MCU Architecture

Final Project Assignment Definition

Hanan Ribo

07.07.22

Table of contents

1. Aim of the project

- Design, synthesis and analysis of a simple (single cycle architecture) MIPS CPU core with Memory Mapped I/O, interrupt capability and ***Serial communication peripheral (as bonus)***
- Understanding of CPU vs. MCU concept.
- Understanding in FPGA memory structure.

2. Definition and prior knowledge

The aim of this project is to design CPU MIPS based MCU. The CPU will use a Single Cycle MIPS architecture and must be capable of performing full instruction set of simple MIPS (given as appendix). The design will be located on Altera Board. The MIPS architecture is Harvard architecture in order to increase throughput and simplify the logic. For additional information regarding MIPS CPU, Architecture, ISA and instructions see MIPS technical documents [1].

3. Assignment definition

The architecture must include a MIPS ISA compatible CPU with data and program memory Caches for hosting data and code. The block diagram of the architecture is given in Figure 1. The CPU will have a standard MIPS register file. The top level and the MIPS core must be structural. The design must be compiled and loaded to the Altera board for testing. A single clock (CLK) should be used in the design.

Note: use push-button **KEY0 as a System RESET** (*see page 10* - brings PC to the first program command) and **SW9 as PC ENA**.

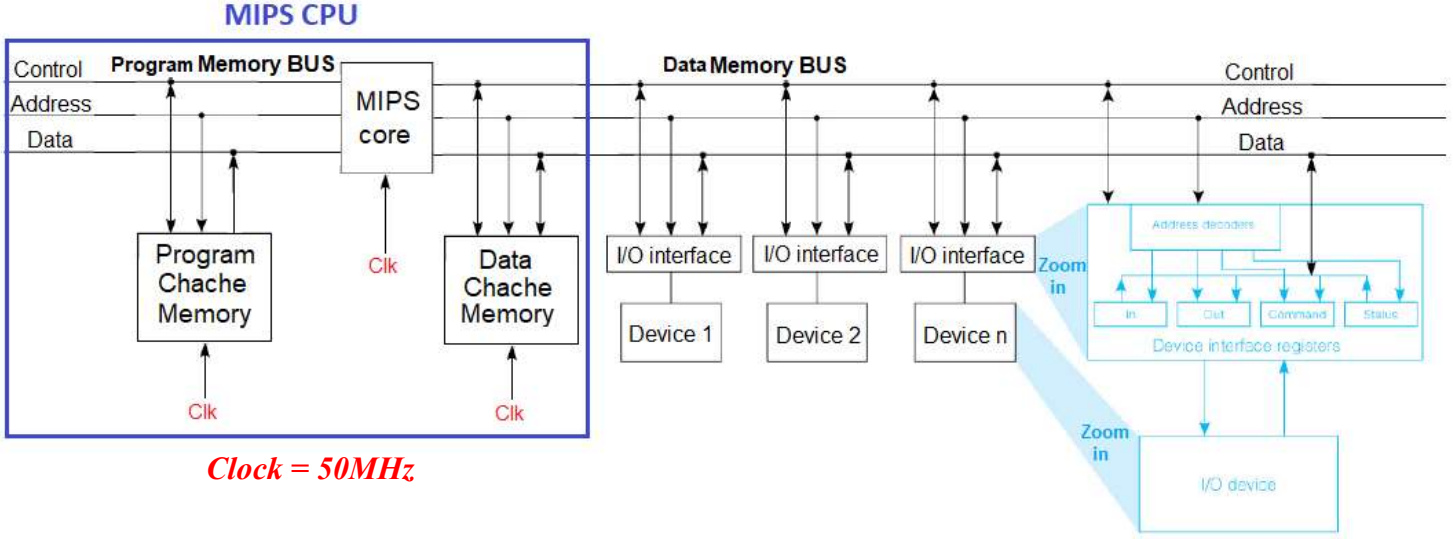
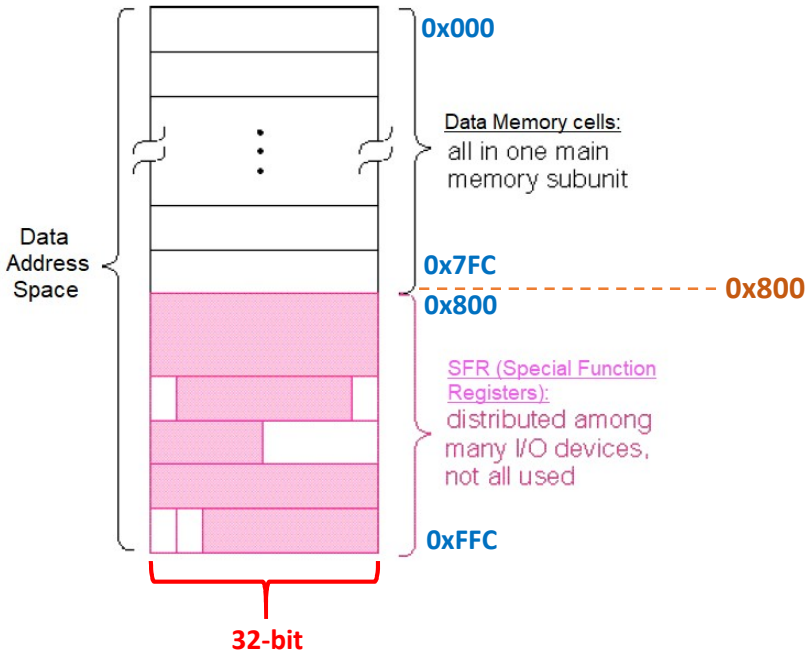


