Introduction

The objective of this digital logic project is to design a simple 16-bit processor on the altera DE2 board that will be capable of executing a set of instructions. A microprocessor is an integrated circuit that functions as the central processing unit of a modern computer. Within this project, the hardware description language known as Very High Speed Integrated Circuit Description Language (VHDL) will be used to design and implement such a processor but instead on a field-programmable gate array device such as the altera DE2 board.

The fundamental set of instructions that the processor executes include: loading a 16-bit constant into a register, moving a 16-bit constant between registers as well as performing addition and exclusive OR operations on two 16-bit operands and overwriting the first operand's register to store the result.

Overall to implement such functionality, the processor consists of a working control unit, program counter, 0.512 kB Random Access Memory (RAM), 8 general purpose registers, an arithmetic logic unit (ALU) and a functional datapath.

Description

Processor Upper Hierarchy

The processor design consisted of a processor entity as the uppermost VHDL block in the design hierarchy which consisted of instantiations of each of the block diagram entities (Figure 1), including the program counter, program memory, control unit, ALU, registers and tristate buffers.

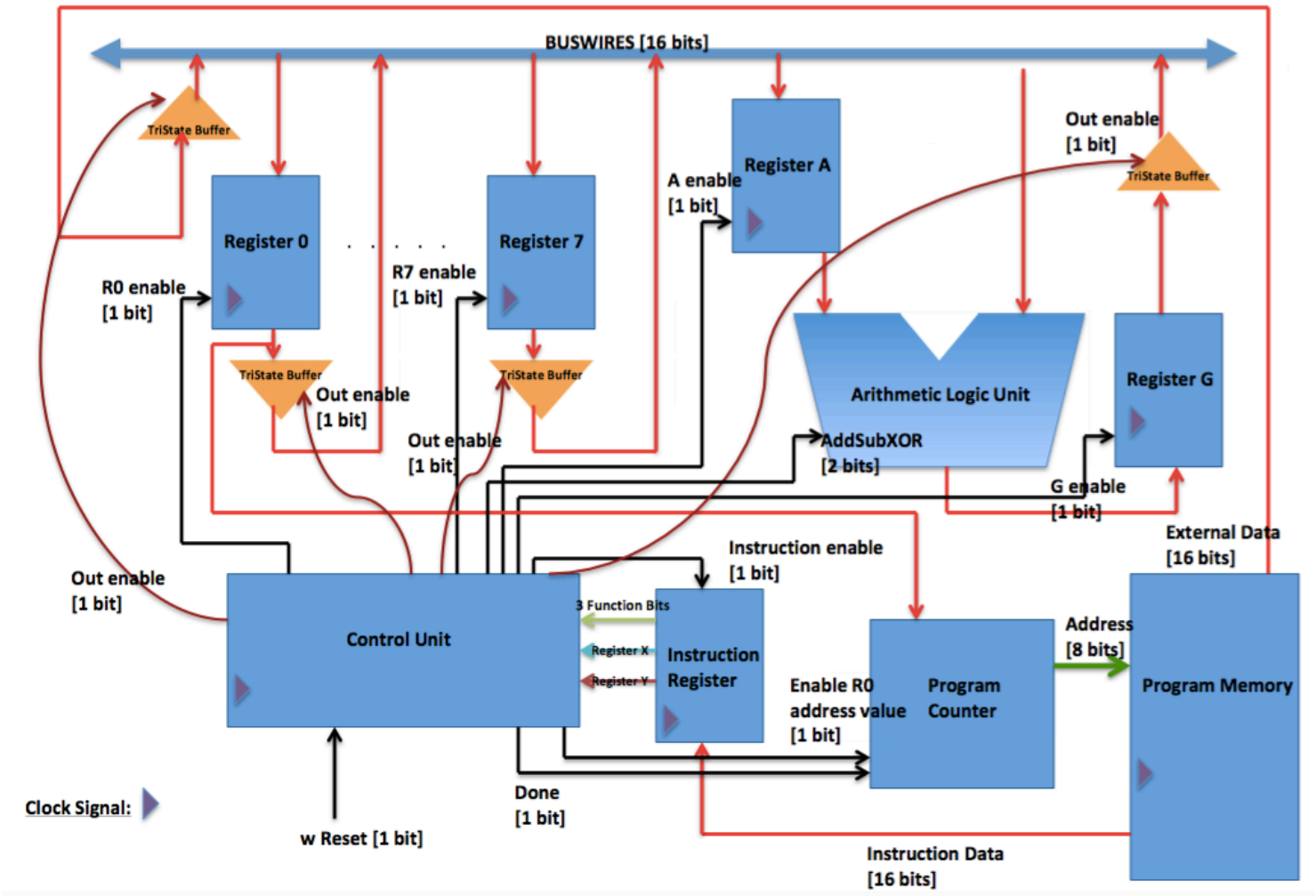

Figure 1: Processor Datapath Block Diagram

Control Unit

The control unit has the purpose of orchestrating the microprocessor's management of machine instructions by setting a future state and output signals of the system based on the current state and input signals. (Figure 2)

