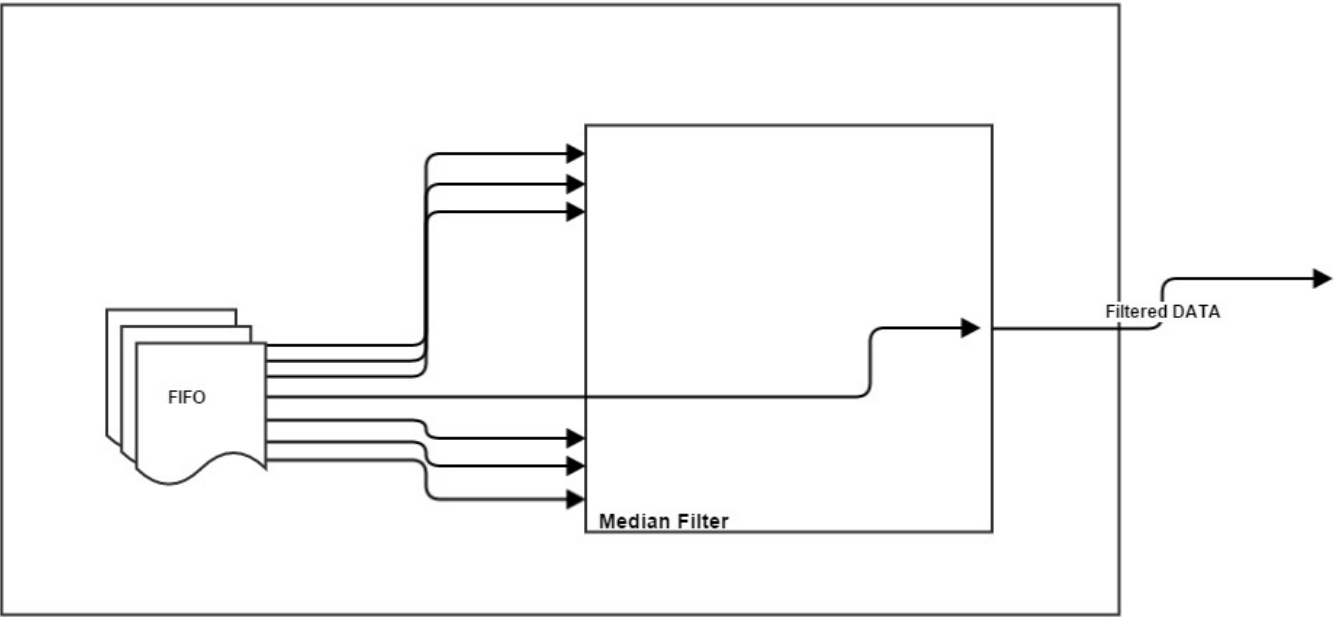


Figure 11

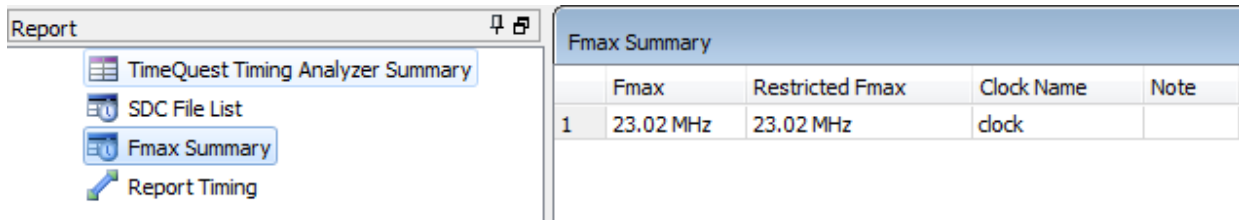
Port table

Port	Direction	Size (Bit)	Functionality
Data0	IN	16	None
Data1	IN	16	None
Data2	IN	16	None
Data3	IN	16	Selected Line
Data4	IN	16	None
Data5	IN	16	None
Data6	IN	16	None
DataOut	OUT	32	Filtered data