Figure 1 : System architecture

- 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 example that enables the CPU to output data to LEDs and 7-Segment and to read the Switches state.



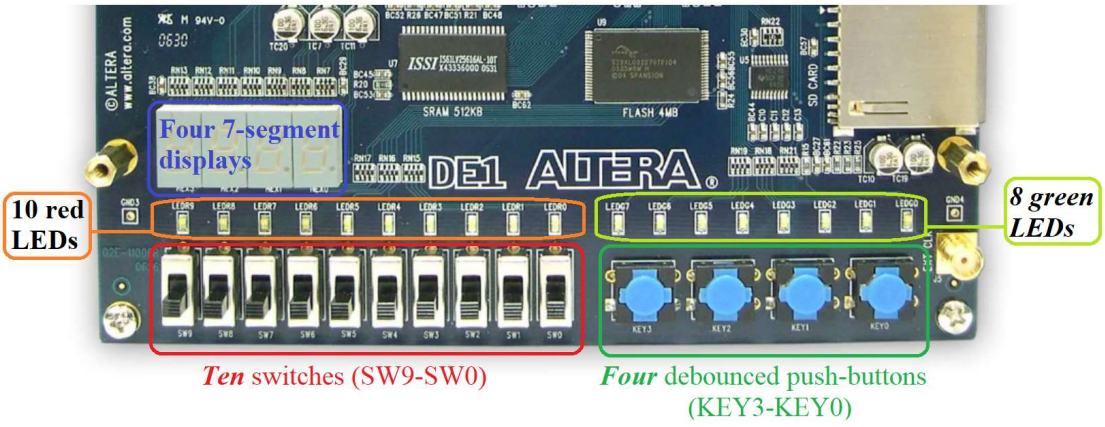
The Data Address Space is 32-bit WORD aligned where the address word is $\mathbf{0} \ldots \mathbf{0}\boldsymbol{A_{11}} \ldots \boldsymbol{A_0}$ with partial mapping.

Figure 2: Data Address Space contains Data Memory and Memory Mapped I/O

I/O devices connected:

In the hardware test case you will have to use at the FPGA board.

- Board ***ten*** switches (SW9-SW0) and ***four*** debounced push-buttons (KEY3-KEY0) will be used as ***Input interface***.
- Board ***ten*** red LEDs (LEDR9-LEDR0), ***eight*** green LEDs (LEDG7-LEDG0) and ***four*** 7-segment displays used as ***Output interface***.




```
#-------------------- MEMORY Mapped I/O ----------------
#define PORT_LEDG[7-0] 0x800 - LSB byte (Output Mode)
#define PORT_LEDR[7-0] 0x804 - LSB byte (Output Mode)
#define PORT_HEX0[7-0] 0x808 - LSB byte (Output Mode)
#define PORT_HEX1[7-0] 0x80C - LSB byte (Output Mode)
#define PORT_HEX2[7-0] 0x810 - LSB byte (Output Mode)
#define PORT_HEX3[7-0] 0x814 - LSB byte (Output Mode)
#define PORT_SW[7-0]   0x818 - LSB byte (Input Mode)
#-------------------------------------------------------
```

