EE314 TERM PROJECT

Introduction:

• Note that this is not a weekend project. Start working on it now. If you would like to test your designs you can use the equipment in the EE314 lab in working hours unless there is a laboratory session proceeding. During weekends, laboratory will be closed.

• The aim of this project work is to make you more familiar with some subjects you were introduced in EE312 and EE348 courses. However, you may need to do some research and study extra material to accomplish the task. This will be a good first step for 4th year graduation projects.

• The project groups will contain at most 2 students. Although it is not recommended, you may do your project alone. So, determine your project partner as soon as possible. It is not necessary that your lab partner and project partner is the same person.

• You are free and encouraged to use your own ideas. Although your design approach is not limited, the systems are supposed to be economical. You are not allowed to use VHDL or schematic design. You should implement your projects using Verilog.

•You are supposed to use FPGA development boards provided in the laboratory. FPGA implementation of the project is required.

• All assistants are responsible for the project. Primary contact mechanism with the assistants is via email.

•No early demonstration will be allowed (apart from the crucial reasons, such as Erasmus, foreign student, etc.).

Important Dates:

-22th May 23.59: Proposal Report

-11-12th June: Project Demonstrations

-13th June 17.00: Final Report Submissions

-13th June 17.00: Video Submission

Report Format

Proposal Report: The aim of the proposal report is for you to start your research early on so that you can have a solid idea about the project. This report will contain preliminary work on your project. A good report should include your proposed way to solve the problem, the equipment required for the solution, some block diagrams of the overall system and any additional info (circuit schematics, example codes, simulations etc.) you see fit. Maximum page limit for the preliminary report is 3 pages (Times New Roman, 10 point font). Longer reports will be rejected. It is crucial that you determine your project partner, and do some brain storming to come out with solutions well before the preliminary report deadline. You have to upload your proposal report in pdf format to ODTUCLASS until 16th of May, 23:59. Late submissions will not be accepted.

Final Report: The final report should be in the IEEE double column paper format (please check the IEEE paper format) and it should not exceed 15 pages in total, any more pages will decrease your grade. The formatting is one of the most important parts of the project. If the final report is not in the IEEE paper format, the project will not be graded and you will get zero from the whole project. Any formatting mistake (such as no figure captions, not referral to the figure in your main text, etc.) will result in grade deduction. You have to upload your proposal report in pdf format to ODTUCLASS until 13th of June, 17:00. Late submissions will not be accepted. Your report should include the following items:

- Theoretical background and literature research
- Design methodology of the subsystems
- Simulation results verifying that your subsystems and overall system is working properly
- Experimental results
- Comparison of the experimental results with the simulation results.

Grading:

Proposal Report: 7%

Demonstration: 50%

Final Report: 40%

Video: 3%

Bonus (Using the ADC on the FPGA development board): 5%

DESIGN OF A SIMPLE MICROCONTROLLER AND A LIGHT INTENSITY MONITORING SCREEN

In this project, you are expected to design a basic ALU which is the core of a microprocessor. Using this simple microcontroller, you are supposed to read resistance change of an LDR and display the environmental light on a VGA screen (not necessarily with true units). The project details are given below:

BACKGROUND INFORMATION ABOUT MICROCONTROLLER AND ALU

A microcontroller consists of CPU (Central Processing Unit), memory and I/O system. The CPU, the core of the microcontroller is also called as microprocessor. It interprets the instructions fetched from the memory and executes them. Internal operations of microcontroller are synchronized with a stable clock, which may be produced by a quartz crystal. The microcontroller fetches the instructions from the memory, decodes them and executes them using ALU (Arithmetic Logic Unit) and registers. A microcontroller also needs a ROM to save a program permanently and a RAM to save data temporarily. Using I/O resources microcontroller communicates with outside world. Basic structure of a microcontroller is given in Figure 1.