Table 6

3. Critical path and Maximum clock frequency

3.1 Fmax



The screenshot shows the 'Report' window in Xilinx ISE. On the left, a tree view lists reports: 'TimeQuest Timing Analyzer Summary', 'SDC File List', 'Fmax Summary' (selected), and 'Report Timing'. The main pane displays the 'Fmax Summary' report, which is a table with the following data:

Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	23.02 MHz	23.02 MHz	clock	

Figure 12

	Property	Value
1	From Node	gpio:GPIO_Unit fifo2_read_enable
2	To Node	gpio:GPIO_Unit STD_FIFO:fifo2 Data3[11]
3	Launch Clock	clock (INVERTED)
4	Latch Clock	clock
5	Data Arrival Time	51.799
6	Data Required Time	58.904
7	Slack	7.105

Figure 13

Maximum clock frequency: 23.02 MHz

3.2 Critical Path of Design

- Critical Path part I

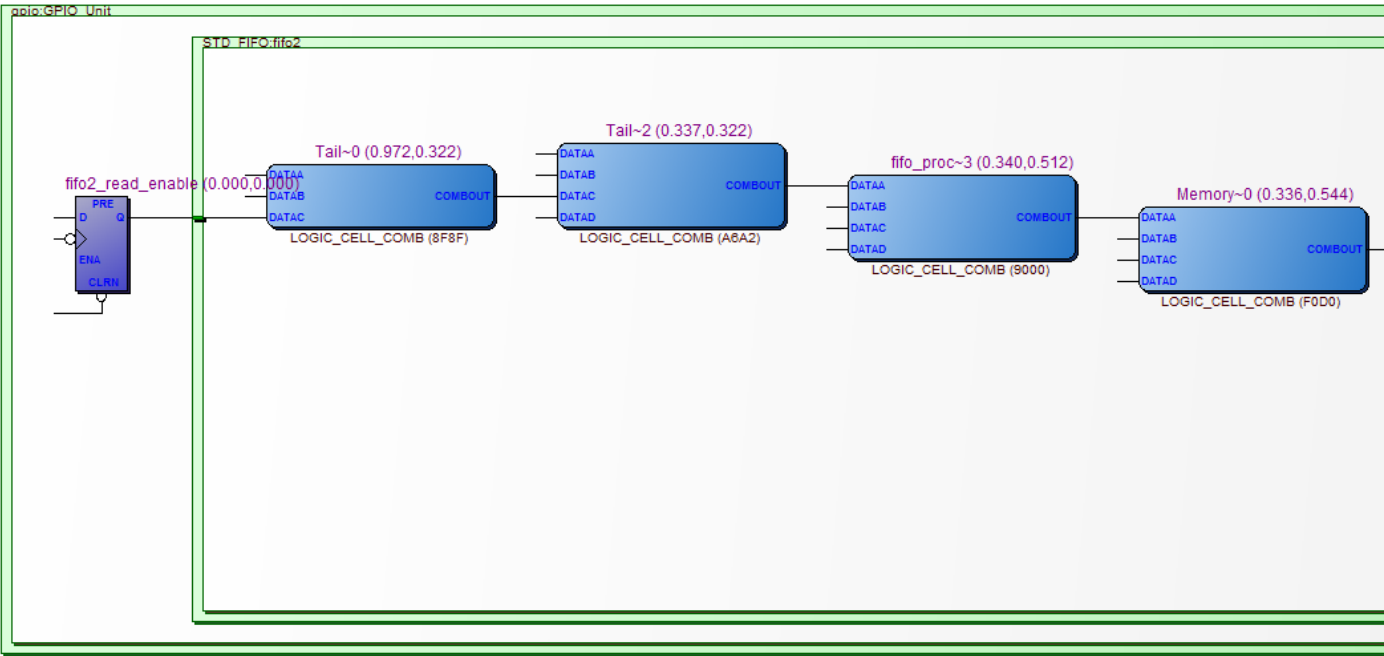


Figure 14

- Critical Path part II

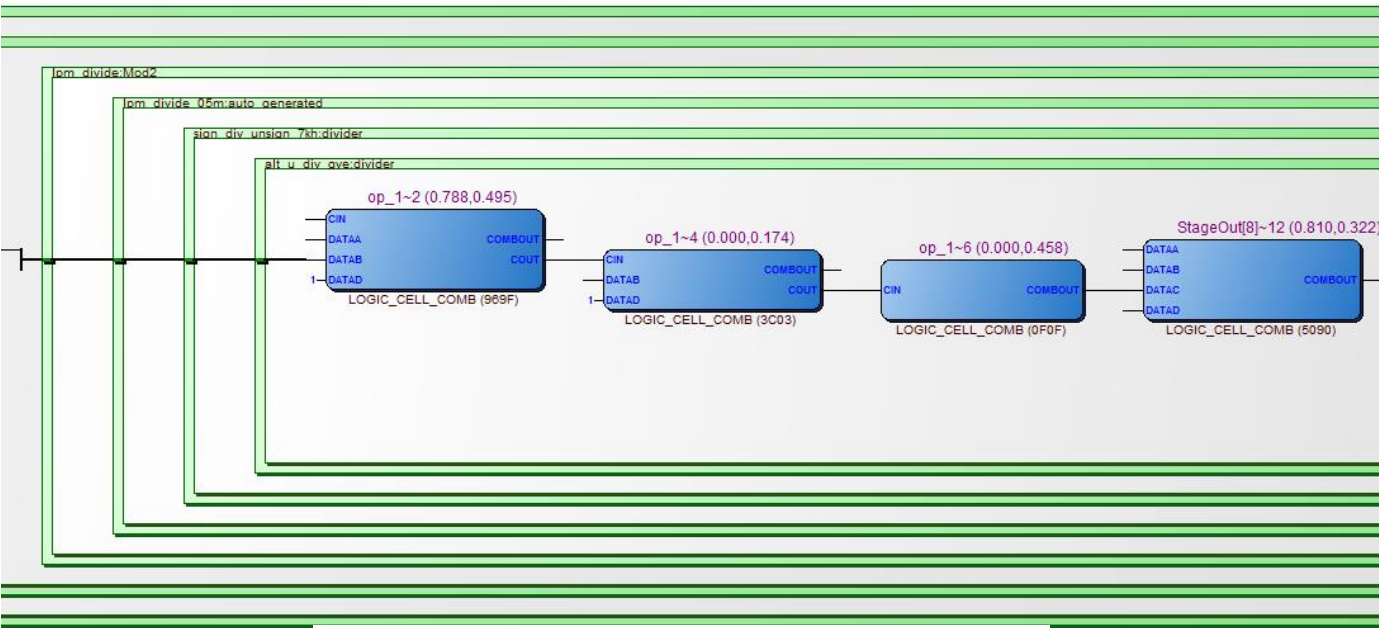


Figure 15

The critical path of our system is lined through FIFO2. FIFO2 is the component which capture data received from the I2C Master, which located in the GPIO component. The output of FIFO2 is connected to the filters.