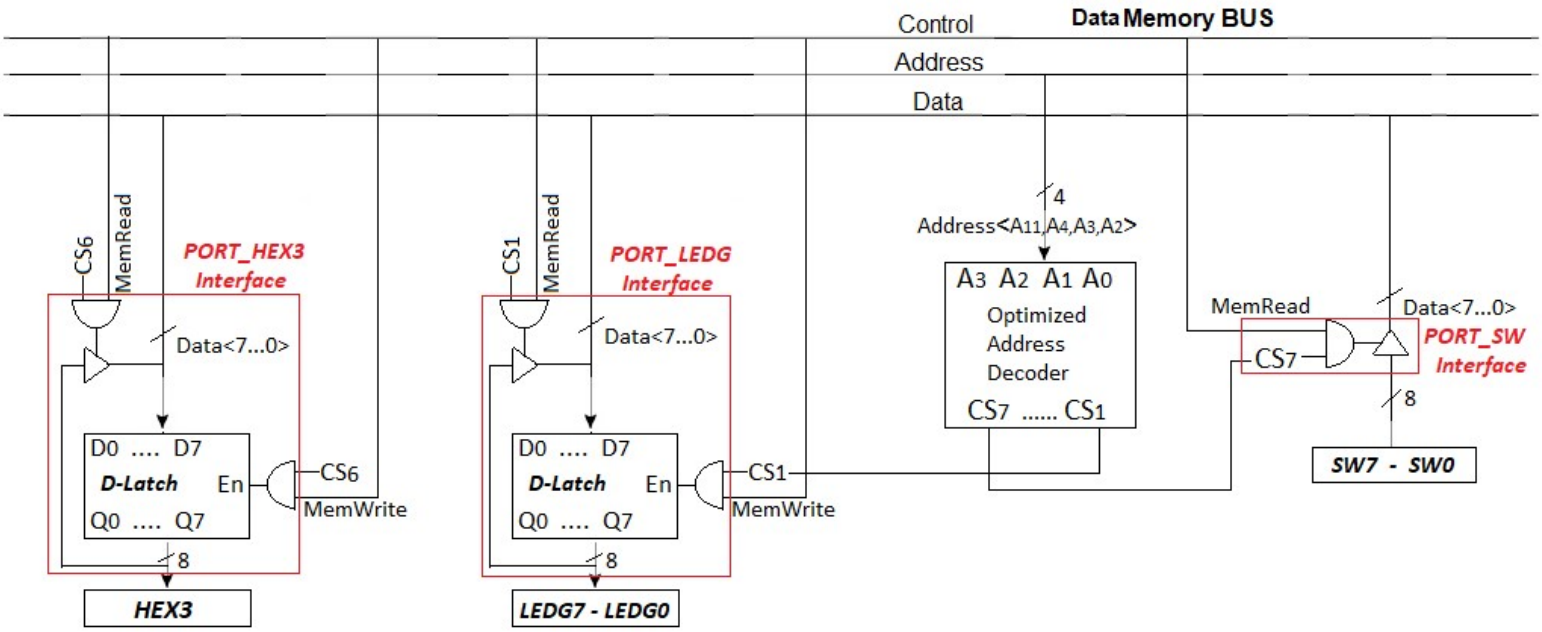


Figure 3: <u>Primitive</u> GPIO peripheral connection using Memory Mapped I/O approach

- The CPU will be based the *standard 32bit MIPS ISA* and the Instructions will be 32 bit wide. The following table shows the MIPS instruction format. For more information, see MIPS technical documents [1].

