EE3207E Group 2

Final Project Report

Team Members

Vaghul Aditya Balaji	A0074270L
Ravi Theja Reddy Singannagari	A0074598L
Jin Qian	A0066110X

Introduction

Despite its relatively old age, the 8051 is one of the most popular microcontrollers in use today. Many derivative microcontrollers have since been developed that are based on and compatible with the 8051. Thus, the ability to program an 8051 is an important skill for anyone who plans to develop products that will take advantage of microcontrollers.

Intel 80C51 Processor

The 8051 is a microprocessor that was developed by Intel in 1980 for use in embedded systems applications. It follows the Harvard architecture, and is very popular even today due to its flexibility.

A main reason for its popularity is because of its configuration that suits a wide range of applications and systems. Another reason for its popularity is the number of supporting software that exists for the microprocessor.

The Intel 8051 supports a program memory of 4Kb, and it is able to address up to 64Kb of external memory which can be used to address peripheral memory which can then be connected to the processor. There are 4 I/O ports configured on the chip of which two ports are exclusively I/O and two ports under special circumstances multiplex the address and data lines. There is an on board data memory RAM of 256 bytes, of which 128 bits are bit addressable and 80 bytes of data memory that is directly addressable. It contains a total of 18 Special function registers (SFR's) that are used to access the entire addressing capability of the processor.

Xilinx Spartan 3A FPGA Board

The Spartan-3 generation of FPGAs include the Extended Spartan-3A family (Spartan-3A, Spartan-3AN, and Spartan-3A DSP platforms), along with the earlier Spartan-3 and Spartan-3E families. These families of Field Programmable Gate Arrays (FPGAs) are specifically designed

to meet the needs of high volume, cost-sensitive electronic applications, such as consumer products.

The Xilinx Spartan 3A is a 90mm FPGA and delivers a lot of functionality. The Spartan-3AN platform provides the additional benefits of non-volatility and large amounts of on-board user flash.

Project Synopsis

Overview:-

The project requirement was such that the Instruction Set Architecture of the 8051 microprocessor was to be written and then simulated using VHDL. Subsequently, a working simulation of the 8051 had to be done. Following the above, a C program had to be written using the ISA that had already been defined by us, which will then be downloaded on the FPGA board. In short, the FPGA board had to resemble the working of an 8051 microprocessor. This required a full understanding of the internal working of the microprocessor. The hardware datasheet and the programmers guide were extensively referred to, in writing the ISA of the processor.

Design Stage:-

After the initial time taken to understand the project requirements, the first assignment was to start writing the ISA of the microprocessor. There are a total of 111 instructions that had to be defined and mapped to the corresponding opcode.

The internal functioning of the microprocessor required the definition of various CPU states and execution states that keep changing synchronously with an internal clock. To closely resemble the working of an actual 8051 processor, two CPU states of T0 and T1, and 12 execution states (E0 – E1 for T0 and E0 – E9 for T1) were used in the definition of the ISA. Depending on the instruction, it either required 12/24/48 execution states to be completed.

The working of the Program counter of the microprocessor was done by having an int_rom file where all the instructions that were to be executed were stored sequentially, and then the PC stored the address of the next instruction while the current instruction was being executed. This was achieved by defining the ISA in a file sequencer2.vhd, where the implementation of each instruction was written using a simple switch case statement. The comparison is done using the opcode of the instruction to be executed, which is fetched

from the int_rom file and stored in the Instruction Register, and then the corresponding case statement that matches the opcode is executed.

So, there were a total of 111 instructions that had to be written in VHDL for synthesis. The work was split amongst the three of us.

Instruction Set Architecture:-

The specification of the native instructions took almost an entire month, as there were 111 instructions along with the definition of the I/O, interrupts and registers. The ISA of the 8051 is a CISC processor capable of addressing the instructions such as MOVX (MOV register to an external peripheral), indirect addressing (storing a value at an address), stack storage for temporary variables, function calls and byte operations that require only the opcode for its functioning.