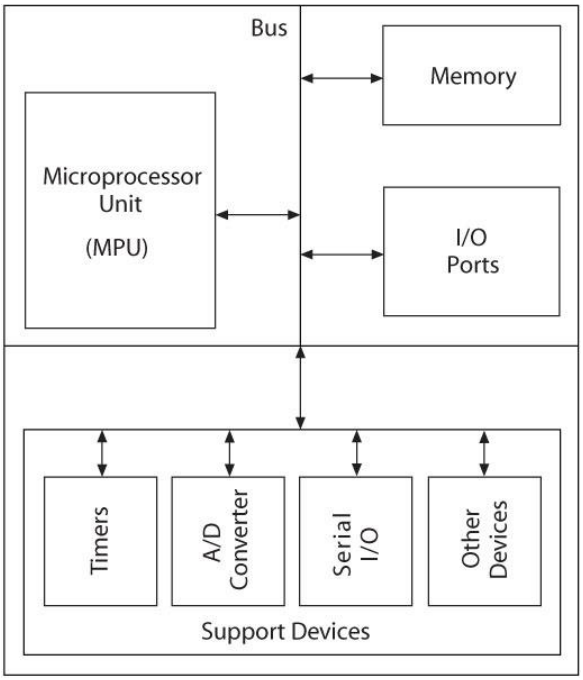


Figure 1: Basic Structure of a Microcontroller

Memory Units

Read Only Memory (ROM): It is a memory unit that is used to store the program permanently. The size of the program that can be written depends on the size of the memory unit. ROM can be built in the microcontroller or can be connected externally to microcontroller.

Random Access Memory (RAM): It is used for temporary storing data and intermediate results during the operation. When the power is turned off, the content of the RAM is cleared.

Electrically Programmable ROM (EEPROM): It is a special type of memory. Its content may be changed while the microcontroller is working but remains permanently after the power is turned off. It is used to store values which are created during operation and which must be saved after the power is turned off.

Central Processing Unit (CPU)

It is the core of the microcontroller and it controls and monitors all processes within the microcontroller. User cannot reach this part of the microcontroller. It contains several subunits some of which are instruction decoder which recognizes program instructions and runs other circuits on the basis of that, Arithmetic Logic Unit (ALU) performs all mathematical and logical operations and Accumulator is a CPU register closely working operation of ALU which stores all data upon which some operations should be executed. Main structure of a CPU is given in Figure 2.

CPU Registers: This is the local storage space known as register performing most of the operations that microcontroller cannot process directly. Any kind of data is needed first to be identified by the register before manipulated by the processor. In this project number and type of CPU registers are left to the designer. However, there are some registers which they are certainly needed by CPU. For example the **"accumulator"** holds the data currently being processed by the CPU. **Program Counter (PC)** points to memory address containing the next instruction to be executed. After each instruction program counter increases by one. However the value of the program counter may be changed at any moment which causes a jump to a new memory location. Subroutines and branch instructions are done by this way. After jump, it continues its usual behavior.

Arithmetic Logic Unit (ALU) is the core of a CPU. All arithmetic and logic operations are performed by ALU. It is composed of combinational and sequential circuits. It performs operations such as addition, subtraction, multiplication, AND, OR.