Type	**-31-**			**format (bits)**		**-0-**
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Table 1 : MIPS Instruction format

The Data address space is 4kB. Memory latency will be according to Table 2

Memory	**Write Latency**	**Read Latency**
Program Memory (I-Cache)	1 clk	1 clk
Data Memory (D-Cache)	1 clk	1 clk

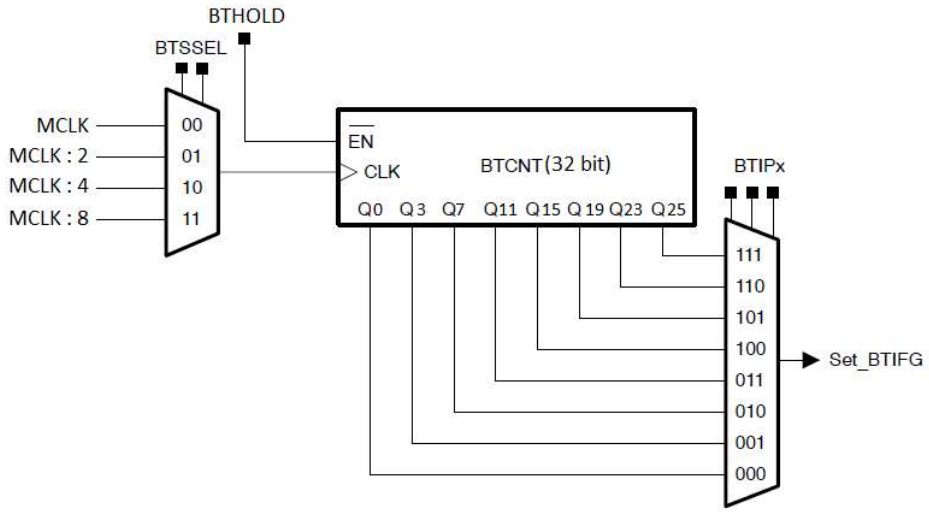
Table 2 : Memory sizes and latency

4. Required Support of CPU Peripherals

i. **Seven GPIO ports** (six Output and one Input) for peripherals depicted in page 5

ii. **KEY[3-1]:** support ***three*** array push-buttons as input device.

iii. **Basic Timer:**



BTCTL, Basic Timer Control Register

7	6	5	4	3	2	1	0
		BTHOLD	BTSSEL			BTIPx	
rw	rw	rw	rw	rw	rw	rw	rw

BTCNT, Basic Timer Counter

31	30	. . . . .	1	0
BTCNT				
rw	rw		rw	rw

iv. USART Peripheral Interface, UART Mode (Bonus 20%):

The required communication peripheral is the universal **USART** (synchronous/asynchronous receive/transmit) peripheral interface in **UART** Mode only (degenerated **USART**).

You are given VHDL design code that need to be adapted to the next UART mode features.

UART mode features include:

- 1-start bit, 1-stop-bit, 8-bit data with non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first data transmit and receive
- Programmable baud rate support
- Status flags for error detection
- Independent interrupt capability for receive and transmit

UCTL, USART Control Register

7	6	5	4	3	2	1	0
BUSY	OE	PE	FE	BAUDRATE	PEV	PENA	SWRST
r	r	r	r	rw	rw	rw	rw

SWRST Bit 0 Software reset enable
0 Disabled. USART reset released for operation
1 Enabled. USART logic held in reset state

PENA Bit 1 Parity enable
0 Parity disabled
1 Parity enabled. Parity bit is generated (TXD) and expected (RXD).

PEV Bit 2 Parity select. PEV is not used when parity is disabled.
0 Odd parity
1 Even parity

BAUDRATE Bit 3 Baud Rate value
0 9600
1 115200

FE Bit 4 Framing error flag
0 No error
1 Character received with low stop bit

PE Bit 5 Parity error flag. When PENA = 0, PE is read as 0.
0 No error
1 Character received with parity error

OE Bit 6 Overrun error flag. This bit is set when a character is transferred into UxRXBUF before the previous character was read.
0 No error
1 Overrun error occurred

BUSY Bit 7 This bit indicates if a transmit or receive operation is in progress (busy).
0 UART module inactive
1 UART module transmitting or receiving

RXBUF, USART Receive Buffer Register

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
r	r	r	r	r	r	r	r

RXBUFx	Bits 7-0	The receive-data buffer is user accessible and contains the last received character from the receive shift register. Reading RXBUF resets the receive-error bits, and RXIFG.

