CHAPTER 5

DESIGN OF RISC ARCHITECTURE

5-1. Architecture

Our goal was to design, implement and test a *RISC* (Reduced Instruction Set Computer) using a *FPGA* (Field Programmable Gate Array). The RISC was characterized by 8-Bit architecture having 8-bit Registers, ALU, RAM, Decoders, Counters, Display Unit and Control Unit. The *instruction set* consists of 15 primitive instructions that were encoded using 16-Bit encoding. The RISC is designed using the Hardware Descriptive Language viz. Verilog HDL.

Machine instructions were implemented directly in hardware. Any task too complex for the hardware to execute in a single cycle was performed by executing a series of basic instructions, either as in-line code or by calling a subroutine. Since, RISC microprocessors was much faster in executing each instruction due to simpler instruction set and no complexity, a net gain in performance results. The following sections would provide background information of RISC architecture and why it was chosen for our project.

The data flow in processor core is illustrated in Figure 5.1, on the next page.

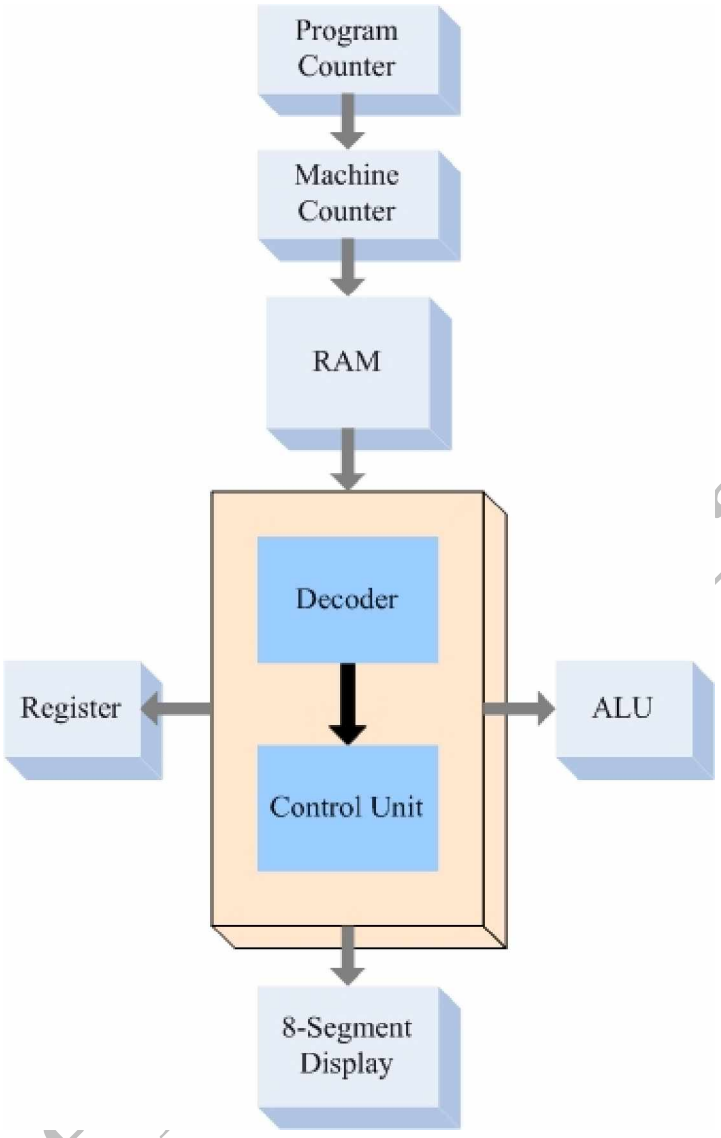

Figure 5.1: Processor Core Design.

5-2. Why RISC?

RISC, in many respects, is nothing more than a collection of evolutionary advances in the way a CPU and its component parts are assembled. It is basically a new style of microprocessor with a few old tricks from the mainframe world.

Most microprocessors in today's market are based on either the *RISC* (Reduced Instruction Set Computer) or *CISC* (Complex Instruction Set Computer) architecture technologies. Research has shown that RISC architecture greatly boosts

computer speed by using simplified machine instructions for frequently used functions. The instruction set in this case is etched into logic circuits using HDLs like Verilog or VHSIC. This instruction set is reduced to basic, often used commands that can be executed in a single machine cycle.

Data collected about general-purpose computers show that up to 80% of their time is spent executing simple instructions such as Load, Store, and Branch. If something difficult is desired, the compiler should generate several simple instructions. The more complex instructions consist of a very small amount of the overall execution time of the CPU. Therefore, to avoid complex instructions and optimize the RISC processing power, no complex addressing is allowed and all instructions sets act on the internal register set.

These factors make RISC an irresistible choice and most modern processors are built on this form factor. Since RISC designs obtain their speed from simplicity, it is understandable why the corporate technology-world is attracted to such a design [19-21].

5-3. Processor External Design

The Figure 5.2 illustrates the processor's 8-Bit architecture and the ability to read and write to external memory. The designed processor core consists of Registers, ALU, RAM, Decoders, Counters, Display Unit and Control Unit connected by a Central Bus denoted as BusWires. Control Unit provide the necessary control signals that allow data to be moved or copied over the BusWires as well as for performing an ALU operation on the data. For example, data can be read from an external device into a selected register. There are control signals to allow the contents of any register to be supplied to the temporary register (A) or placed on the BusWires, which in-turn inputs the data to the ALU. Other control signals allow the result from the ALU to be stored back into memory, driven externally to display unit, I/O or driven on the BusWires.

Data that is picked from the BusWires can either be clocked into the Register for memory storage or can be latched into the Control Unit as an Instruction. A Display Unit was developed in addition to the core design for hardware debugging

purposes. This Display Unit was capable of driving the value of the Data, BusWires or ALU to the Seven Segment Display external to FPGA. This Display Unit has one control signal for its operation known as Done. This Done line works as is an internal switch which selects which value will be shown on seven segment display. For a detailed description of the display unit see Appendix F.