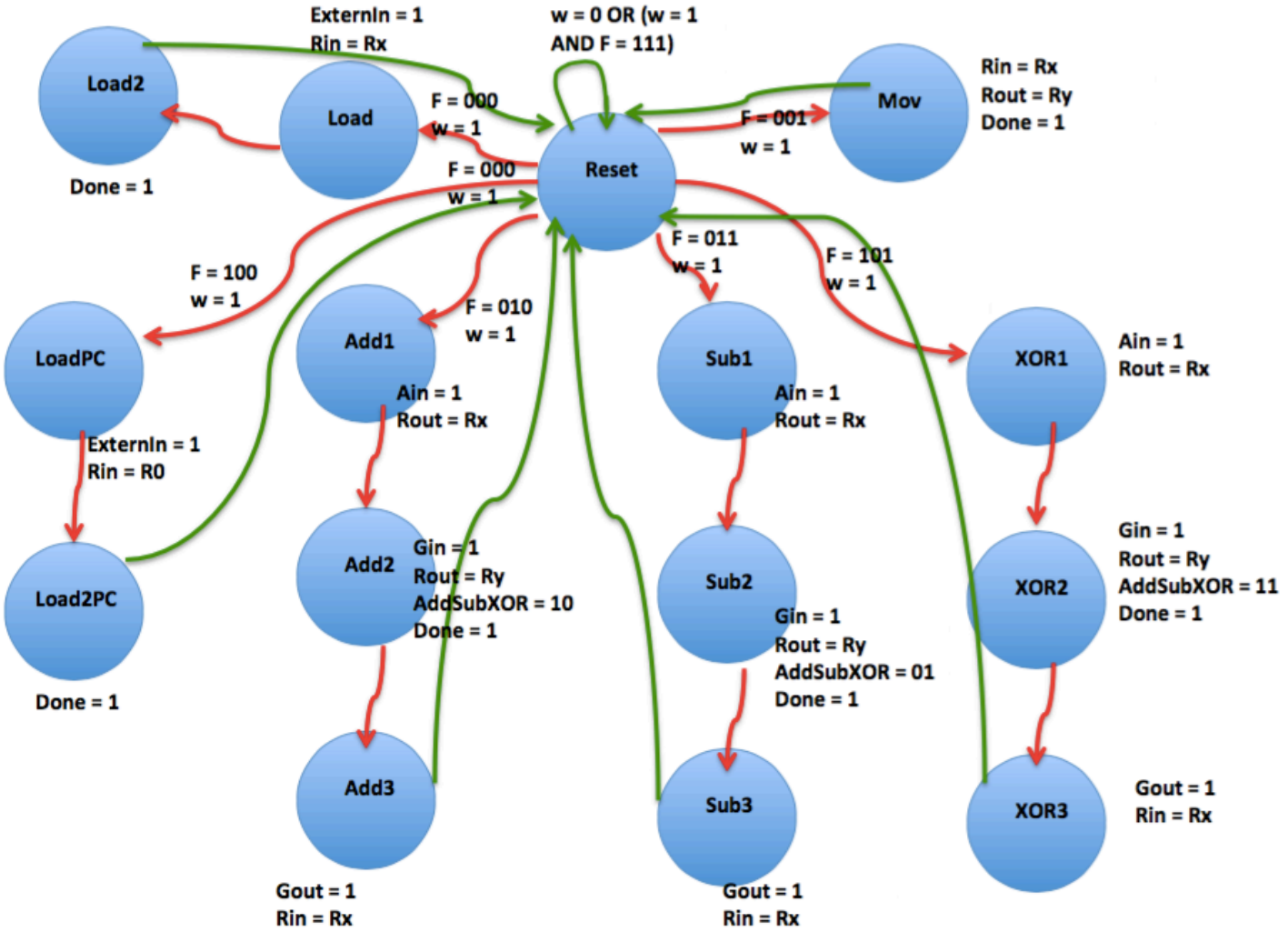

Figure 2: Control Unit Finite State Machine

```
ENTITY controlUnit IS
PORT (
       w : IN std_LOGIC;
    clock: IN std_logic;
      Fcu: IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- 3 instruction bits
      Rx : IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- Rx 3 bits from instruction
      Ry : IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- Ry 3 bits from instruction
      Rin: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --driving ENTERING data for all registers
     Rout: OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --driving EXITING data for all registers VIA A MULTIPLEXER OR TRINODE STRUCTURE
      Ain: OUT std_logic;
      Gin: OUT std_logic;
     Gout: OUT std_logic;
AddSubXOR: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
 ExternIn: OUT std_logic;
      isI: OUT STD_logic;
     Done: OUT std_logic;
 manualCu: OUT std_logic);
END controlUnit;
```

Figure 3: Control Unit VHDL Entity

The control unit consisted of w and clock as external signals as well as the F (instruction code), Rx (register x code), Ry (register y code) as input signals from the instruction register (Figure 3).

The control circuit had the function of determining the next state based on the current state and these input parameters (Figure 2). The F, Rx and Ry signals were decided to be 3 bits in order to allow for a total of 8 instructions and 8 possible registers, this shaped the final instruction protocol (Table 1). As evident, 16 bits of data must have a unique 3 bit function code in order to be loaded rather than be processed as an instruction within the instruction register.