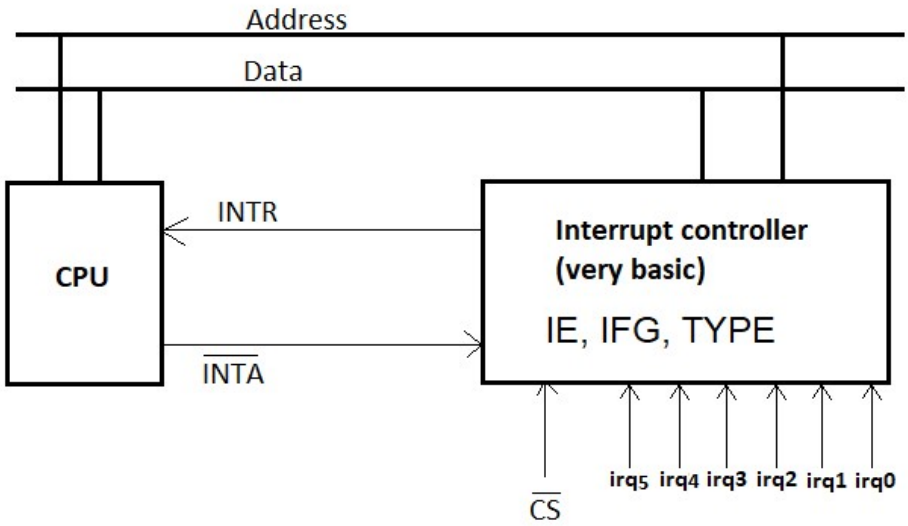
TXBUF, USART Transmit Buffer Register

7	6	5	4	3	2	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
rw	rw	rw	rw	rw	rw	rw	rw

TXBUFx	Bits 7-0	The transmit data buffer is user accessible and holds the data waiting to be moved into the transmit shift register and transmitted on TXD. Writing to the transmit data buffer clears TXIFG.

Note: for UART module reference, see block diagram of MCUs that use acquainted with.