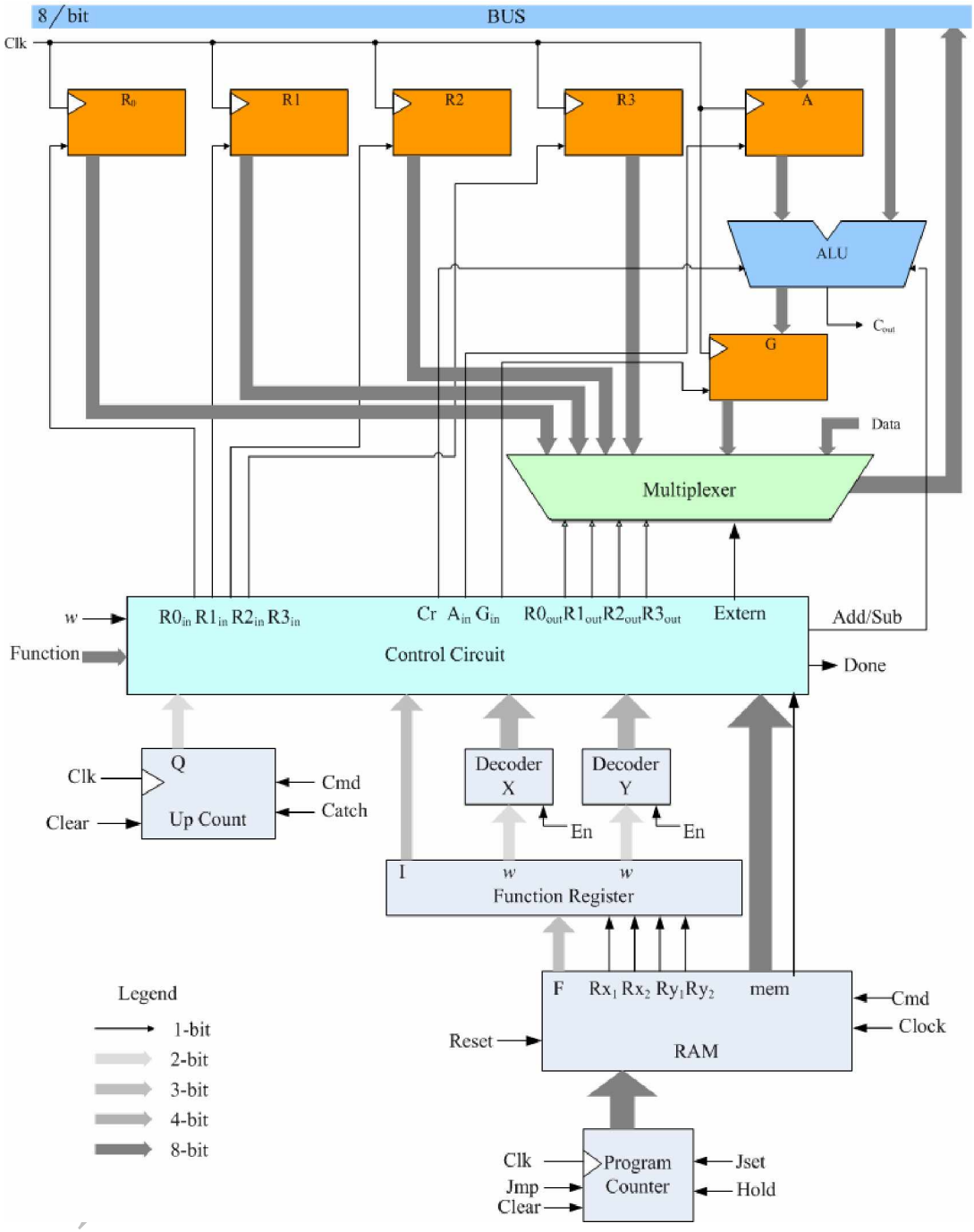

Figure 5.2: Block Diagram of Proc Module.

5-4. Processor Internal Components

5.4-1. EXTERNAL CONTROL SIGNALS

As illustrated in Figure 5.2, the following control signals are provided externally to the processor core for its control and operational functionality.

Resetp is an active hi input, when active; the processor is resetted and various variables are initialized to their starting default values. Similarly *Holdp* is an active low input, when active; only one instruction from memory is allowed to be executed.

The *Clockp* is an externally generated clock signal that is crucial for the operation of all processor components. This signal oscillates at a frequency of 50 or 100 MHZ. All operations occur on the rising edge of the clock signal.

5.4-2. CONTROL UNIT

The Control Unit is the brain of the RISC processor. It consists of numerous outputs and inputs to control the internal functions of the processor as illustrated in Figure 5.2. This unit has two main inputs; *Count* is the 4-Bit signal generated by the machine cycle counter and *pcoutput* is the 16-Bit instruction fetched from RAM. First different parts of the 16-bits instruction are decoded. After decoding, the Control Unit performs the required operation according to the decoded operand I. The Control Unit has two main outputs; Done is a 1-Bit active high signal, which shows the completion of an operation and EXE is a 8-Bit signal, which drives the value of data to be displayed on the external Seven Segment Display.

The Clockp is generated external from the FPGA and driven through an I/O into the top module Proc. The Proc module includes the Control Unit code and instantiate all others sub modules named as pcounter, upcount, Ram, alu, Display, dll, regn and dec2to4. The Spartan-II family FPGA provides four dedicated DLLs for advanced clock domain control, zero propagation delay and low clock skew between output clocks signals distributed throughout the device. Therefore, in order to purify the external clock signal, it is fed into the DLL module. The output clock signal from DLL module is used as input for rest of the processor circuitry because these

dedicated DLLs can be used to implement several circuits which improve and simplify system level design.

The maximum number of clock pulses consumed by an instruction is four. The Jump, Move and Load instruction require one machine cycle, while the rest of instruction consume four machine cycles. On every rising edge of the clock, the Control Unit generates the correct signal logic to perform the desired operation. The Resetp and Holdp input are also generated externally and is driven through an I/O into the top module Proc. The Resetp is configured to be an asynchronous reset, meaning the reset must be held down until sufficient clocks have expired to bring the state machine back to the beginning and the Holdp signal is used to execute the code unit step.