4. I2C Communication

4.1 Logic Capture

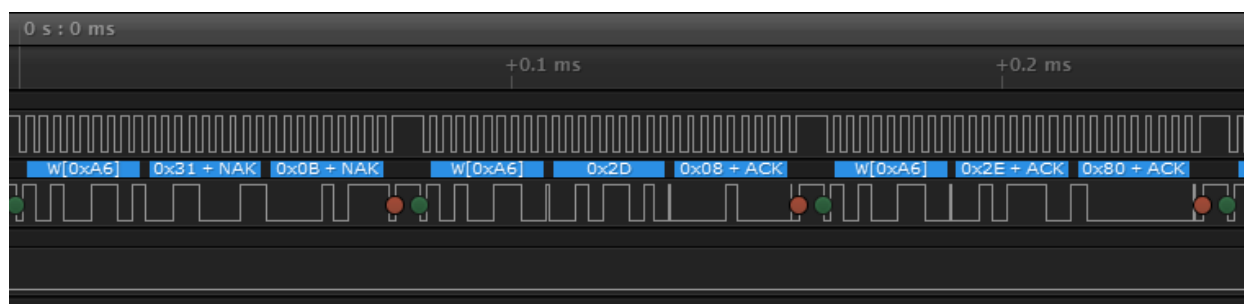


Figure 16 – Initializing I2C Slave to required settings

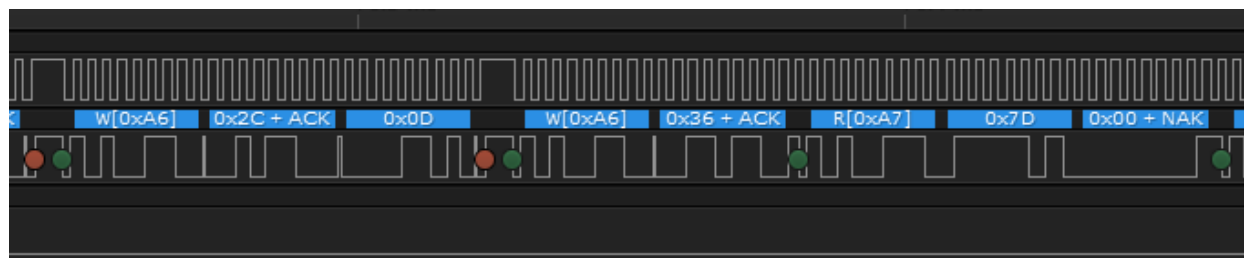


Figure 17– Initializing I2C Slave to required settings and sending Read Request

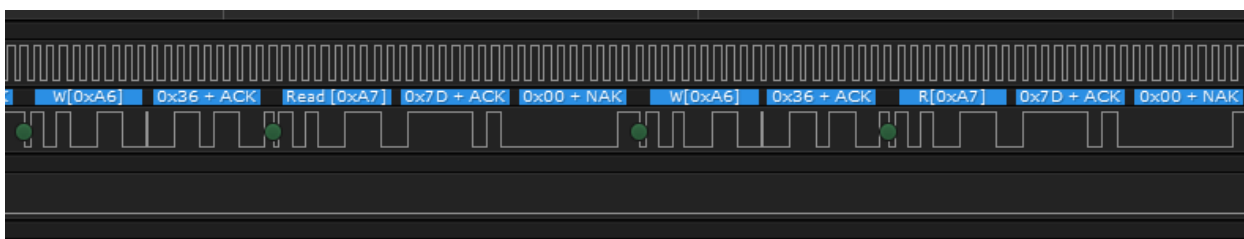


Figure 18 – Sending read request and receiving 2 bytes per request

5. Interrupts Mechanism

Interrupts from I2C - Sending interrupt flag to the MIPS.

Here we will see when exactly we send interrupt flag to the MIPS. Our goal is to send interrupt flag (for one clock cycle) only when a new data was received from the ADXL. For that reason, we connected the INT1 pin to the FPGA, and when a new sample was taken, we get '1' for a bunch of clock cycles.

However, by the time we want to save the data to the registers, this pin goes back down to '0'. For that reason we keep it in a signal called "data_has_changed". This signal remains '1' until we decide to lower it down.

So when do we raise an interrupt to the MIPS?

There are 3 conditions:

- 1) I2C went to Master Acknowledge state.
- 2) First byte was already received
- 3) Data Has Changed flag is up

Under these 3 conditions we know the we already received the entire data, and this data is a new sample from the ADXL. In the code it looks like this:

```
if (master_ack_flag = '1' and first_byte_received = '1') then
    first_byte_received <= '0';
    data_read_HI <= masterDataRead;
    if (data_has_changed = '1') then
        data_has_changed <= '0';
        interrupt_read_out <= '1';
    end if;
elsif (master_ack_flag = '1' and first_byte_received = '0') then
    first_byte_received <= '1';
    data_read_LO <= masterDataRead;
end if;
```

Code 2

Whereas 'masterDataRead' is the OUT signal from the I2C component.

