Chapter 2

Design Method for 16 bit RISC processor

Usually a design process comprises of basic sequence of tasks that are performed in various situations. The first step is to generate an initial design from an initial concept. This step often requires a lot of manual effort because most of the designs goals can be reached only through the designer's knowledge and skill. The next step is the simulation of the design. For the successful simulation and testing of the design necessary input conditions are required. Applying these input conditions, the simulator tries to verify functionalities of the product as required under the original product specifications. If the simulation has some errors, then the design is changed to overcome the problems. The redesigned version is again simulated to rectify the errors. This loop is repeated until the simulation indicates a successful design. A product designer should rectify errors during simulation because errors are typically much harder to fix, if they are discovered later in the design process.

The flow chart of FPGA design method is shown in Fig.2.1 [93]. The computer aided design tools are there in the hardware design process. Design automation tools help designer with design entry, hardware generation, verification, and design management. VHDL (very high speed integrated circuit hardware description language) is used to describe hardware for the purpose of simulation, modelling testing, design and documentation of the processor. FPGA (field programmable gate arrays) provide the benefit of high integration levels, design and quick verification. FPGA devices are software configured and field programmable. Hence, there is a significant cost saving in design and productions. Design entry of the processor is carried out by using VHDL code. Initial synthesis generates an initial circuit. Functional simulation verifies the functionality of the circuit. Logic synthesis and optimization derive optimized circuits. Physical design implements the optimized circuit in a FPGA chip. Timing simulation determines the propagation delays that are
expected in the implemented circuit. Chip configuration configures the actual chip to realize the design.

![Design Concept Flowchart](image)

**Fig.2.1. FPGA Design Method**

In this Chapter, we describe the design method of the proposed 16 bit RISC processor. Design of digital hardware unit is discussed in section 2.1. CAD tools is presented in section 2.2. The FPGA implementation technology is discussed in section 2.3. A brief review of design method has been provided in previous Chapter 1.

### 2.1 Design of a Digital Hardware Unit

Usually, all classical design methods have PCBs (printed circuit board) that contain many chips together with other components. Development of these products starts with the definition of the overall structure. After that the required integrated circuits chips are selected for the design. Xilinx software is used for complex design.

A complex design is partitioned into smaller blocks and then each block is designed separately. The circuitry in each block is defined, and the chips are selected.
The design is simulated, and any necessary corrections are made. The interconnection between the blocks is defined, which effectively combines these blocks into a large circuit. The complete circuit is simulated. Functional simulation performs all design functions. The next step is to realize this circuit on a PCB. The physical locations of each chip on the board are determined, and the wiring patterns are defined. This step is called physical design of the PCB.

The physical layout may affect the performance of the circuit on the finished board. The CAD (computer aided design) tools will ensure that the required functional behaviour will not change for final circuit. The functional behaviour may be correct but the realized circuit may operate slowly than desired one. Because the physical wiring on the PCB involves metal traces that offers resistance and capacitance to electrical signals. So there may be significant impact on the speed of operation. Then the design is physically implemented. The minor errors are corrected by making changes directly on the prototype board.

2.2 CAD Tools

A typical CAD system comprises of tools for performing the following tasks:

- Design entry allows the designer to enter a description of the desired circuit in the form of truth tables, schematic diagrams, or HDL code.
- Initial synthesis generates an initial circuit, based on data entered during the design entry stage.
- Functional simulation issued to verify the functionality of the circuit, based on inputs provided by the designer.
- Logic synthesis and optimization applies optimization techniques to derive optimized circuits.
- Physical design determines how to implement the optimized circuit in a given target technology, for example, in a FPGA chip.
- Timing simulation determines the propagation delays that are expected in the implemented circuit.
- Chip configuration configures the actual chip to realize the designed circuit.
Introduction to CAD Tools

The classical design is a basic approach for synthesis of logic circuits. A designer can use this approach manually for small circuits. However, logic circuits found in complex systems cannot be designed manually. They are designed using sophisticated CAD tools that automatically implement the synthesis techniques. To design a logic circuit, a number of CAD tools are needed. They are packaged together into a CAD system, which includes tools for the following tasks: design entry, synthesis and optimization, simulation and physical design.