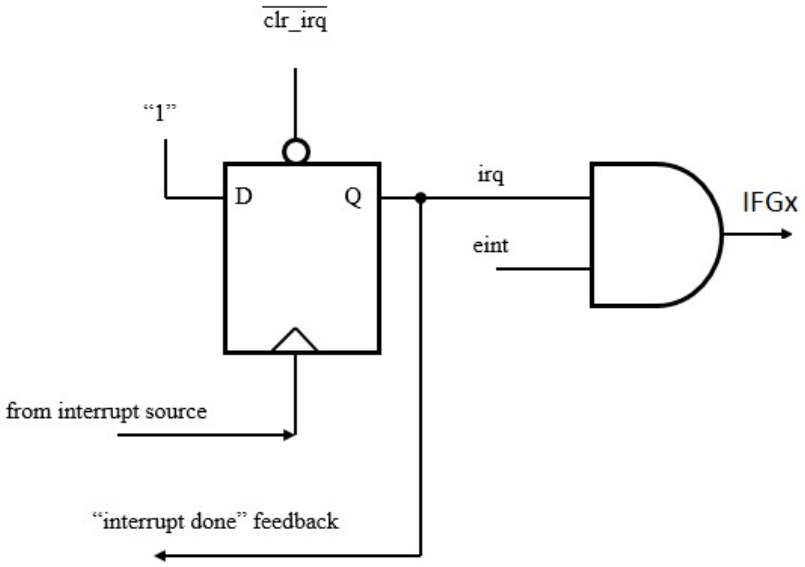
v. **Interrupt controller:**



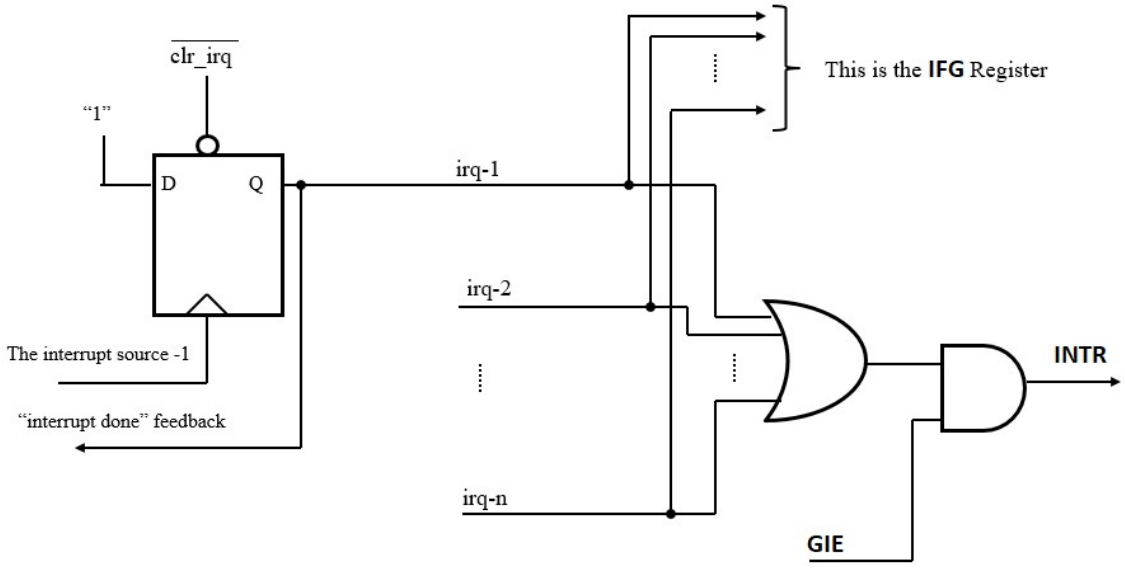
Not

i. The **BTIFG** flag is reset automatically when the interrupt is serviced.
ii. **RXIFG** is automatically reset if the pending interrupt is served or when **RXBUF** is read.
iii. **TXIFG** is automatically reset if the interrupt request is serviced or if a character is written to **TXBUF**
iv. The **KEYiIFG** is reset manually with software (**BTIFG, RXIFG, TXIFG** as well).
v. As part of CPU services an interrupt, **GIE** is clear (***in HW***) means DINT of other interrupts. Symmetrically, as part of CPU returning from interrupt, **GIE** is set (***in HW***) means EINT of interrupts (back the origin state).

Handling an interrupt:



Handling interrupts from several sources:



IE, Interrupt Enable Register

7	6	5	4	3	2	1	0
0	0	KEY3IE	KEY2IE	KEY1IE	BTIE	TXIE	RXIE
r-0	r-0	rw	rw	rw	rw	rw	rw

IEx Bit x 0 Interrupt not enabled
1 Interrupt enabled

IFG, Interrupt Flag Register

7	6	5	4	3	2	1	0
0	0	KEY3IFG	KEY2IFG	KEY1IFG	BTIFG	TXIFG	RXIFG
r-0	r-0	rw	rw	rw	rw	rw	rw

IFGx Bit x 0 No interrupt pending
1 Interrupt pending

TYPE, Interrupt Type Register

7	6	5	4	3	2	1	0
0	0	0	TYPEx				0
r	r	r	r	r	r	r	r

TYPE Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	RESET	NMI	Highest
04h	UART status error	-	
08h	UART RX	RXIFG	
0Ch	UART TX	TXIFG	
10h	Basic Timer	BTIFG	
14h	KEY1	KEY1IFG	
18h	KEY2	KEY2IFG	
1Ch	KEY3	KEY3IFG	Lowest

(Non)-Maskable Interrupt

Maskable Interrupt

GPIO ***without*** *interrupt capability*

Peripherals ***with*** *interrupt capability*

```
#-------------------- MEMORY Mapped I/O ----------------------
#define PORT_LEDG[7-0] 0x800 - LSB byte (Output Mode)
#define PORT_LEDR[7-0] 0x804 - LSB byte (Output Mode)
#define PORT_HEX0[7-0] 0x808 - LSB byte (Output Mode)
#define PORT_HEX1[7-0] 0x80C - LSB byte (Output Mode)
#define PORT_HEX2[7-0] 0x810 - LSB byte (Output Mode)
#define PORT_HEX3[7-0] 0x814 - LSB byte (Output Mode)
#define PORT_SW[7-0]   0x818 - LSB byte (Input Mode)
#define PORT_KEY[3-1]  0x81C - LSB nibble (3 push-buttons - Input Mode)
#-------------------------------------------------------------
#define UCTL           0x820 - Byte
#define RXBF           0x821 - Byte
#define TXBF           0x822 - Byte
#-------------------------------------------------------------
#define BTCTL          0x824 - LSB byte
#define BTCNT          0x828 - Word
#-------------------------------------------------------------
#define IE             0x82C - LSB byte
#define IFG            0x82D - LSB byte
#define TYPE           0x82E - LSB byte
```

5. Pin Planner

Only MCU IO devices need to be connected to FPGA location legs via pin planner. Location legs that used for proof of work phase (signal tap) need to be removed at the final step.