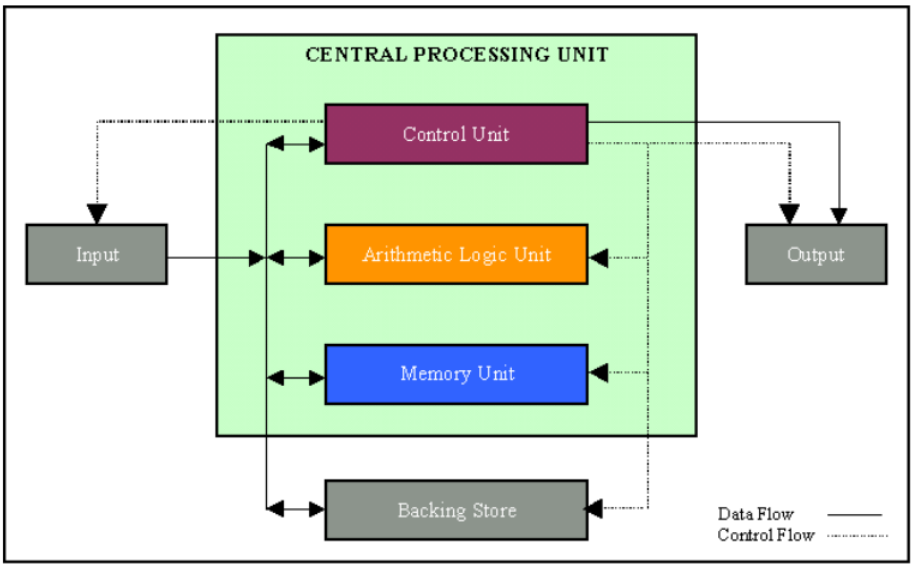


Figure 2: Main Structure of a CPU

Phase 1: Design of the Basic Microcontroller

You will design a microcontroller with the given specifications using Verilog. You are supposed to show your microcontroller works properly using simulation tools and you are supposed to implement your microcontroller to the FPGA boards in our laboratory.

The block diagram of the first step of your project is given in Figure 3:

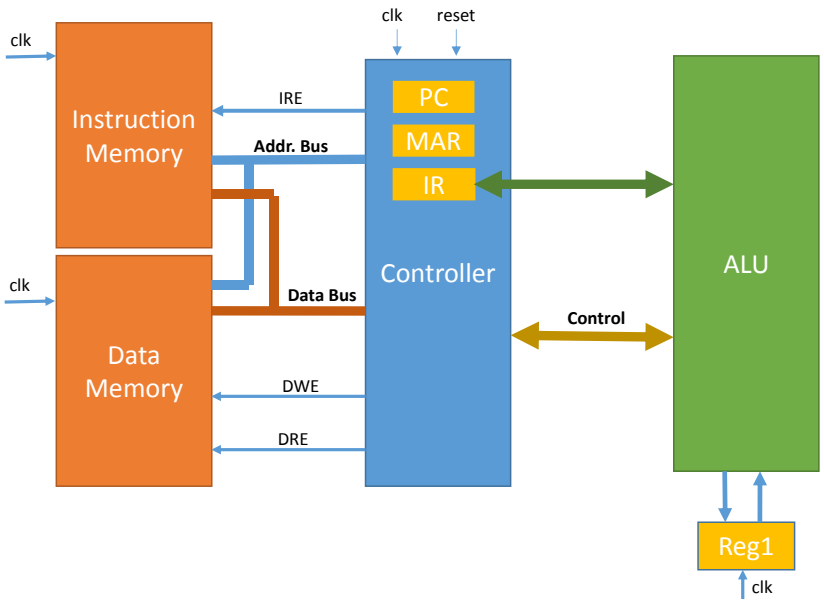


Figure 3: Block diagram of the basic microcontroller which will be designed

Main Blocks: This microcontroller consists of three systems basically: memory, controller and ALU

Memory: Required data and instructions are stored in the memory. Using basic flip flops will be enough for this project.

Instruction Memory: It is the memory block which instructions are stored. Instructions are read by the controller and commands are given to the ALU. The content of the instruction data is predefined by the user. You can define the content in a specific region of your Verilog code. The content of the instruction memory is determined by the sets of instructions used in the operation of the microcontroller and your algorithm of application.

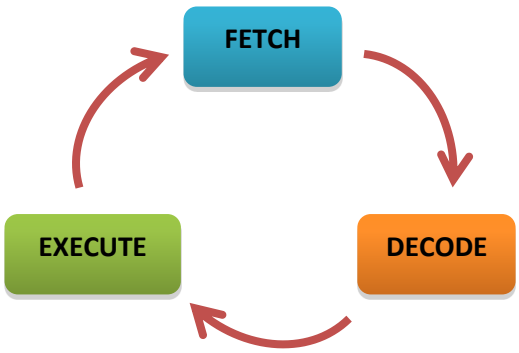
Data Memory: It stores the data taken from outside. ALU reads required data from this memory and uses it. Data taken from input ports are stored into the data memory. You can design your Data memory as a shift register to make write operation easier. However you should be able to read the stored data in any address of your data memory using microcontroller designed.

Controller: This unit will generate required signals between ALU and Memory units.