The circuit design (or RTL Schematics) of the top level module *Proc* is shown in the following figures. Figures 5.3, 5.4, 5.5 and 5.6 describe all circuitry that will be implemented in the FPGA. Initially, when Resetp signal is high all sub modules of the processor are reset to their default values and the RAM is reloaded with the processor instruction set. The design code for Control Unit is also given in Appendix F.

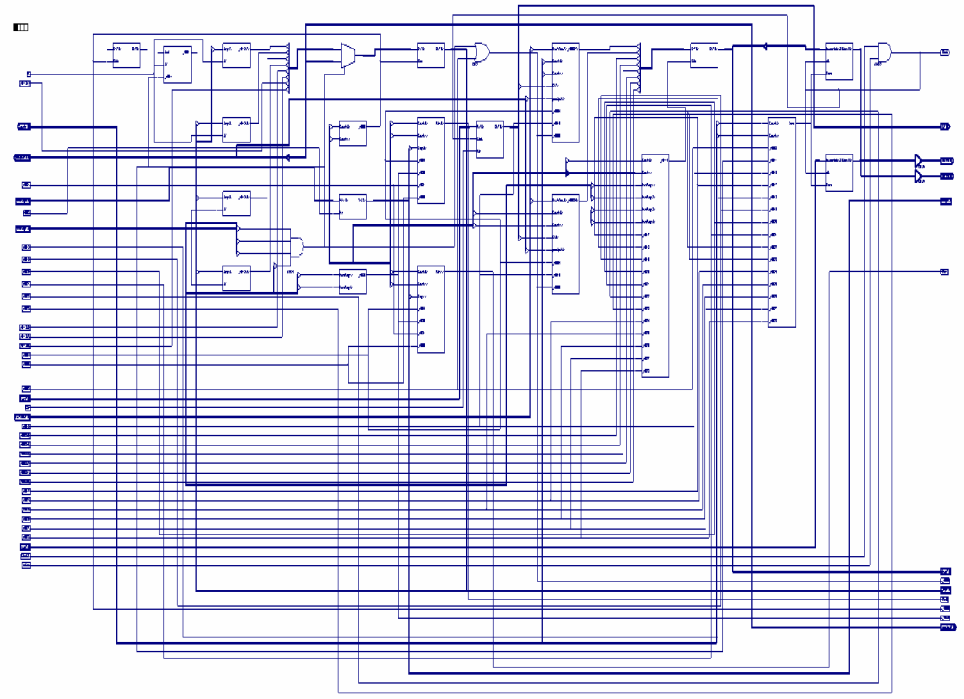

Figure 5.3: Proc module circuit design 1.

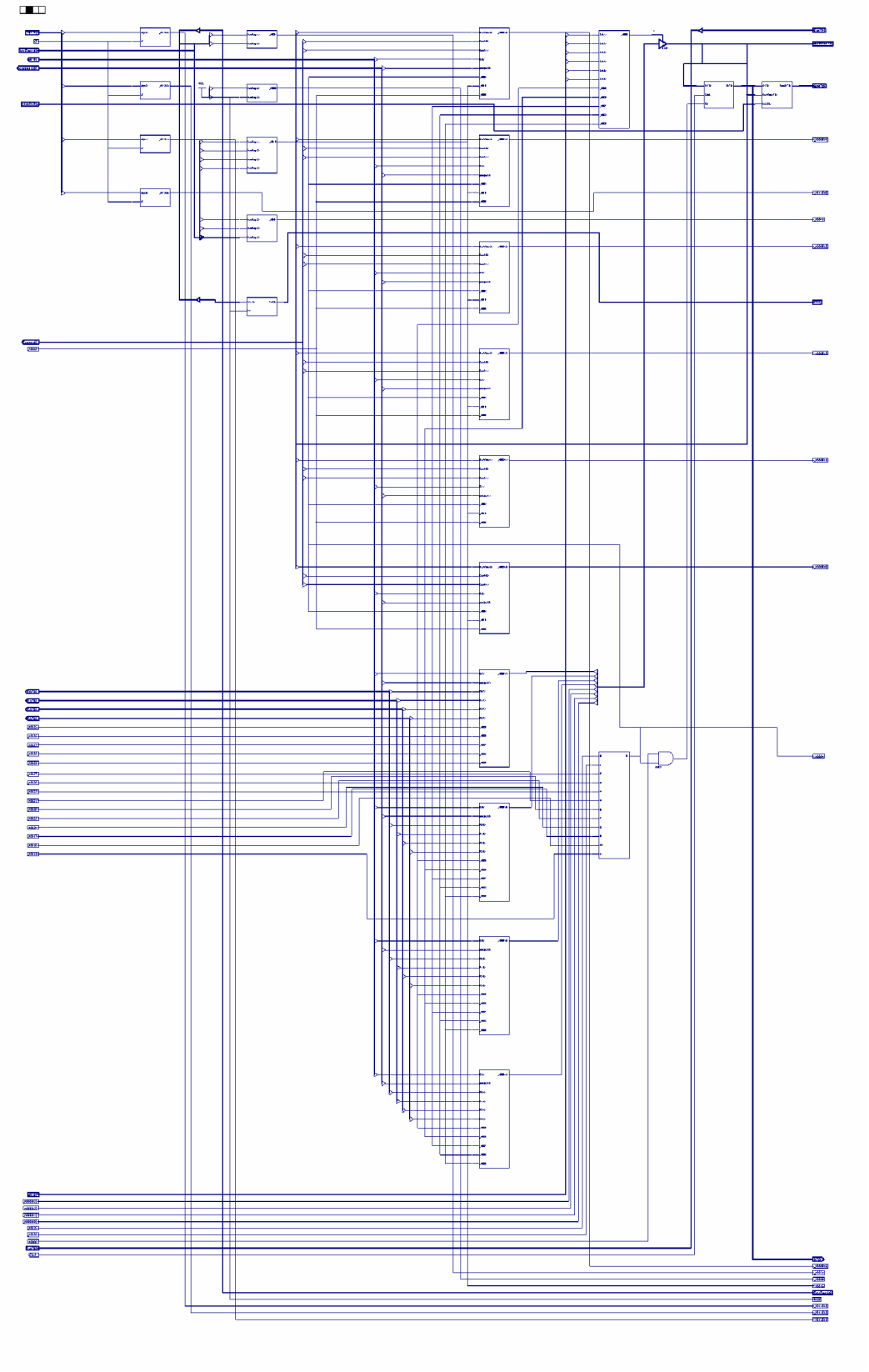

Figure 5.4: Proc module circuit design 2.