6. Signal Tap proof of work

a. An interrupt was sent after data was changed:

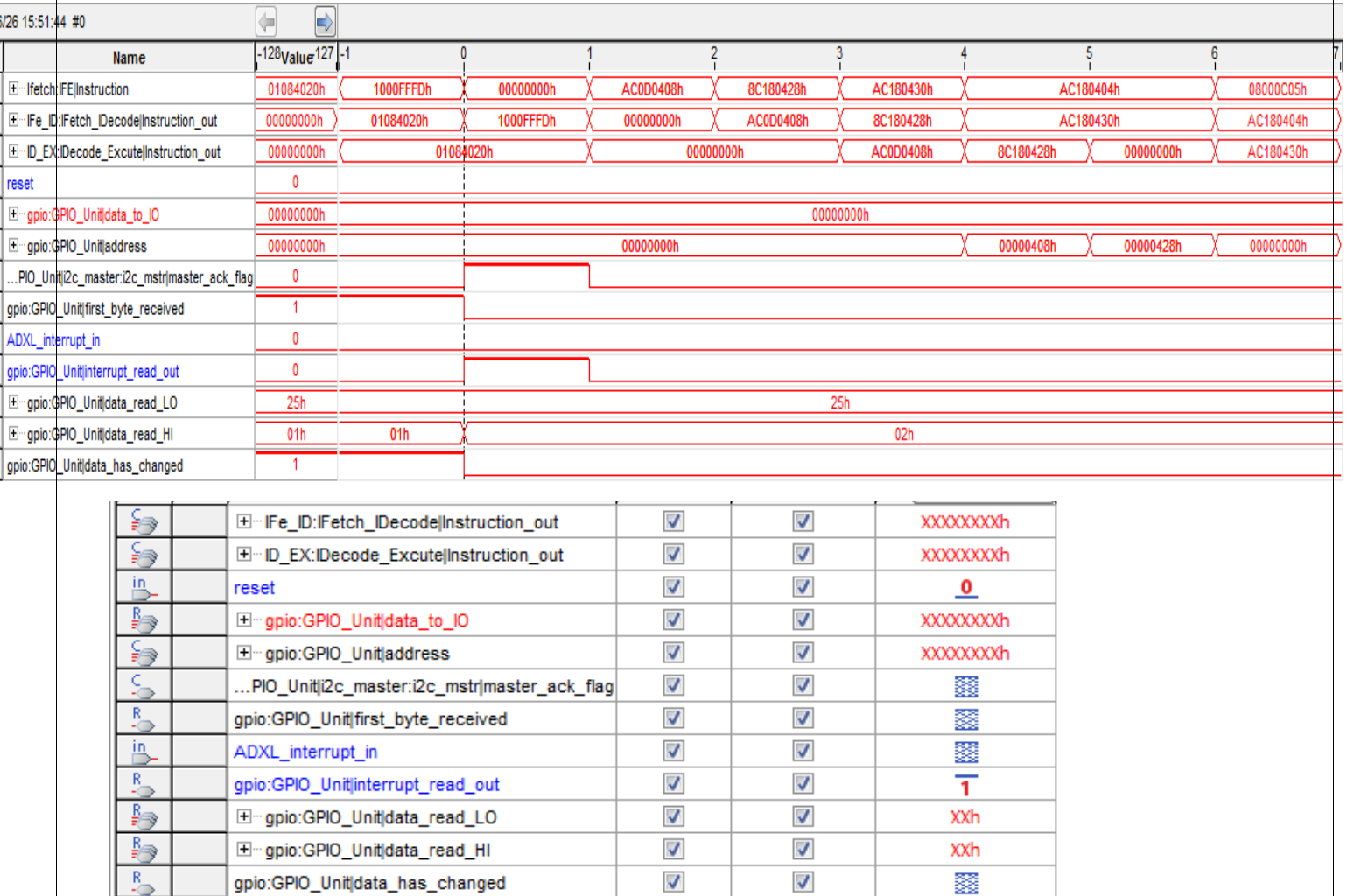


Figure 19

b. Commands executed before the interrupt:

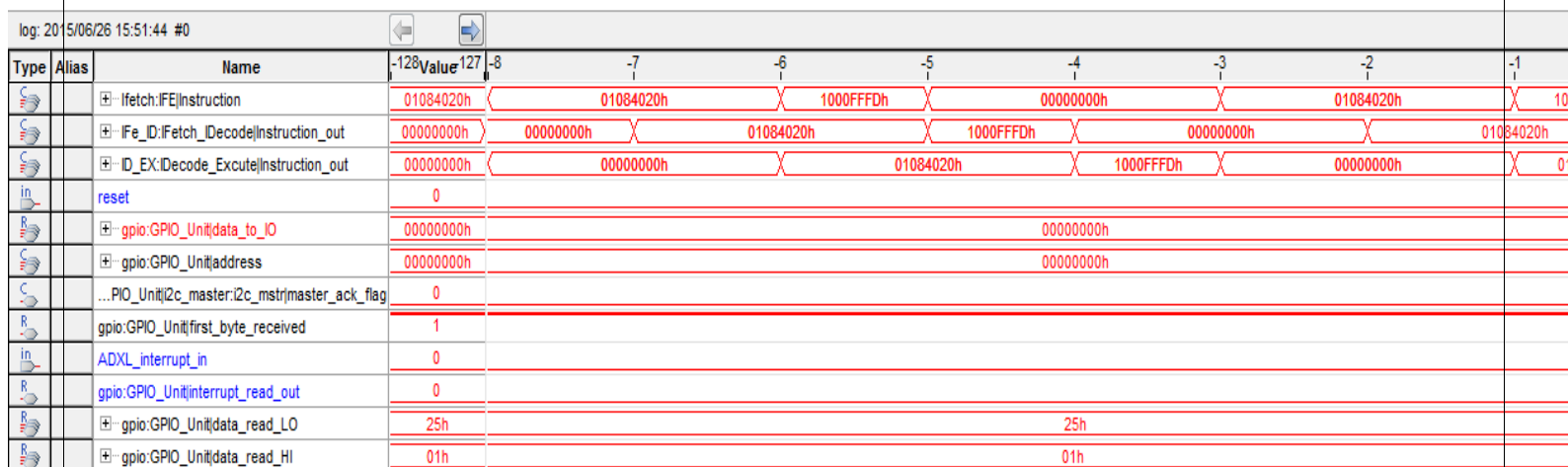
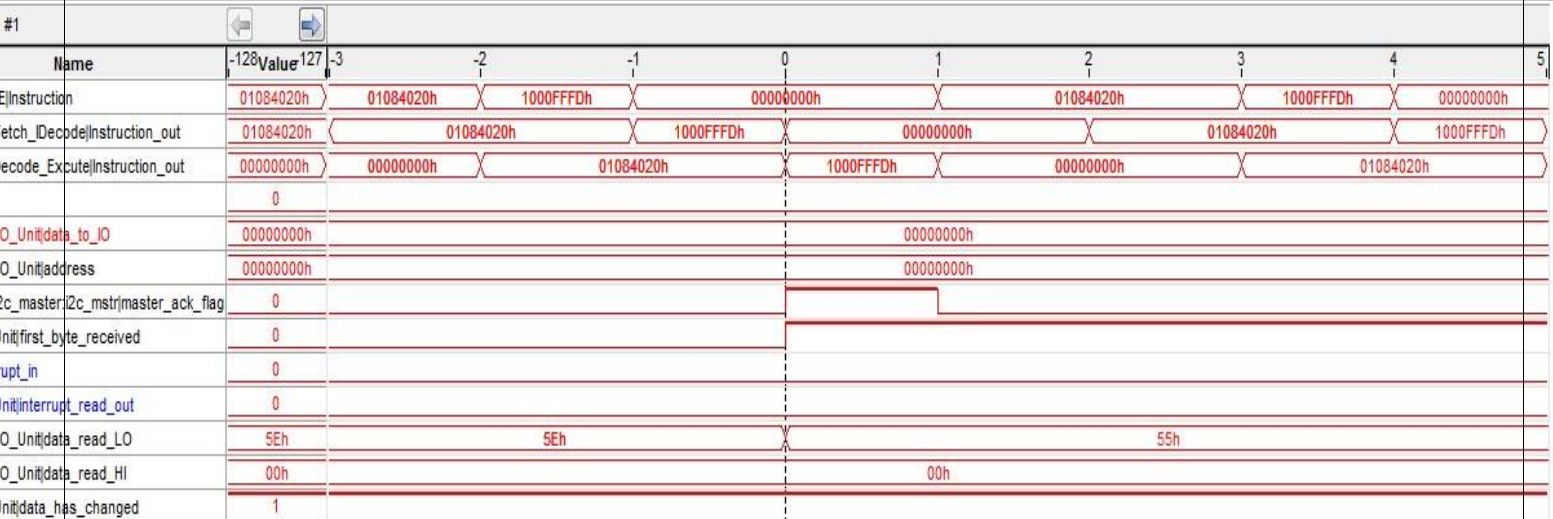