The programmers guide for the microprocessor gives a description of each instruction along with the number of bytes that are required for each instruction and the number of execution states. Typically, the number of execution states varies directly with the number of bytes of data needed by the instruction.

The salient feature of the ISA includes the fact that it is able to support the full addressing capability of the microprocessor (all the 64Kbytes). This feature makes the 8051 a very versatile microprocessor in applications.

Description of the various stages of Operation

Top configuration (i8051_top)

This is the file that is used to connect all the source files that are used in the simulation. It includes the definition and mapping of the I/O ports, SFRs, RAM, ROM and external peripherals if any. From the hardware point of view, i8051_top is the connection, or the bus between the various parts of the processor. When the processor powers up, the i8051_top is the one where the simulator looks for any component declarations.

Adder, Multiplier, Divider (adder_comp, multiplier, divider)

These are individual arithmetic units designed to carry out specific tasks assigned to them. Each one takes in the desired inputs and gives the output in one or more clock cycles. These are separate entities on their own and are utilized by other components of the processor.

Arithmetic Logical Unit (fastalu)

The ALU is the logic brain of the processor where all arithmetic and logic processes are carried out. Each instruction at the lowest level is a combination of the four basic mathematical operations, viz. addition, subtraction, multiplication and division. The ALU is where all these operations are carried out.

Internal ROM (int_rom)

The int_rom file contains the instructions that are to be executed in the form of the corresponding opcodes of the instruction. The value of the opcode is loaded into the instruction register from the int_rom file after it is located using the PC register in T0 phase of the Sequencer. Each instruction is loaded sequentially, and the very first instruction usually defines where to look for the first instruction inside the ROM file. After its execution, the opcode of the next instruction is then loaded into the instruction register. This cycle is executed until the last instruction in the int_rom file has completed execution. After this the no operation instruction, NOP executes for an infinite number of time.

The interrupt vector table is also defined within the int_rom file, and whenever an external interrupt is enabled, the corresponding interrupt is vectored using this interrupt vector table. We can find the predefined Interrupt Vector Table address for each of the interrupt from the manual. So at these locations we must have the address of the Interrupt Service Routines.

Sequencer (sequencer2)

The sequencer is where the instruction set is implemented. The processor executes instructions as defined in the sequencer. As soon as the processor powers up, the instruction register retrieves the instruction specified in the int_rom file, and then the respective instruction is executed. This is the "fetch" and "execute" state of the instruction. The sequencer also polls for any external interrupt, and then vectors the interrupt accordingly. So to summarize, the sequencer has 3 very important phases viz. T0- The fetching phase, T1- The phase where the instruction is decoded and then executed, I0 phase where the sequencer looks for any interrupt. The I0 phase must be after the T1 phase. This is to ensure that the current instruction which is being executed is completed before we jump to any Interrupt Service Routine.

Besides this, the I0 phase must also make a judgment whether to jump to the ISR or not. Suppose we are already executing the Interrupt Service Routine of a high priority interrupt, then suppose we have a low priority interrupt flag set, we must not acknowledge this

interrupt. There is another situation where we use Interrupt Priority Register to elevate the priority of a particular interrupt. In this case, before jumping to the ISR, the I0 phase must also consider the values which are stored in the Interrupt Priority register.

In our sequencer design, we have tried to adhere to the template provided. However some modifications and changes were made to simulate a full working program. Clock cycles are represented by execution states and each instruction takes a specified number of states to complete depending on the complexity of their RTL and the number of bytes that are needed to complete its execution. We have also implemented procedures to implement frequently used function, e.g. read and write operations. Some of these functions are RD_RAM, WR_RAM etc.

Internal RAM (int_ram)

The internal RAM is a 256 byte region of which 128 bits are bit addressable from the location 20H to 2FH and 80 bytes of data memory that is directly byte addressable from the location 30H to 7FH. The higher 128 bytes of the RAM in 8051 contains a total of 19 Special function registers (SFRs). There are a total of 4 register banks. The special function registers are cleared when the processor is turned off or the master reset is pressed. The values are then again reloaded using the int_rom and sequencer file. Although the 8051 is capable of addressing an external memory space treating it as an external RAM and an external program memory treating it as an external ROM, we are not using that in our project.