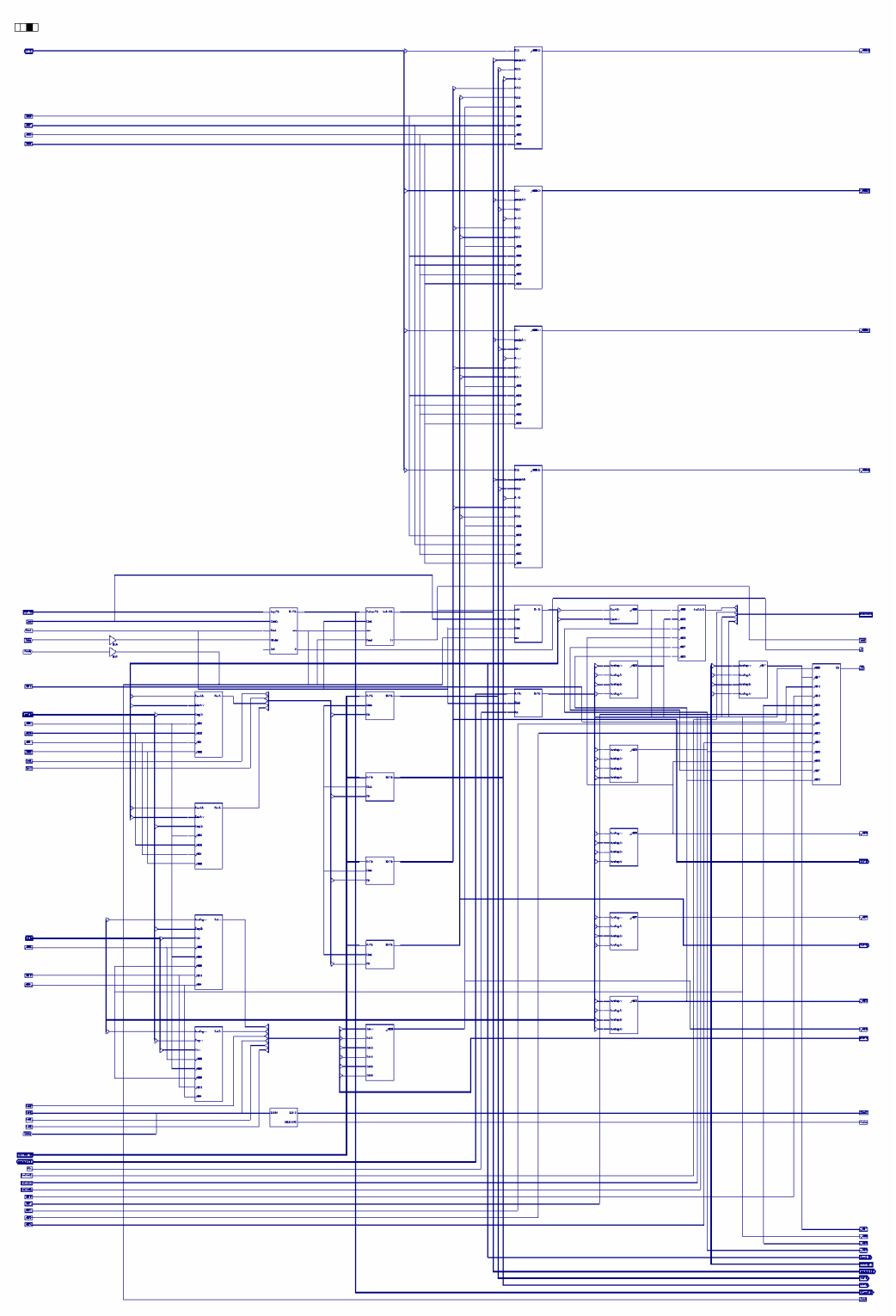

Figure 5.5: Proc module circuit design 3.