Design Entry

The starting point in the process of designing a logic circuit is the design concept. This step is done manually by the designer because it requires design experience. The rest of the design process is done with the aid of CAD tools. The first stage of this process involves entering into the CAD system a description of the circuit. This stage is called design entry. There are three design entry methods: using truth tables, using schematic capture, and writing source code in a hardware description language.

Truth Tables

Any logic function of a few variables can be described by a truth table. Many CAD systems allow design entry using truth tables. The truth table is specified as a plain text file. The CAD system transforms this timing diagram automatically into a network of logic gates.

The design entry method using truth table is not appropriate for large circuits. It can be applied for a small logic function that is part of a larger circuit. In this case the truth table becomes a sub circuit that can be interconnected to other sub circuits and logic gates. The most commonly used type of CAD tools for interconnecting such circuit’s elements is called a schematic capture tool. The schematic refers to a diagram of a circuit in which circuit elements, such as logic gates, are depicted as graphical symbols and connections between circuit’s elements are drawn as lines.
Schematic Capture

A schematic capture tool uses the graphics capabilities of a computer and a computer mouse to allow the user to draw a schematic diagram. The tool provides a collection of graphical symbols that represent gates of various types with different numbers of inputs. This collection of symbols is called a library. The gates in the library can be imported into the user's schematic, and the tool provides a graphical way of interconnecting the gates to create a logic network.

Any sub circuits can be represented as graphical symbols and included in the schematic. A user should create a circuit that includes within it other smaller circuits. This methodology is known as hierarchical design and deals with the complexities of large circuits. In comparison to design entry with truth tables, the schematic capture facility is better for larger circuits. A disadvantage of using schematic capture is that it has a unique user interface and functionality. Therefore, extensive training is often required for a designer to learn how to use such a tool. Another drawback is that the graphical user interface for schematic capture becomes awkward to use for design of large circuit. A useful method for dealing with large circuits is to write source code using a hardware description language.

Hardware Description Languages

A hardware description language is a language that describes the hardware of digital systems in a textual form [81-83]. It resembles a programming language, but describes hardware structures and behaviour. It can represent logic diagrams, Boolean expressions and other more complex digital circuits. HDL represents and documents digital systems in a form that can be read by both humans and computers. HDL is suitable as an exchange language between designers. The language content can be stored, retrieved and processed by computer software in an efficient manner. There are two applications of HDL processing: simulation and synthesis.

Logic simulation is the representation of the structure and behaviour of a digital logic system. A simulator interprets the HDL description and produces output that predicts how the hardware will behave before it is actually fabricated. Simulation allows the detection of functional errors in a design without having to physically create the circuit. Errors that are detected during the simulation can be corrected by modifying the appropriate HDL statements. The stimulus that tests the functionality
of the design is called a test bench. The design is first described in HDL for simulation. The design is verified by simulation and checking it with a test bench.

Logic synthesis is the process of deriving a list of components and their interconnections (called a net list) from the model of a digital system. The gate-level net list is used to fabricate an integrated circuit or to layout a printed circuit board. Logic synthesis is similar to compiling a program in a conventional high-level language. The logic synthesis produces a database with instructions on how to fabricate a physical piece of digital hardware. Logic synthesis is based on formal exact procedures that implement digital circuits.

A hardware description language (HDL) is used to describe hardware. Many commercial HDLs are available. Some are proprietary, meaning they are provided by a particular company and implement circuits only in the technology provided by that company. VHDL is supported by all vendors that provide digital hardware technology. Both VHDL and Verilog HDL are IEEE standards. Both offer similar features and are in widespread use in the industry. Design entry using VHDL offers a number of advantages. Because it is supported by most companies that offer digital hardware technology, VHDL provides design portability. A circuit specified in VHDL can be implemented in different types of chips without changing the VHDL specification. Design portability is an important advantage because digital circuit technology changes rapidly. The designer can focus on the required functionality of the desired circuit.