The controller reads the instructions from instruction memory unit and sends required signals to ALU. In other words it translates from machine instructions to the control signals that implement them.

It includes several registers such as PC (program counter), MAR (Memory Address Register), IR (Instruction register).

reg1(Accumulator): 16 bit register. It can be read or write enabled.



The instructions are processed in the “Fetch-Decode-Execute” cycles. In the “Fetch” cycle, the instruction is read from memory location pointed by the PC and written into the Instruction Register(IR). In “Decode” cycle, the “Opcode”(OPC) of the instruction in IR is decoded. If the OPR is a memory address, then the required data should be read from that memory location first. In Execute cycle, the operation OPC is executed using the operand OPR.

ALU(Arithmetic Logic Unit): The ALU is the basic part of the project where it makes the necessary operations by using the data stored in memory and the 16 bit register "reg1". The command bits are used to determine the operation that is going to be done by using the input data in and register "reg1". The list below summarizes the operations that are going to be included at the ALU block.

Operation	**Explanation**	**Instruction Format (Binary)**
Idle	Keep the registers same	00000
Increment_memory	Increment the data in the specified memory location by 1	00001XXXXXXX
Decrement_memory	Decrement the data in the specified memory location by 1	00010XXXXXXX
Load_reg1	Load data into the register "reg1" from the specified memory location	00011XXXXXXX
Add_reg1	Add data into the register "reg1" from the specified memory location	00101XXXXXXX
Logical Shift Left	Shift the data in register "reg1" by 1 bit left (Unsigned)	00110
Logical Shift Right	Shift the data in register "reg1" by 1 bit right (Unsigned)	00111
INVERT_reg1	Invert the bits of reg1	01010
AND_reg1	Logical AND operation of "reg1"(least significant 8 bits) and data in specified memory, store result in "reg1"	01011XXXXXXX
OR_reg1	Logical OR operation for "reg1"(least significant 8 bits) and data in specified memory, store result in "reg1"	01100XXXXXXX
XOR_reg1	Logical XOR operation for "reg1"(least significant 8 bits) and data in specified memory, store result in "reg1"	01101XXXXXXX
GOTO_address	Sets the program counter to a new address	01110XXXXXXX
Store_reg1	Stores the content of Reg1 into the specified memory location (Most significant 8 bit of "reg1" is stored to the specified memory location, Least significant 8 bit of "reg1" is stored to the (specified memory location+1).)	01111XXXXXXX
Skip_if_zero	Increments the program counter(skip next instruction) if the specified memory location is zero	10000XXXXXXX

Instruction Format

First five bit of the instruction is Opcode (OPC) and remaining seven bit is Operand (OPR). You can increase number of bits in OPR if needed. OPC represents the instruction code of the operation and OPR represents the data to be used in the operation. If the OPR is a memory address instead of data, then

the required data should be read from that memory location first. In Execute cycle, the operation OPC is executed using the operand OPR.

Each instruction lasts for 7 clock cycles. At 8th clock cycle, program counter increases its instruction address value by one and the next instruction is fetched.

In a basic microcontroller, each operation is performed by a set of microoperations in fetch and execute cycles:

1. **Fetch** an instruction from memory
2. **Decode** the instruction
3. **Execute** the instruction
4. After an instruction is executed, the cycle starts again at step 1, **for the next instruction.**

Example: Cycle by cycle operation of two examples are given below.

Ex. 1: Add_reg1

Cycle:		**Operation:**	**Explanation:**
Fetch	c_0:	MAR ← PC	Load program counter value into the Memory Address Register(MAR)
	c_1:	IR ← M(MAR)	Load data in memory into IR (Instruction Register)
Decode	c_2	OPC ← IR(OPC)	Load OPC register with the Opcode part of IR
Execute		if OPC== 00101	If the OPC is equal to 00101 then it is an add operation
	c_3:	MAR ← IR(OPR)	Load operand address into MAR
	c_4:	IR ← M(MAR)	Load data in memory location MAR into IR
	c_5:	AC ← AC+IR	Add data into the accumulator (reg1)
	c_6:	-	Empty cycle
	c_7:	PC ← PC+1	Increment Program Counter