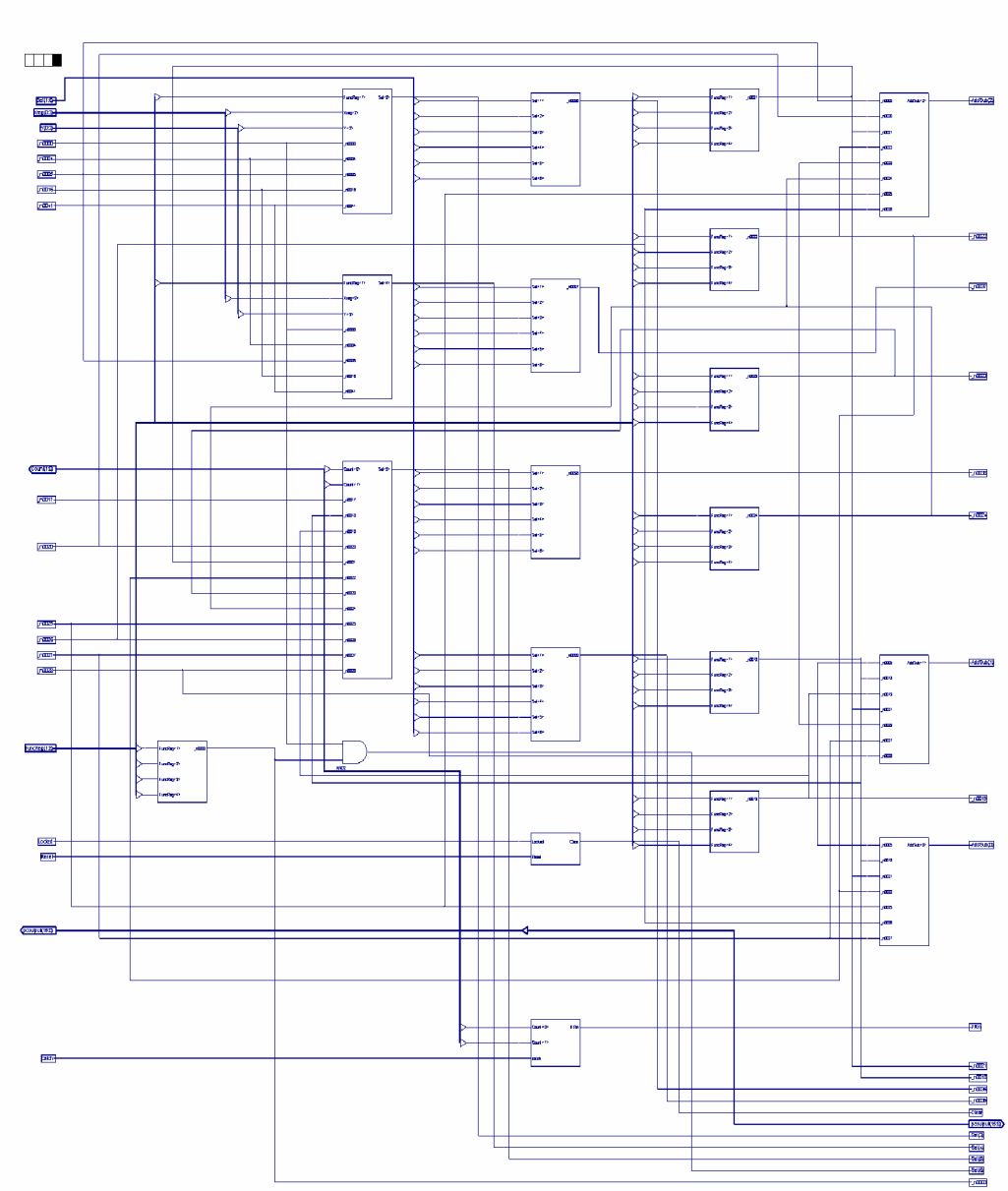

Figure 5.6: Proc module circuit design 4.

5.4-3. ALU

The *ALU* (Arithmetic Logic Unit) is used to perform various bitwise operations. When the ALU performs any operations, it generates the status. Using the flags the Control Unit is able to determine when an overflow, zero, carry, half-overflow, negative and other conditions have occurred. The ALU perform operations like Additions, Subtractions, Multiplication, AND, NAND, OR, NOR, Not, Shifts and

Rotate on the input data from the temporary register A and BusWires. Circuit diagram for ALU is illustrated in Figure 5.7 and the design code is also given in Appendix F.

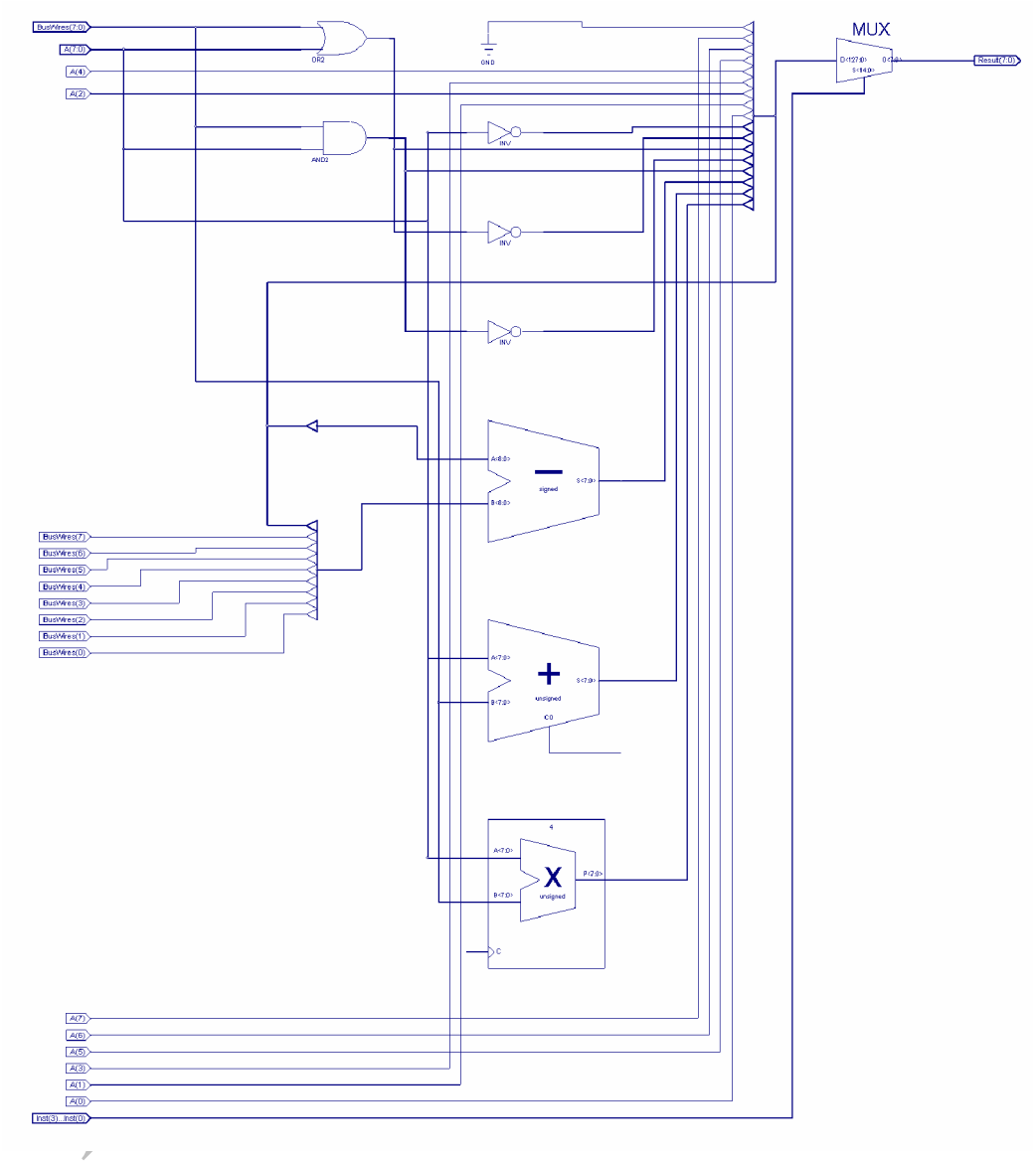

Figure 5.7: ALU Module's Circuit Design.