Design entry of a logic circuit is done by writing VHDL code. Signals in the circuit are represented as variables in the source code. Logic functions are expressed by assigning values to these variables. VHDL source code is plain text that explains how the circuit works. This feature encourages sharing and reuse of VHDL circuits. This allows faster development of new products.

VHDL code can be written in a modular way. VHDL provides hierarchical design of large circuits. Both small and large logic circuit designs can be represented in VHDL code. VHDL code has been used to design microprocessors with millions of transistors.
The terms associated with VHDL are:

**Entity**: Entity is the part of VHDL which gives the idea about the input and output ports of the design. Entity describes the external view of the design. Entity can be described with the help of entity declaration statement.

**Architecture**: Architecture is the part of VHDL which gives the idea about the logic implemented in the design. It describes the internal view of the entity. Architecture can be described with the help of architecture declaration statement.

**Driver**: Driver is defined as the source on a signal. If the signal is having two sources, then the signal is said to have two drivers.

**Bus**: Bus is the group of wires which connects two or more blocks in the design. Bus is the groups of wires with its driver turn off.

**Package**: Package is defined as the collection of sub-programs and the data types used to build design.

The programmer can write the VHDL program in any one or two or the combination of the following three forms or models:

i. **Data flow model**

   In the data flow model, the logic of the design is implemented with the help of the simple equations. So, it is always simple and efficient way to write the VHDL program so as to get the result by applying certain stimuli (i.e. input).

ii. **Behavioural model**

   In the behavioural model, there is certain process to write the logic of the design. The behavioural model uses the process statement.

iii. **Structural model**

   In the structural model, the logic of the design is implemented by describing the input and output ports and logic of each component associated with the design.
Synthesis

Synthesis is the process of generating a logic circuit from a truth table. Synthesis CAD tools perform this process automatically. The process of translating or compiling VHDL code into a network of logic gates is part of synthesis. The VHDL code in initial synthesis process produces lower level description of the circuit. This process produces a set of logic expressions to realize the circuit. The synthesis tools produce a set of logic equations from the schematic diagram. If truth tables are used for design entry, then the synthesis tools generate expressions for the logic functions.

The initial logic expressions are not in an optimal form. It is difficult for a designer to manually produce optimal results. The synthesis tools automatically produce an equivalent but better circuit. This step of synthesis is called logic synthesis, or logic optimization. The optimized circuit is still represented in the form of logic equations. The synthesis process realizes the circuit in a specific hardware technology. The task involves two steps called technology mapping, followed by layout synthesis or physical design.

Functional Simulation

Functional simulation is to verify that the designed circuit functions as expected. The tool that performs this task is called a functional simulator. It uses two types of information. First, the user’s initial design is represented by the logic equations generated during synthesis. Second, the user specifies valuations of the circuit’s inputs that is applied to these equations during simulation. For each valuation, the simulator evaluates the outputs produced by the equations. The output of the simulation is provided either in truth-table form or as a timing diagram. The user examines this output to verify that the circuit operates as required.

The logic equations used by the simulator are those produced by the synthesis tools. The basic functionality of the design does not change as a result of optimization. The functional simulator assumes that the time needed for signals to propagate through the logic gates is negligible. In real logic gates this assumption is not realistic. The functional simulation provides validating the basic operation of a design. Accurate simulations with timing details can be obtained by using a timing simulator.

The correct operation of the designed circuit can be verified by using functional simulation. The functionality of circuit should be verified as early in the
design process as possible. Then the designer proceeds with logic synthesis, layout synthesis, timing simulation and others.

**Logic Synthesis and Optimization**

The optimized circuit depends both on the type of logic resources available in the target chip and on the particular CAD system. If the target chip is a CPLD, then each logic function is expressed in terms of the gates available in a macro cell. For an FPGA, the number of inputs to each logic function in the circuit is constrained by the size of the LUTs. If the target chip is a gate array, then the logic functions in the optimized circuit are expressed using logic cells.