6. Host Interface (to L1 caches)

After the last system developing stage, you will be given a made JTAG based code wrapper of communication interface to L1 memory caches (data and program) in order to upload/download their content without reloading the system hardware design onto the FPGA chip.

7. Compiler, Simulator and Memory

The MARS compiler and simulator, or any other can be used to compile and simulate the assembly code. MARS compiler can also export the memory contents into the file in format that VHDL can easily read. It can also simulate a cache performance.

The mars compiler, installation instructions and documentation are available at: http://courses.missouristate.edu/KenVollmar/MARS/

8. CPU and MCU Test

a) ***Mandatory:*** supporting all of given ***test$_i$.asm*** *assembly* source files.

b) ***Bonus (= under condition of working properly):*** Using serial communication support application of PC side (as ***Hyper-Terminal***, ***Tera-Term***, ***puTTY*** etc) and write code for MCU side that support the next menu (transmitted from MCU to PC):

Menu

1. Count up from 0x00 onto LEDG with delay ~0.5sec
2. Count down from 0xFF onto LEDR with delay ~0.5sec
3. Clear all LEDs
4. On each KEY1 pressed, send the massage "I love my Negev"
5. Show Menu

9. MCU and PC communication using RS-232 interface (= Bonus)

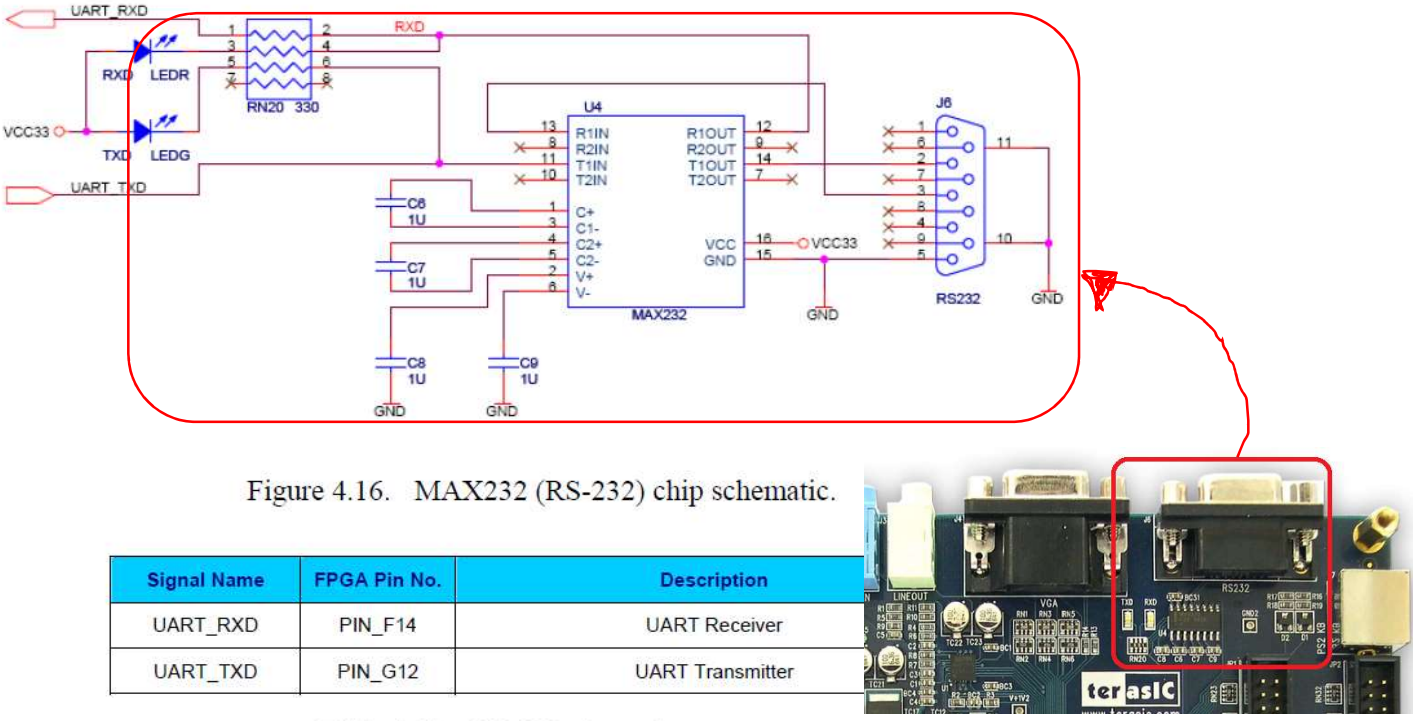


Figure 4.16. MAX232 (RS-232) chip schematic.

Signal Name	FPGA Pin No.	Description
UART_RXD	PIN_F14	UART Receiver
UART_TXD	PIN_G12	UART Transmitter

Table 4.10. RS-232 pin assignments.

10. Requirements

You have to do the following tasks:

- ModelSim Simulation with maximal coverage.
- Analyze the critical path, explain where it is in your VHDL design and find the maximal operating clock.
- Load the design onto the FPGA and verify the simulation results.
- **Run the required assembly source codes and explore them.**

The following must be presented in **final.pdf** report file.

1. Top level block review diagram of your design.
2. For each block in the top level design:
   - RTL Viewer results
   - Logic usage for each block (Combinational and Flip-Flops).
   - Graphical description (a square with ports going in and out).
   - Port Table (direction, size, functionality).
   - Short description.
3. Maximum (Critical) path of your design – explain where it is in the code and how it is possible to optimize if you would have more time. What is the maximum clock frequency?
4. Minimum path analysis.
5. Documentation Style - Content with page numbers, Images and tables will be numbered. The caption of an images and tables below the images or tables.