5.4-4. DECODER

The CPU core has the 2to4 decoder for the 2-Bit valued signals Rx and Ry, which are generated by the Control Unit from the decoded 16-Bit instruction. With the help of 2to4 decoder we can select four different registers with the 2-Bit value of

either Rx or Ry. Therefore four registers are selected by the 2-Bits of Rx and next four registers are selected by the 2-Bits of Ry. Circuit diagram for Decoder is illustrated in Figure 5.8 and the design code is also given in Appendix F.

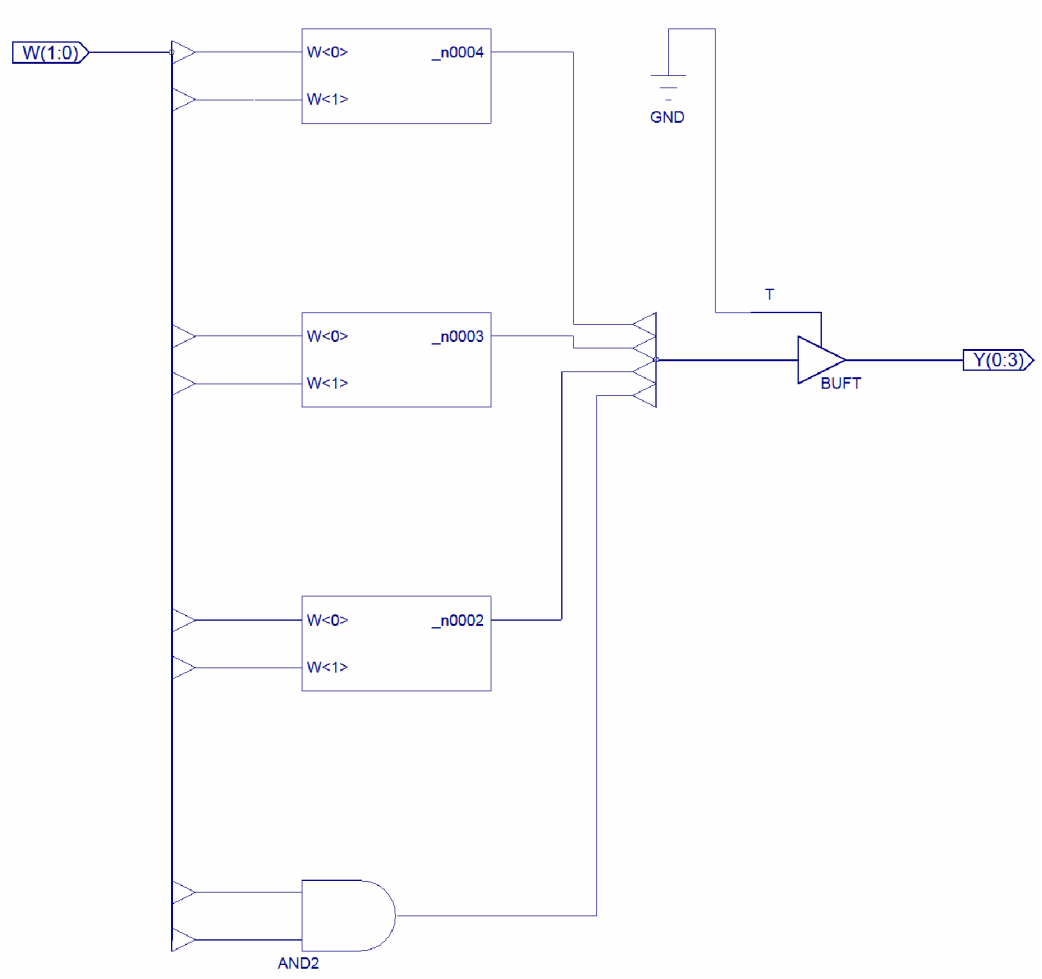

Figure 5.8: Decoder Module's circuit representation.

5.4-5. PROGRAM COUNTER

The program counter controls the execution sequence of instructions. After an instruction is executed, program counter is incremented by one value. When microprocessor is resetted, the default value of program counter is loaded i.e. zero. Circuit diagram for Program Counter is illustrated in Figure 5.9 and the design code is also given in Appendix F.