Register file (regfile)

The top 128 bytes of the RAM which are from 80H to FFH belong to the SFR region. This region of the on chip RAM is managed by the regfile. The SFRs are declared as signals here. Any instruction which reads from an SFR or writes to an SFR uses the regfile to access the particular register. There are a total of 19 Special Function registers that are defined in this file. Some of the important registers that have been used include accumulator A, timer control register TCON, Processor Status Word PSW, Power Control Register PCON, Interrupt Enable register IE and the Interrupt Priority register IP.

Interrupt Handler

The 8051 has 5 interrupts defined, viz. two timer interrupts (timer0 and timer1), two external interrupts (EXT0 and EXT1) and one serial communication interrupt. The order of priority of these interrupts is as follows:
EXT0 > timer0 > EXT1 >timer1 > Serial communication interrupt.

The priority of these interrupts is interchangeable, and can be changed using an interrupt priority register, IP.

This is one of the most important components of 8051top. The job of this component is to look at the TCON register to check if any interrupt flag is set. Once the interrupt flag is set, the interrupt handler will choose the interrupt type of highest priority and store it in a signal. This signal will be used by the I0 phase in sequencer to take care of handling that interrupt. The interrupt handler also includes the IP register in its decision making process. At times we can have cases like TIMER1 being assigned a higher priority than TIMER 0 in IP register. So the interrupt handler will just assign a value to the interrupt type. I0 phase of sequencer will continue from that point.

Project Working

Description

Our project is a simple 4-Bit Binary Calculator which performs the four arithmetic operations (addition, subtraction, multiplication and division) on the four-bit binary operands. It involves the usage of the four capacitive touch-pads and the four LEDs on the FPGA Board as well as an external circuit which is used to get input from the user and display the final output. The sequencer file for this project consists of the complete definition of all the 111 instructions which strictly adhere to the clock cycles as prescribed by the 8051.Moreover, no temporary variables have been used while defining the instructions. The touchpads and the LEDs on the FPGA are manipulated in the program through Port 0, the LEDs on the breadboard through Port 1 and the dip switch through Port 2 and Port 3.

Design

Other than the software and hardware that have already been provided, we have used the following additional components:-

- 8-Bit Dip Switch
- 8 Green LEDs
- 40-Way Data Cable

These components were wired on a breadboard and then they were connected to the General Purpose Input/Output pins on the FPGA Board with the help of the data cable. The

input from the dipswitch is taken by the processor and it is manipulated as required by the user and then the output is sent back again through the cable to the LEDs.

Details

As soon as a power supply is given and the .bit file is loaded into the FPGA board, the program starts executing. One of the four capacitive touchpads is meant to act as a reset button, which when pressed resets the program back to its beginning. The other three touchpads are meant to allow the user to choose one of the four arithmetic operations. One of the touch pads when pressed performs the add operation, another one for subtract operation, another one for the multiplication operation and simultaneously holding on to two touchpads performs the division operation. The four LEDs on the FPGA board act as indicators to tell the user what operation has been executed. The input is given through the 8-bit Dip Switch. The first four bits are for the first operand and the next four bits are for the second operand (from Left to Right => LSB to MSB). So once the user sets the bits on the dipswitch according to his/her preference, the touchpads on the FPGA board are then pressed to obtain the results on the 8 LEDs present on the breadboard. The program is written in such a way that the user has to first give his/her input on the dipswitch and then choose his/her operation through the touchpads.