Within the control unit, the Rx and Ry are 3 bit signals passed internally to a register decoder which then returns an 8 bit hot-code encoding for any particular register.

When in a particular instruction state, the control unit is designed to propagate control signals to the ALU and the program counter as well as enable data in/out signals to registers and tri-state buffers for transferral of data between different processor blocks (Figure 2).

Table 1: Processor Instruction Format

16 bit Instruction Format	(3 instruction bits)(Rx 3 bits)(Ry 3 bits)(7 unused bits)
f2 f1 f0 **[function binary encoding]**	**Instruction**
000	Load
001	Move
010	Add
011	Sub
101	XOR
100	Branch (LoadPC)
110	DATA

Program Memory

The program memory block (Figure 1) has the purpose of storing 16-bit instruction and numerical data into memory cells within Random Access Memory (RAM) structure and reading corresponding data from any specific address.

```
ENTITY progMem16 IS
GENERIC ( S : INTEGER := 2;
  N : INTEGER := 8 );
port(
  clock : in std_logic; -- from the processor
  data_in : in std_logic_vector (15 downto 0); -- not required
  write_addr : in std_logic_vector (N-1 downto 0); -- not required
  read_addr : in std_logic_vector (N-1 downto 0); -- from the program counter
  write_enable : in std_logic; --this always 0
  data_out : out std_logic_vector (15 downto 0)); --inks to external in AND is instruction
END;
```

Figure 4: Program Memory VHDL Entity

The program memory depends only on the clock and an 8 bit read address (Figure 4), and depending on a read address would output either instruction data to the instruction register or load data to the bus wires. The disuse of data in, write enable and write addresses is justified, as the program memory within this project is required only to load programs and data from memory.