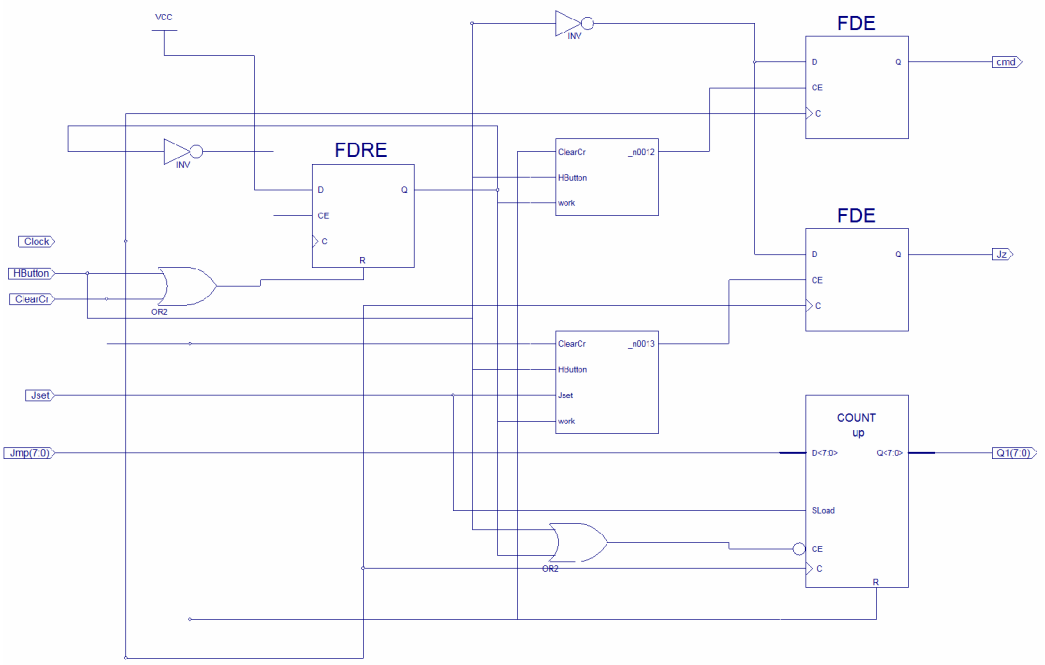

Figure 5.9: Program Counter Module's Circuit Representation.

5.4-6. MACHINE CYCLE COUNTER

The Machine Cycle Counter is a simple 2-Bit up counter and provides the necessary timing for the instruction execution in the control unit. Also we can say that it counts the number of machine cycles consumed by a single instruction. Circuit diagram for Machine Cycle Counter is illustrated in Figure 5.10 and the design code is also given in Appendix F.

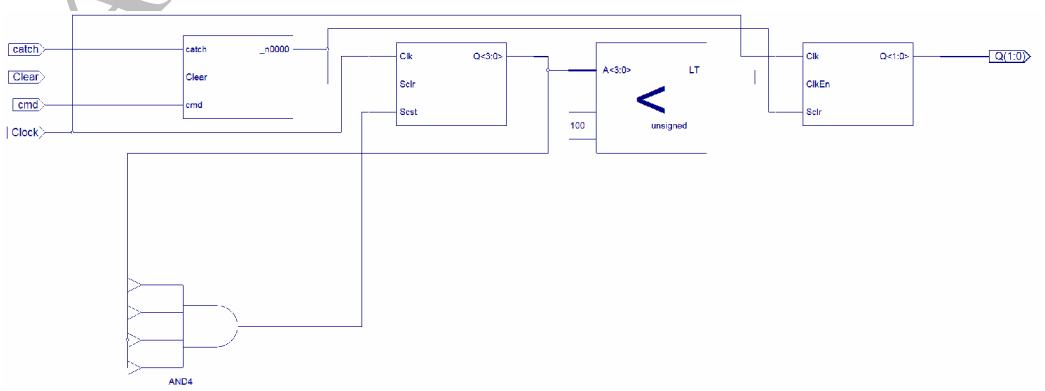

Figure 5.10: Machine Cycle Counter Module's Circuit Representation.

5.4-7. REGISTER

The Register module is an 8-Bit Verilog construct by which several registers can be instantiated. The current processor core utilizes six 8-Bit registers but several others can also be created if required. Registers act as temporary storage for data under processing. Circuit diagram for Register is illustrated in Figure 5.11 and the design code is also given in Appendix F.

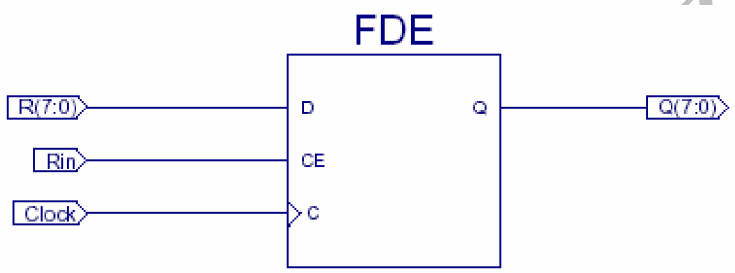

Figure 5.11: Register Module's Circuit Representation.

5.4-8. DELAY LOCKED LOOP

As we know, associated with each global clock input buffer is a fully digital *DLL* (Delay-Locked Loop) that can eliminate skew between the clock input pad and internal clock-input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arriving at internal flip-flops is in synchronism with clock edges arriving at the input pads. Circuit diagram for Delay Locked Loop is illustrated in Figure 5.12 and the design code is also given in Appendix F.

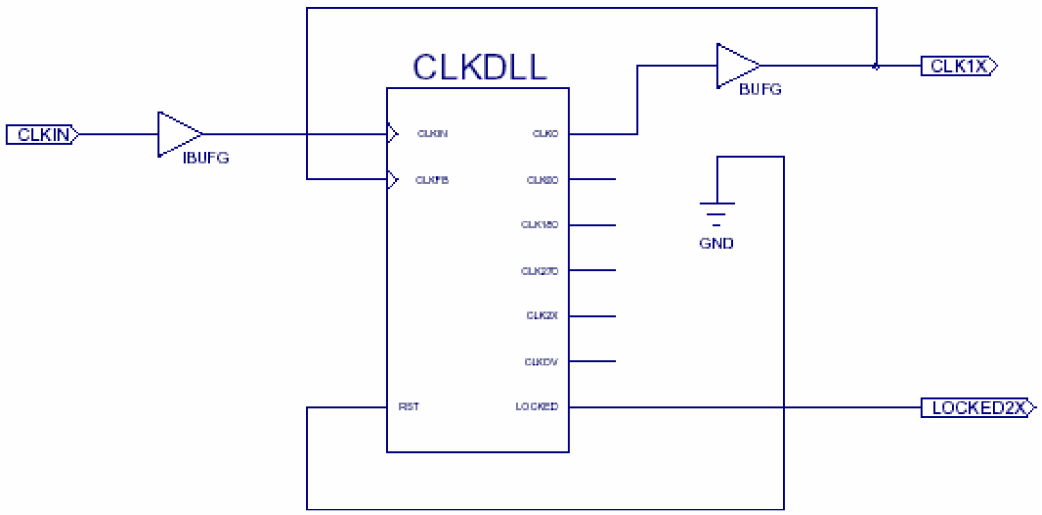

Figure 5.12: CLKDLL Module's Circuit Representation.