Ex. 2: Increment_memory

Cycle:		**Operation:**	**Explanation:**
Fetch	c_0:	MAR ← PC	Load program counter value into the Memory Address Register(MAR)
	c_1:	IR ← M(MAR)	Load data in memory into IR (Instruction Register)
Decode	c_2	OPC ← IR(OPC)	Load OPC register with the Opcode part of IR
Execute		if OPC== 00001	If the OPC is equal to 00101 then it is an add operation
	c_3:	MAR ← IR(OPR)	Load operand address into MAR
	c_4:	IR ← M(MAR)	Load data in memory location MAR into IR
	c_5:	IR ← IR+1	Increment IR
	c_6:	M(MAR) ← IR	Store IR into the memory addres in MAR
	c_7:	PC ← PC+1	Increment Program Counter

Specifications

I/O Specifications

- *clk*: clock signal driving the microcontroller
- *IRE*: Read enable signal for instruction memory. When this signal is high, data is read from the instruction memory.
- *DWE:* Write enable signal for data memory. When this signal is high, microcontroller is ready for taking data from outside environment. This signal is controlled by a module outside of the microcontroller.
- *DRE*: Read enable signal for data memory. When this signal is high, data is read from the data memory.
- *reset:* Asynchronous reset signal. When this signal is activated, data in data memory is cleaned, microcontroller turns back to its initial state.

Phase 2: Light Intensity Monitoring Screen

In this phase of the project you are supposed to use the microcontroller to collect and process the data from your LDR. Analog data from LDR will be converted into digital data using an at least 8-bit analog to digital converter and saved into a specified location of the data memory block in your microcontroller. You should measure the light intensity once in 2 seconds. You should plot the graph of temperature with time for last 64 seconds (32 measurements) on a VGA screen. Moreover you should calculate the average light intensity in 64 seconds using the microcontroller you designed and print it on the VGA screen you used. To drive VGA screen you do not have to use your microcontroller. You can write an independent Verilog module to drive VGA screen. Details about VGA screen and Analog to Digital Converter are given in Appendix. You can use any ADC which is available on the market. Note that the FPGA development board has an ADC on it. However the output of the ADC on the FPGA board communicates with FPGA with SPI. To use this ADC you should write an SPI module. You are encouraged to use the ADC on the FPGA development board. The students using ADC on the FPGA development board will be awarded with bonus grades.

Appendix A: VGA Interface

VGA is a widely used standard in video industry for the transmission of video signals from a computer or microprocessor into a monitor or TV. Each 640x480 image is called a 'frame' and each frame contains 480 lines which are made up of 640 pixels.

The monitor starts displaying each frame by beginning from the first line and then the first pixel of this line. In each line, the display order is from left to right; and each frame is written in an order from top to bottom. So, your first pixel is always at the top left corner, while the last pixel at the bottom right.

You will need to generate an image buffer with at least 640x480 = 307200 bits to store each line and frame in order to form a coherent image; however you will also need to adjust two synchronization signals called HSync (Horizontal Synchronization) and VSync (Vertical Synchronization) in order to see a video. These signals tell the monitor when a line or frame is finished, and the monitor should start from the next line or frame.

As shown in Figure 3, VGA interface is actually very simple, and you will only need to make 3 connections, namely R-G-B. For example, for a white pixel all three inputs should be high, and for a black pixel the inputs should be low. The FPGA cards in the laboratory already have a VGA output port with color outputs, so you will only need to supply the R-G-B data digitally to the VGA port. Necessary pins for these assignments can be found in the user manual. (http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=836&FID=eac30a7aaacf5187a4ace0d613cd4676)

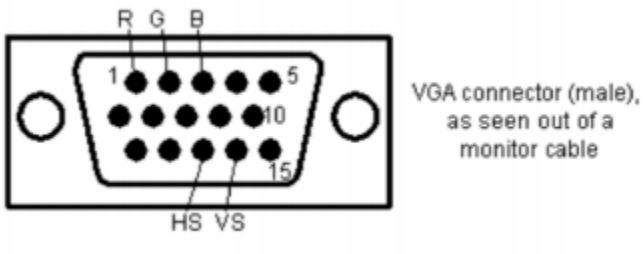


Figure 1: VGA interface.