Program Counter

The program counter has the purpose of storing the 8 bit address of the instruction within program memory being executed by the entire processor.

```
ENTITY programCounter IS
PORT (
         cluk : IN std_logic; --clock as determined by the done signal
       specify: IN std_logic_vector(7 downto 0); --specific address to count from
  resetCounter : IN std_logic; -- to reset must pass 00000000 as the specify address
   enableCount : IN std_logic; --variable to enable counting
         count : OUT std_logic_vector(7 downto 0)); --the actual output address to be used by RAM
END programCounter;
```

Figure 5: Program Counter VHDL Entity

The program counter depends on the clock, a start address to count from, a reset and enable signal (Figure 5) thereby storing a count of the current address, which is passed to the program memory block (Figure 1). With the rising edge of the clock signal and the high enable signal, the program counter correspondingly increments its count until the end of the memory block, eventually restarting from the initial memory cell.

An additional functionality includes the specify address, such that when the reset signal is active the program counter resets to the address given by the specify signal which is loaded into register 0 by the LoadPC instruction states. This effectively gives the functionality of a Branch instruction.

ALU

The arithmetic logic unit has the purpose of carrying out addition and subtraction as arithmetic operations as well as the exclusive OR logical operation. The arithmetic logic unit consists of two input operand signals, being A and B as well as 'As' as the AddSubXOR control signal that is used to control which arithmetic or logic operation is to be performed. Furthermore, the G out signal represents the result of an operation and the carry signal corresponds to the carry bit returned from an addition or subtraction operation.

```
ENTITY myALU IS
PORT (
        A : IN std_logic_vector(15 downto 0);
        B : IN std_logic_vector(15 downto 0);
        As: IN std_logic_vector(1 downto 0); --As is AddSub signal control signal
        G : OUT std_logic_vector(15 downto 0);
    Kcarry: OUT std_logic);
END;
```

Figure 6: ALU VHDL Entity

The ALU also consists of sub components including a multiplexer, 16-bit adder and a fundamental full adder unit. Depending on the AddSubXOR signal, the ALU uses the multiplexer to choose between the B signal and an inverted B signal for addition and subtraction operations, respectively. Meanwhile, the structural VHDL instantiation of the complete adder with the full adder is used to implement the overall adding function with carry signals between 16 adding units. Contrastingly, the XOR is implemented using a behavioural method in VHDL.

General Purpose Registers

The general-purpose registers have the function of being quickly accessible locations for storing and retrieving 16 bits of data within the processor structure (Figure 1). Whereas, register A has the function of storing one of the two operands during an ALU operation, Register G has the function of storing the result of an operation. Contrastingly, the instruction register functions as storing the instruction data that is loaded from program memory

```
ENTITY R16bReg IS
PORT (
       Dreg : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        clk: IN STD_LOGIC;
     enable: IN STD_LOGIC;
       Qreg: OUT STD_LOGIC_VECTOR(15 DOWNTO 0));

END R16bReg;
```

Figure 7: General Purpose Registers VHDL Entity

The main structure of the registers is based in on an enable input signal that allows the register to latch input data based on the rising edge of the input clock signal, this resulting latched data is driven as the output signal of the register (Figure 7).

Hex Decoder

The hexadecimal decoder functions to convert a 4 bit input positive integer into the 7-bit output of the hex display that corresponds to the input number.

```
entity hexdecode is
  port ( A : IN std_logic_vector(3 downto 0);
         D : OUT std_logic_vector(0 to 6));
end;
```

Figure 8: Hex Decoder VHDL Entity

Tristate Buffer

The tristate buffer implemented in the processor design has the purpose of disabling an output port by setting the output signal to a high impedance state in order to remove it from the digital circuitry. This can be justified by the fact that having multiple output sources writing to the BusWires can potentially electrically damage or harm the operating state of the processor hardware.

```
entity tristate is
  port(         Atr : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
              enable : in std_logic;
                 Qtr : out std_logic_VECTOR(15 DOWNTO 0));
end tristate;
```

Figure 9: Tristate Buffer VHDL Entity

Conclusion

In conclusion, the design of a 16-bit microprocessor was successful in carrying out a set of instructions. Overall, these instructions include load, move, add, sub, XOR, branch as well as return.