In addition to eliminating clock-distribution delay, the DLL provides advanced control of multiple clock domains. In addition the DLL also provides four quadrature phases of the source clock, can double the clock, or divide the clock by 1.5, 2, 2.5, 3, 4, 5, 8, or 16. It has six outputs.

The DLL also operates as a clock mirror. By driving the output from a DLL off-chip and then back on again, the DLL can be used to de-skew a board level clock among multiple Spartan- II devices.

In order to guarantee that the system clock is operating correctly prior to the FPGA starting up after configuration, the DLL can delay the completion of the configuration process until after it has achieved lock.

5.4-9. BLOCK-RAM

The bus implemented in our design known as *BusWires*. Its width is 8-Bits, providing addresses between 00H-FFH. Since our RISC design have instruction of fixed length i.e. 16-Bits wide, therefore for this design, the memory map consists of the 16-Bit Block-RAM in the address range (00H-FFH). Spartan-II FPGAs incorporate several large block RAM memories. These complement the distributed RAM Look-Up Tables (LUTs) that provide shallow memory structures implemented in CLBs.

Block RAM memory blocks are organized in columns. All Spartan-II devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Spartan-II device eight CLBs high will contain two memory blocks per column, and a total of four blocks as illustrated in Figure 5.13, and the design code is available in Appendix F.

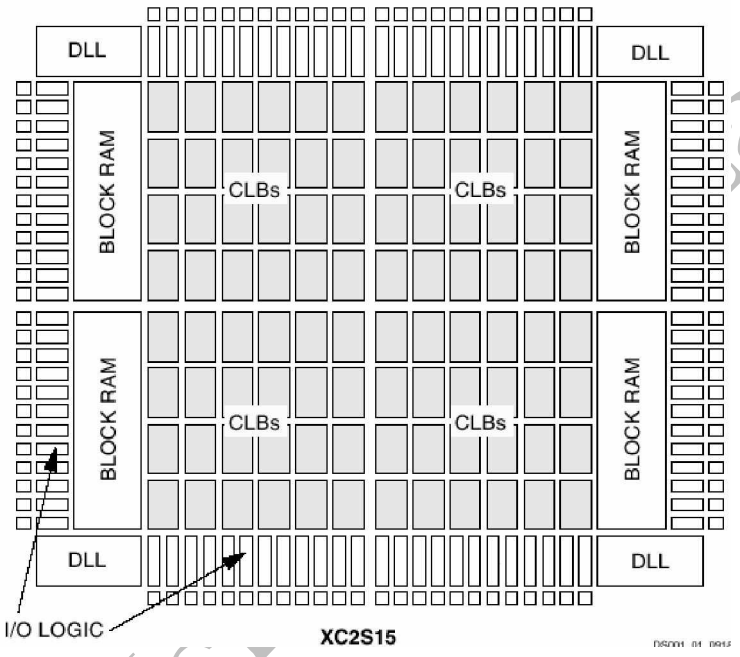

Figure 5.13: Spartan II FPGA Block Diagram.

The Spartan-II FPGA XC2S100 provides ten dedicated blocks of on-chip, true dual-read/write port synchronous RAM, with 4096 memory cells. Each port of the block RAM memory can be independently configured as a read/write port, a read port, a write port, and can be configured to a specific data width. The block RAM memory offers new capabilities allowing the FPGA designer to simplify designs.

Each block RAM cell, as illustrated in Figure 5.14, is a fully synchronous dual-ported 4096-bit RAM with independent control signals for each port. The data widths of the two ports can be configured independently, providing built-in bus-width conversion.

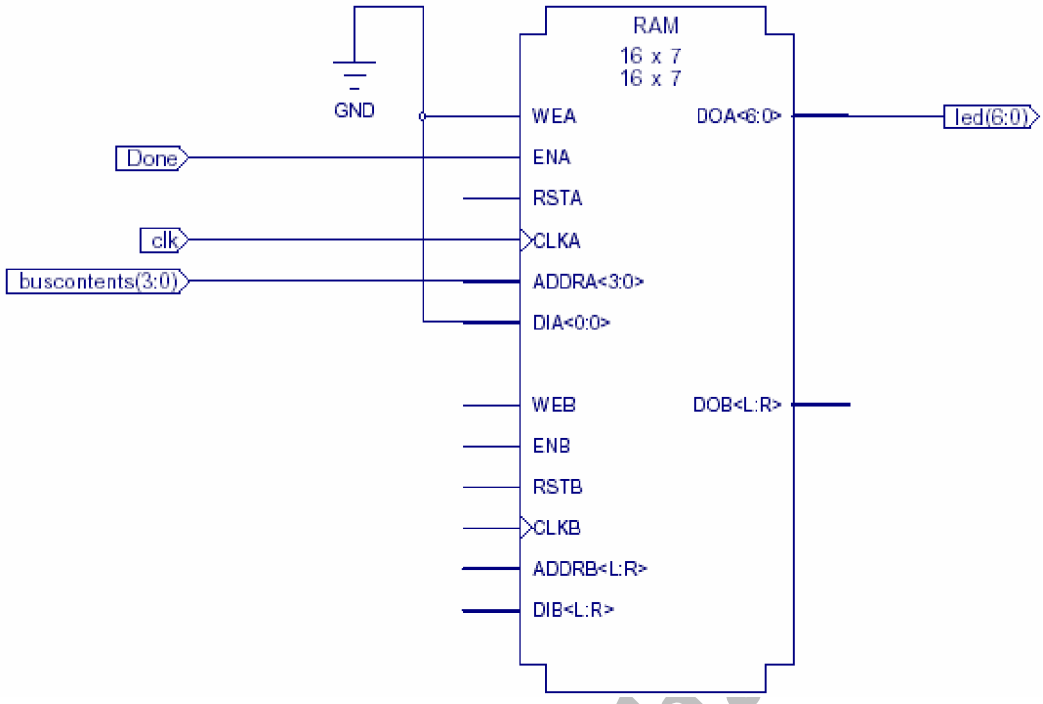

Figure 5.14: Block-Ram Module's Circuit Representation.