6. Elaborated analysis and wave forms:
   - Maximal Frequency and critical paths from Timing Analyzer.
   - Proof of work using Signal Tap shot screens.
     Recall that, proof of work using Signal Tap is mandatory.
   - **One** basic waveform to explain the system timing.

Design requirements:

1. The design must be well commented.
2. The system must work from only one clock.
3. System RESET (KEY0) must be synchronous.
4. Conclusions
5. A ZIP file in the form of **id1_id2.zip** (where id1 and id2 are the identification number of the submitters, and id1 < id2) *must be upload to Moodle only by student with id1* (any of these rules violation disqualifies the task submission).

6. The **ZIP** file will contain the next six subdirectories (***only the exact next sub folders***):

Directory	Contains	Comments
DUT	Project VHDL / Verilog HDL files (***you must use only a single version of DUT files which are adapted to ModelSim and Quartus IDEs under method of conditional compilation using generic map of Boolean parameter***)	Only VHDL / Verilog HDL files, ***excluding test bench*** **Note: your project files must be well compiled (in ModelSim and Quartus separately) without errors as a basic condition before submission**
TB	VHDL files that are used for test bench	Only one **tb.vhd** for the overall DUT
SIM	ModelSim DO files	Only for **tb.vhd** of the overall DUT
DOC	Project documentation	**Readme.txt** and **final.pdf** full report file
Quartus	• Signal Tap files used in project verification • Project SOF file • Project SDC file	Do not place files that are not relevant for compilation or is a result of compilation!
CODE	The assembly source code of clause 6b	

Table 3 : Directory Structure

11. Grading policy

Weight	Task	Description
10%	Documentation	The "clear" way in which you presented the requirements and the analysis and conclusions on the work you've done
90%	Analysis and Test	The correct analysis of the system (under the requirements)

Table 4: Grading

Late submissions will be not gotten.

12. References

[1]. MIPS32® Architecture for Programmers Volume I to III (from Moodle under Final Project)
[2]. ALTPLL User Guide: http://www.altera.com/literature/ug/ug_altpll.pdf
[3]. Altera RAM user guide: http://www.altera.com/literature/ug/ug_ram_rom.pdf
[4]. Altera MegaFunction User Guide: www.altera.com/literature/ug/ug_intro_to_megafunctions.pdf
[5]. Bin2Hex utility
32bit - http://www.keil.com/download/docs/113.asp
64bit - http://www.ht-lab.com/freeutils/bin2hex/bin2hex.html