Figure 20

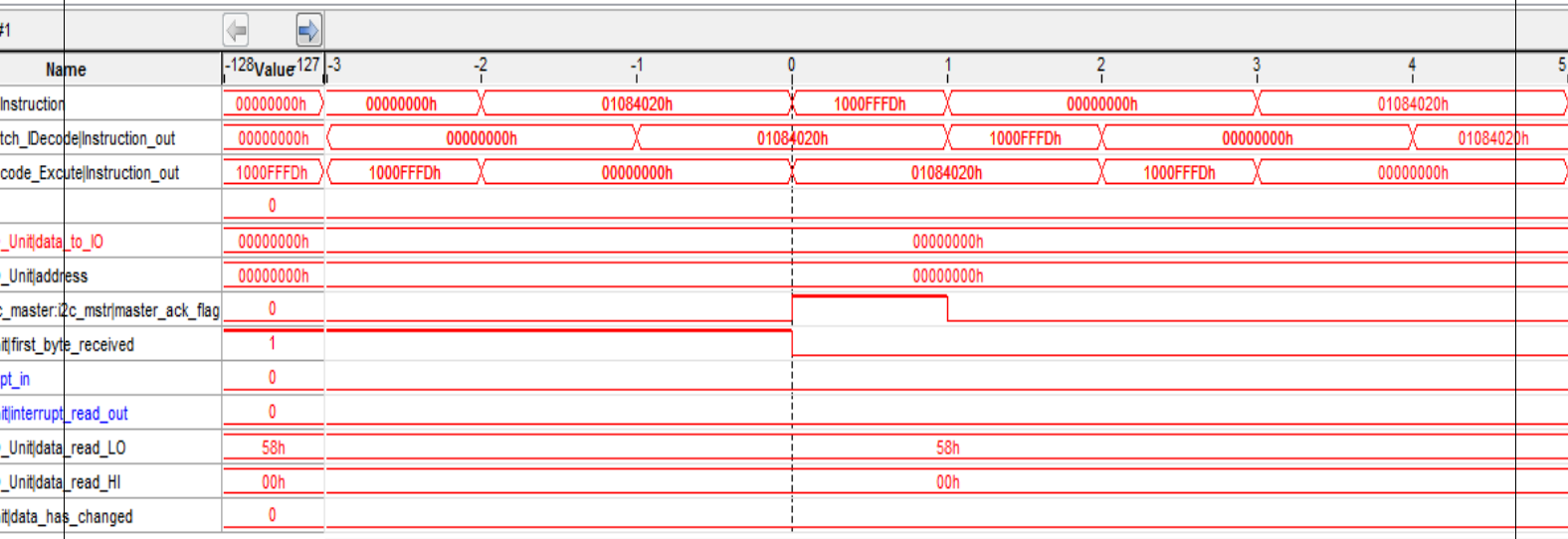
- c. First byte was received (Interrupt is not sent yet, but Data read LO is changed)



trigger: 2015/06/26 15:44:28 #0				Lock mode: Allow all changes	
Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			182	182	1 Basic
		IFetch:IFE Instruction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		IFe_ID:IFetch_IDecode Instruction_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		ID_EX:IDecode_Excute Instruction_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		reset	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		gpio:GPIO_Unit data_to_IO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		gpio:GPIO_Unit address	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		...PIO_Unit i2c_master:i2c_mstr master_ack_flag	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		gpio:GPIO_Unit first_byte_received	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		ADXL_interrupt_in	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		gpio:GPIO_Unit interrupt_read_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		gpio:GPIO_Unit data_read_LO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXh
		gpio:GPIO_Unit data_read_HI	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXh
		gpio:GPIO_Unit data_has_changed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1