Special Cases:-

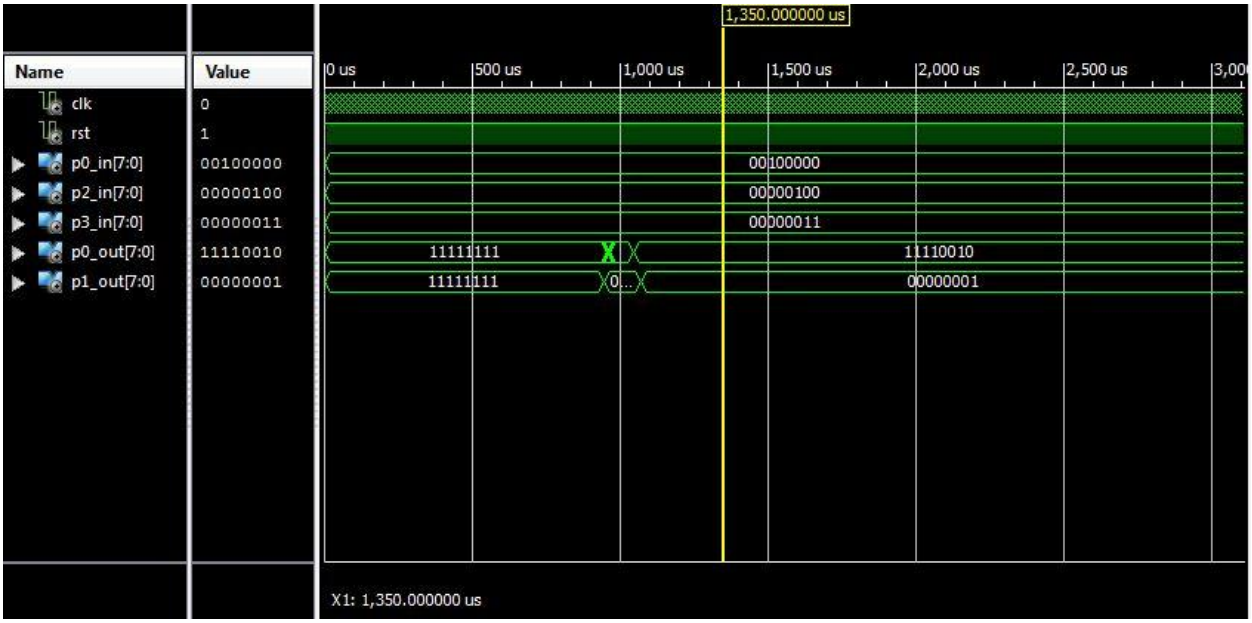
- While performing the subtraction operation, if the user tries to subtract a larger number from a smaller number, all the LEDs on the breadboard will glow indicating a negative result.
- While performing the division operation, if the user tries to divide by zero, all the LEDs on the breadboard will glow indicating an undefined result.

Xilinx Simulations

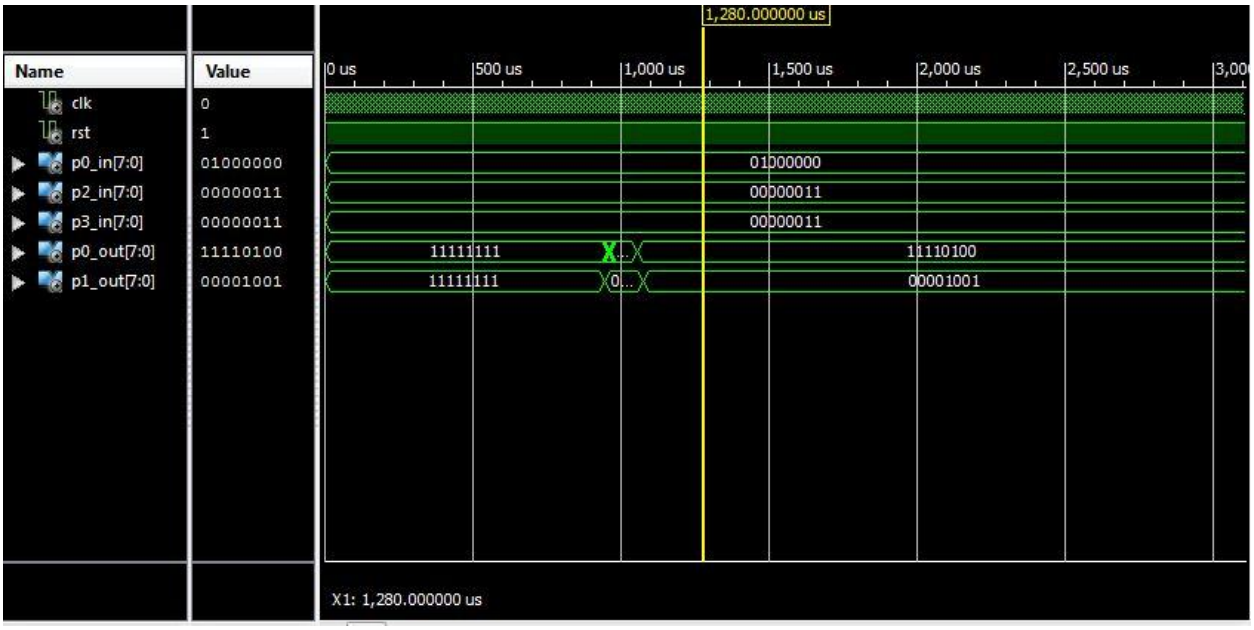
The simulations on the next page demonstrate the working of the four arithmetic operations. P0_in is used to choose the type of operation, p2_in and p3_in shows the user's input through the dip switch, p0_out shows the output on the LEDs on the FPGA board, p1_out shows the final output on the LEDs on the breadboard.