**Physical Design**

The implementation of the circuit in the target technology is called physical design. This step is called physical design, or layout synthesis. There are two main parts to physical design: placement and routing. A placement CAD tool determines location in the target device for realization of each logic function in the optimized circuit. The placement task depends on the implementation technology. If a PLD is used for implementation, then the structure of the chip is predefined. In the case of a CPLD the logic functions are assigned to macro cells. For an FPGA, each logic function is assigned to a logic cell.

The wires in the chip are used to realize the required interconnections. This step is called routing. Routing depends on the implementation technology. For a CPLD, the programming switches attached to the interconnection wires are set to connect the macro cells together. Similarly, for an FPGA the programming switches are used to connect the logic cells together. If the implementation technology is a gate array or a standard cell chip, then the routing tool specifies the interconnection wires between the rows of logic cells.

**Timing Simulation**

A logic circuit description entered into a CAD system functions as expected by the designer. In functional simulation, it is assumed that signal propagation delays through logic gates are negligible. The timing simulation simulates the actual propagation delays in the technology chosen for implementation.
Summary of Design Flow

After initial synthesis, the logic synthesis tool automatically optimizes the circuit. The physical design tool implements the circuit in the chosen technology. Timing simulation ensures that the implemented circuit meets the required performance. For functional errors it is necessary to revisit the design entry step. For timing errors, it is possible to correct the problems by using the logic synthesis tool. Cost is optimized by minimizing the amount of area needed on the chip to implement the circuit. Speed is optimized by minimizing the propagation delay of signals in the circuit. It is possible to use a faster speed grade of the selected chip or to select a different type of chip that results in a faster circuit. The final step is to configure the target chip to implement the desired circuit.

2.3 Implementation Technology

PGAs (Programmable gate arrays) and CPLDs (complex programmable logic devices) are used to implement a complete digital system on a single IC chip. A PGA is an IC that contains an array of identical logic cell with programmable interconnections. The user can program the functions realized by each logic cell and the connections between the cells. Such PGAs are called FPGAs since they are field programmable.

2.3.1 Programmable Gate Arrays

Programmable gate arrays are semi custom designs. They can be classified as MPGAs (mask programmable gate arrays) and FPGAs (field programmable gate arrays). In MPGAs batches of wafers, with arrays of sites are manufactured. The chip fabrication process entails programming the sites by contacting them to wires. Only the metal and contact layers are used to program the chip in MPGAs.

Field programmable gate arrays can be programmed in the field, i.e., outside the semiconductor foundry. They consist of arrays of programmable modules to implement a logic function. Programming is achieved in two ways. Wires present in the form of segments can be connected by programming anti fuses. Memory elements inside the array can be programmed for module configuration and interconnections.
2.3.2 Field Programmable Gate Arrays (FPGAs)

A programmable gate array is an IC that contains an array of logic cells with programmable interconnections [93]. The user can program the functions realized by each logic cells and the connections between the cells. Such PGAs are called FPGAs since they are field programmable. Field programmable gate arrays offer the benefits of both programmable logic arrays and gate arrays. FPGAs implement thousands of gates of logic in a single integrated circuit. FPGAs are programmable at field to eliminate long delays and tooling costs.

A FPGA consists of configuration logic blocks (CLBs) surrounded by a ring of input output blocks (IOBs). The interconnections between these blocks are programmed by storing data in internal configuration memory cells. Each configuration logic block contains some combinational logic and two D flip-flops. The configuration memory cells are programmed after power is applied to the logic cell array. During configuration, each memory cell is selected in turn. When a write signal is applied to the transistor, data is stored in the cell. Each connection point in the logic cell array has an associated memory cell.

The programmable interconnections between the configuration logic blocks and I/O blocks can be made in several ways: general purpose interconnects, direct interconnects and long lines. In the general purpose interconnect system signals between CLBs or between CLBs and IOBs can be routed through switch matrices. Direct interconnection of adjacent CLBs is possible. Long lines are provided to connect CLBs that are far apart. All the interconnections are programmed by storing bits in internal configuration memory cells.

i. SRAM: Based on static memory technology (CMOS).
ii. Anti fuse: One time programmable (CMOS).
iii. EPROM: Erasable Programmable Read Only Memory technology (CMOS).
iv. EEPROM: Electrically Erasable PROM (CMOS).
v. Flash: Flash Erase EPROM technology (CMOS).