Figure 21

d. Second byte received but the data didn't change: (No interrupt was sent).



trigger: 2015/06/26 16:00:25 #0			Lock mode: Allow all changes		
Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
			182	182	1 Basic
		IFetch:IFE Instruction	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		IFe_ID:IFetch_IDe Instruction_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		ID_EX:IDe Instruction_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		reset	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0
		gpio:GPIO_Unit data_to_IO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		gpio:GPIO_Unit address	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XXXXXXXXh
		...PIO_Unit i2c_master:i2c_mstr master_ack_flag	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1
		gpio:GPIO_Unit first_byte_received	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		ADXL_interrupt_in	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		gpio:GPIO_Unit interrupt_read_out	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
		gpio:GPIO_Unit data_read_LO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	xxh
		gpio:GPIO_Unit data_read_HI	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	xxh
		gpio:GPIO_Unit data_has_changed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Figure 22

7. Interrupt Mechanism MIPS side

Once an interrupt flag is raised for a one clock cycle, we must handle it in the fetch phase.

The basic idea, is to save the current PC in a register, and put in the Next PC an address of the interrupt service routine. (Ours is 0x04). However, life is tough, and there might be a branch command somewhere in the MIPS, which sends the Next PC to a different address.

So what do we do? The following are the key actions we take:

- Maintain the PC plus 4 address in Execute Phase to a register
- Flush 2 commands
 - Flushing the commands eliminates the hazards of branch in Decode and Fetch phases, as they disappear.
- Jump to a service routine PC address (0x04)
- Implement a ret_i command (op 000100)

What happens if the Branch in Execute, Memory or Write back phases?

- Interrupt while Branch in Execute Phase
 - Maintain the Next PC that would have been unless the interrupt occurred
- Interrupt while Branch in Memory Phase
 - The command in Fetch will be flushed. We need it.
 - Maintain the current PC in Fetch phase
- Interrupt while Branch in Write Back Phase
 - The command in Decode will be flushed. We need it.
 - Maintain the current PC in Decode phase

The code in IFetch.vhd looks like this:

```

Next_PC <= "00000000100" when (IE = '1' and I2CReadIF = '1')
        ELSE PC_RET_REG when Ret_i = '1'
        ELSE Next_PC_No_Interrputs;

PROCESS
BEGIN
    WAIT UNTIL ( clock'EVENT ) AND ( clock = '1' );
    IF reset = '1' THEN
        PC <= "00000000000" ;
    ELSIF IE = '1' AND I2CReadIF = '1' THEN
        IF Flush = '1' THEN
            PC_RET_REG <= Next_PC_No_Interrputs;
        ELSIF (PCOfCommandInEX = "00000000000" and PCOfCommandInID = "00000000000") THEN
            PC_RET_REG <= PC;
        ELSIF (PCOfCommandInEX = "00000000000") THEN
            PC_RET_REG <= PC_plus_4;
        ELSE
            PC_RET_REG <= PCOfCommandInEX;
        END IF;
        PC <= Next_PC;
    ELSE
        PC <= Next_PC;
    END IF;
END PROCESS;

```

Code 3

8. Conclusion

We had a very comprehensive project. We went through a lot of stuff. Communication, MIPS architecture, CPU interrupts, Filters and monitor communication (VGA protocol).

We touched almost every aspect of the whole system.

If we had more time, we could improve the access to some settings of the ADXL more easily (push buttons or switches) or even display 3D results on the monitor (from every axes).

END