1. Addition of “0100” + “0011”

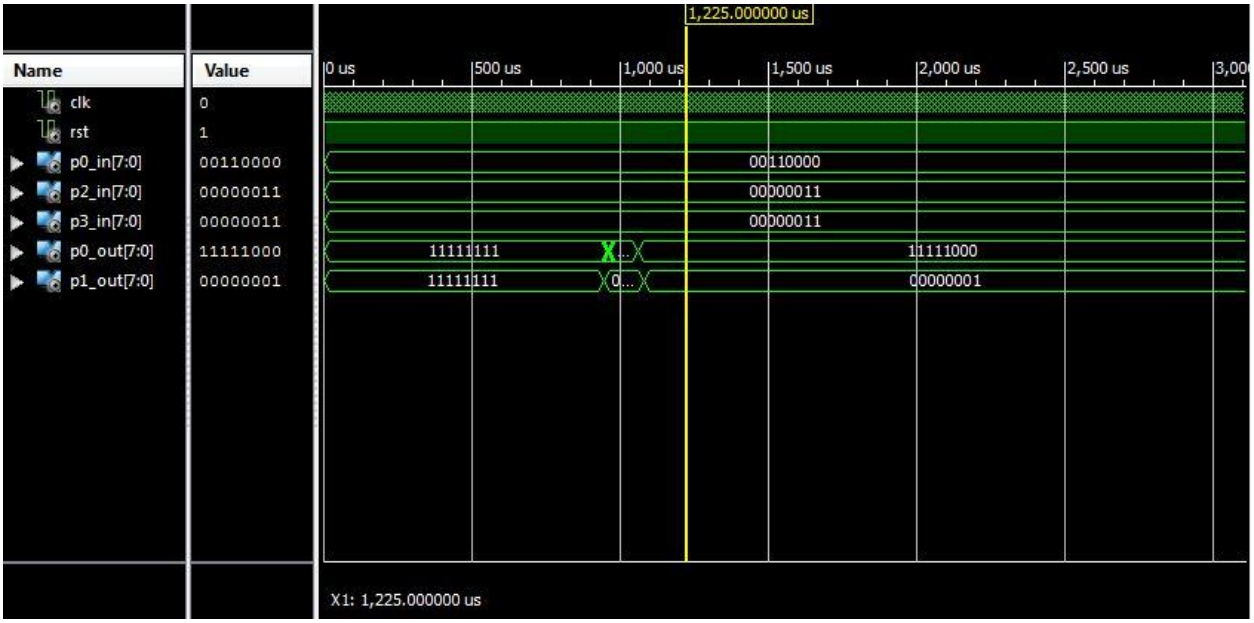
2. Subtraction of “0011” from “0100”



3. Multiplication of “0011” and “0011”



4. Division of “0011” by “0011”



Snapshot of our Project