The families of FPGAs are:

a. Xilinx 3000 series
b. Xilinx 4000 series
c. Virtex - 2 series
d. Spartan - 2 series
e. Spartan - 3 series
Table 2.1. Design and Simulation packages for VHDL

The Spartan-II Field-Programmable Gate Array family gives users high performance, logic resources, and a rich feature set, all at an exceptionally low price [89]. The six-member family offers densities ranging from 15,000 to 200,000 system gates. System performance is supported up to 200 MHz. Design and simulation packages for VHDL are shown in Table 2.1.
The Spartan-II family of FPGAs has a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input Output Block (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing channels.

Spartan-II FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes. Spartan-II FPGA is used in high volume applications. Spartan-II FPGAs are ideal for shortening product development cycle with offering a cost effective solution for high volume production. Architectural description of Spartan-II FPGA is shown in Fig. 2.2. Spartan-II FPGA device Options are given in Table 2.2.

![Fig. 2.2. Architectural Description of Spartan-II FPGA](image-url)
<table>
<thead>
<tr>
<th>Device</th>
<th>Speed Grade</th>
<th>Number of Pins, Package Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC2S15</td>
<td>5 Standard Performance</td>
<td>VQ100 100-pin Plastic Thin QFP</td>
</tr>
<tr>
<td>XC2S30</td>
<td>6 Higher Performance</td>
<td>CS144 144-ball Chip-Scale BGA</td>
</tr>
<tr>
<td>XC2S50</td>
<td></td>
<td>TQ144 144-pin Plastic Thin QFP</td>
</tr>
<tr>
<td>XC2S100</td>
<td></td>
<td>PQ208 208-pin Plastic QFP</td>
</tr>
<tr>
<td>XC2S150</td>
<td></td>
<td>FG256 256-ball Fine Pitch BGA</td>
</tr>
<tr>
<td>XC2S200</td>
<td></td>
<td>FG456 256-ball Fine Pitch BGA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temperature Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>C=Commercial</td>
</tr>
<tr>
<td>I=Industrial</td>
</tr>
</tbody>
</table>

**Table 2.2. Spartan-II FPGA Device Options**

The Spartan-II user programmable gate array is composed of five major configurable elements:

i. IOBs provide the interface between the package pins and the internal logic.

ii. CLBs provide the functional elements for constructing most logic.

iii. Dedicated block RAM memories of 4096 bits each.

iv. Clock DLLs for clock-distribution delay compensation and clock domain control.

v. Versatile multi-level interconnects structure.

The CLBs form the central logic structure with easy access to all support and routing structures. The IOBs are located around all the logic and memory elements. The static memory cells control all the configurable logic elements and interconnect resources.
Fig. 2.3 shows the basic Spartan-II FPGA block diagram. The Spartan-II IOB features input outputs that support a wide variety of input output signalling standards. These high-speed inputs and outputs are capable of supporting various state of the art memory and bus interfaces. The basic Spartan-II FPGA block diagram is shown in Fig. 2.4. The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a 4-input function generator: carry logic, and storage element. Output from the function generator in each LC drives the CLB output and the D input of the flip-flop. Each Spartan-II CLB contains four LCs, organized in two similar slices. The Spartan-II CLB contains logic to provide functions of five or six inputs.

Spartan-II function generators are implemented as 4 input look-up tables (LUTs). Each LUT can provide a 16 x 1 bit synchronous RAM. The two LUTs within a slice can be combined to create a 16 x 2 bit or 32 x 1 bit synchronous RAM, or a 16 x 1 bit dual port synchronous RAM. The Spartan-II LUT can also provide 16
bit shift register that is ideal for capturing high speed or burst mode data. Storage elements in the Spartan-II slice can be configured either as edge-triggered D-type flip-flops or as level sensitive latches. The D inputs can be driven either by function generators within the slice or directly from slice inputs.