HSync and VSync: HSync and VSync are necessary in order to tell the monitor to 'start' or 'stop' writing a line or frame. You will need to build the necessary digital blocks in order to correctly form these two signals. These blocks are basically counters with some modifications and are very easy to implement in Verilog. You can see the horizontal and vertical synchronization signals in Figure 2 with the corresponding timing in Table 1.

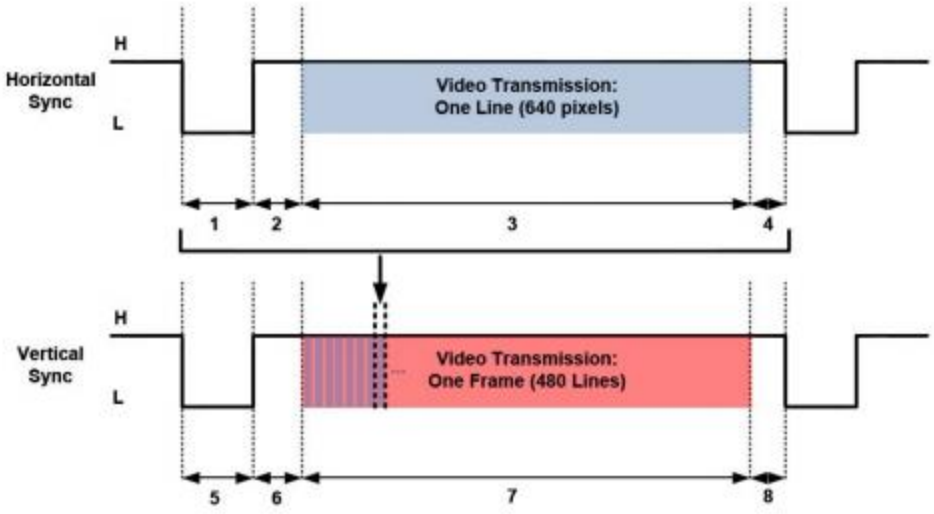


Figure 2: HS and VS.

Timeline # on Fig. 1	Name	Duration	Clock Count
1	H. Sync	3.84 µs	96
2	Back Porch (H)	1.92 µs	48
3	Video Signal (One Line)	25.6 µs	640
4	Front Porch (H)	0.64 µs	16
5	V. Sync	0.064 ms	2
6	Back Porch (V)	1.056 ms	33
7	Video Signal (One Frame)	15.36 ms	480
8	Front Porch (V)	0.32 ms	10

Table 1: Timing.

By observing Figure 2 and Table 1, we can understand that the HSync signal is used to synchronize one line in a frame, while VSync is used to synchronize each frame. Basically, when HSync or VSync is low, the monitor understands that it needs to switch from one line or frame to the next. Back and front porch are idle stages where the monitor is getting ready to write the next pixel or line. They also include 8 pixel and line over scan or 'border' pixel/lines outside our standard view of the monitor.

IMPORTANT NOTE: The video input signals (R, G, B) of a VGA monitor should be off (or black) during H. or V. Sync stages, and front/back porch stages. The video input signals should only be active during an active video transmission stage, which are highlighted in Figure 2.
In order to construct these HSync and VSync signals and to achieve transmission of each line/pixel, you will need a 25 MHz clock signal. This will also mean that each pixel will be transmitted at 25 MHz to the monitor during active video stages.
Internal clock information about ALTERA can be found under the Clock Circuitry part of the user manual.

http://www.epanorama.net/documents/pc/vga_timing.html
http://martin.hinner.info/vga/640x480_60.html

Appendix B: Analog to Digital Converter

-Read chapter 3.6.12 and chapter 5.9 from the user manual.

-Read the datasheet of the ADC which is on the FPGA development board.

Reference Readings:

http://www.ee.iitm.ac.in/~nitin/teaching/ee5480/micro.html

http://andrew.gibiansky.com/blog/electrical-engineering/your-very-first-microprocessor/

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7092116