Each slice has synchronous set and reset signals in addition to clock and clock enable signals. Synchronous set forces a storage element into the initialization state specified in the configuration. Synchronous reset forces it into the opposite state. Alternatively, these signals may be configured to operate asynchronously. The F5 multiplexer in each slice combines the function generator outputs. This combination provides either a function generator that can implement any 5-input function, a 4:1 multiplexer, or selected functions of up to nine inputs. Similarly, the F6 multiplexer combines the outputs of all four function generators in the CLB by selecting one of the F5-multiplexer outputs. This permits the implementation of any 6-input function, and 8:1 multiplexer, or selected functions of up to 19 inputs.

Dedicated carry logic provides fast arithmetic carry capability for high-speed arithmetic functions. The Spartan-II CLB supports two separate carry chains, one per slice. The width of the carry chains is two bits per CLB. The arithmetic logic includes an XOR gate that allows a 1-bit full adder to be implemented within an LC. In addition, a dedicated AND gate improves the efficiency of multiplier implementation. The dedicated carry path can also be used to cascade function generators. Each Spartan-II CLB contains two 3-state drivers that can drive on-chip busses.

Spartan-II FPGAs incorporate several large block RAM memories. 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. A Spartan-II device contains two memory blocks per column and a total of four blocks. The Spartan-II family provides high-speed, low skew clock distribution through the primary global routing resources. 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.

System Development

Spartan-II FPGAs are supported by the Xilinx Foundation. The basic methodology for Spartan-II design consists of three interrelated steps: design entry,
implementation and verification. Industry-standard tools are used for design entry and simulation. Xilinx provides proprietary architecture-specific tools for implementation.

The program command sequence is generated prior to execution and stored for documentation. The Xilinx FPGA development system provides interfaces to several synthesis design environments. Electronic Design Interchange Format (EDIF) simplifies file transfers into and out of the development system. Spartan-II FPGAs is supported by a unified library of standard functions. The place and route tools (PAR) automatically provide the implementation. The PAR takes the EDIF net list for the design and maps the logic into the architectural resources of the FPGA. The placer determines the best locations for these blocks based on their interconnections and the desired performance. Finally, the router interconnects the blocks. The PAR algorithms support fully automatic implementation of most designs. User partitioning, placement and routing information are optionally specified during the design entry process. The implementation of highly structured designs can benefit from basic floor planning. Designers specify timing requirements along entire paths during design entry. The timing path analysis routines in PAR recognize these users specified requirements and accommodate them.

The development system supports both software simulations and in-circuit debugging techniques. For simulation, the system extracts the post-layout information from the design database. The user can verify timing-critical portions of the design using the static timing analyzer. After downloading the design into the FPGA, the designers can single step the logic, read back the contents of the flip-flops, and observe the internal logic state.

2.3.4 Complex Programmable Logic Devices

A PLA (programmable logic array) performs the same basic function as a ROM. The PLA structure consists of a bunch of AND gates, OR gates and inverters interconnected through some programmable switch arrays. The PAL (programmable array logic) is a special case of the PLA in which the AND array is programmable and the OR array is fixed. The basic structure of the PAL is the same as the PLA. The PAL is less expensive than PLA. The PAL is easier to program.
A CPLD is an IC that consists of a number of PAL like logic blocks together with a programmable interconnect matrix. Each PAL has a programmable AND array that feeds macro cells. The outputs of these macro cells can be routed to the inputs of other logic blocks within the same IC. Many CPLDs are electrically erasable and reprogrammable. They are referred as EPLDs (erasable PLDs). The Altera MAX 7000 series is a family of high performance CMOS CPLDs.

**Conclusion**

In summary, programmable logic devices allow the designers to:

- Build designs faster because manual wiring is minimized.
- Avoid mistakes caused by errors in wiring.
- Save the design for a longer period in electronic files and recall them whenever necessary. Easily correct design errors.
- Design larger projects because the tedious manual procedures are automated.

As FPGA architecture gives cost effective and faster solutions, it is suitable for implementing processor in FPGA rather than using CPLDs.