

REAL TIME MONITORING SYSTEM FOR AGRICULTURAL FIELD

Submitted in partial fulfilment of the requirements for the award of

**Bachelor of Engineering degree in
Electronics and Communication Engineering**

BY



KAMMARI KISHORE KUMAR-38130098

M.JY0THISWAR REDDY-38130123

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

**DEPARTMENT OF ELECTRONICS AND
COMMUNICATIONENGINEERING**

**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMEDTO BE UNIVERSITY)**

Accredited with Grade“A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI CHENNAI –600119.

MAY-2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that Project Report is the bonafide work of **K.KISHORE KUMAR** and **M.JYOTHISWAR REDDY** who carried out the Project entitled **“Real Time Monitoring System for Agricultural Field”**. Under the guidance of **V.VIJAYA KUMAR, Associate Professor, ECE** during the period DEC 2021 to MAY 2022

INTERNAL GUIDE

V.VIJAYA KUMAR, Associate Professor, ECE

HEAD OF THE DEPARTMENT

Dr. T. RAVI, M.E, PHD

Submitted for Viva Voce held on _____

Internal Examiner

External Examiner

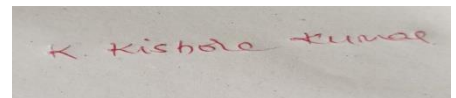
DECLARATION

We, **K.KISHORE KUMAR(3813098)** and **M. JYOTHSWAR REDDY (3813123)**, hereby declare that the Project Report entitled **Real Time Monitoring System for Agricultural Field** is done under the guidance of **Dr.V.VIJAYA KUMAR,Associate Professor,ECE**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

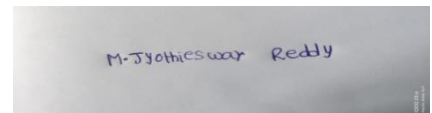
DATE:

PLACE:

1.



2.



SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my profound gratitude **to Dr. N.M. NANDHITHA, M.E., Ph.D., Prof.** & Dean, School of Electrical and Electronics **and Dr. T. RAVI, M.E., Ph.D.,** Head of the Department of **Electronics and Communication Engineering**, Sathyabama University for having been a constant source of support and encouragement for the completion of the project.

I consider my self-most fortunate and immensely privileged to have worked under the guidance of **Dr.V.VIJAYA KUMAR,Associate Professor** , Department of Electronics and Communication Engineering to complete this project successfully.

I express our heartfelt thanks to our family members and all our friends who helped me to complete our project successfully & finally, I thank the exciting Super Natural Power, the Almighty GOD.

ABSTRACT

The most important factors for the quality and productivity of plant growth are temperature, humidity, light and the level of the carbon dioxide. Continuous monitoring of these environmental variables gives information to the grower to better understand, how each factor affects growth and how to manage maximal crop productiveness. The optimal greenhouse climate adjustment can enable us to improve productivity and to achieve remarkable energy savings - especially during the winter in northern countries. The system itself was usually simple without opportunities to control locally heating, lights, ventilation or some other activity, which was affecting the greenhouse interior climate. This all has changed in the modern greenhouses. The typical size of the greenhouse itself is much bigger what it was before, and the greenhouse facilities provide several options to make local adjustments to the lights, ventilation, heating and other greenhouse support systems. However, more measurement data is also needed to make this kind of automation system work properly. Increased number of measurement points should not dramatically increase the automation system cost. It should also be possible to easily change the location of the measurement points according to the particular needs, which depend on the specific plant, on the possible changes in the external weather or greenhouse structure and on the plant placement in the greenhouse.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE No.
	ABSTRACT	5
1	INTRODUCTION	8
	1.1 OBJECTIVES	
2.	LITERATURE REVIEW	9-11
	2.1 LITERATURE SURVEY	
3.	EXISISTING SYSTEM	12
4.	PROPOSED SYSTEM	13-15
	4.1 RTL SCHMEMATIC DIAGRAM	
	4.2 FLOW CHART	
5.	SYSTEM DESIGN	16-30
	XILINX ARCHITECTURE TERMINOLOGY	
	5.1 BEL	
	5.2 SITE	
	5.3 TILE	
	5.4 FSR	
	5.5 SLR	
	5.6 DEVICE	
6.	IMPLEMENTATION BASICS	31-34
	6.1 PLACEMENT	
	6.2 ROUTING	

7	A PRE IMPLEMENTED MODULE FLOW	35-39
	7.1 RAPID PROTOTYPING FLOW	
	7.2 HIGH PERFORMANCE FLOW	
8	XILINX ISE SOFTWARE	40-43
	8.1 OVERVIEW	
	8.2 DESIGN FLOW	
	8.3 NAVIGATOR WINDOW SCREENSHOT	
	8.4 DEVICES TYPES	
9	OUTPUT SCREENSHOT	44
10	CONCLUSION	45
	REFERENCE	

LIST OF ABBREVIATIONS

FPGA	-	FIELD PROGRAMMABLE GATE ARRAY
BEL	-	BASIC ELEMENT OF LOGIC
FSR	-	FABRIC SUB REGION
SLR	-	SUPER LOGIC REGION
IMPL	-	IMPLEMENTATION
INST	-	INSTANCE
HDL	-	HARDWARE DISCRIPTION LANGUAGE
ISE	-	INTEGRATED SYNTHESIS ENVIRONMENT

CHAPTER 1

INTRODUCTION:

Implementation of agricultural technologies, low cost and real time remote monitoring are needed, in this sense, programmable Logic Devices (PLDs) present as a good option for the technology development and implementation, because PLDs allow fast development of prototypes and the design of complex hardware systems using FPGAs (Field Programmable Gate Arrays) and Complex Programmable Logic Devices. The contribution of this research is the development of a real-time remote monitoring system that acquires data from any kind of sensor to be transmitted by radiofrequency to a computer with an interface module. This allows the sensing of large area fields with a system capable of monitoring crop local environmental or physiological status; the data transmission and storage in the computer is made in real-time. To design this device, the system on a chip approach was followed. Implementation was done in a field programmable gate array, which ensures a low cost

1.1 OBJECTIVES

The prime objective is to collect the data from moisture sensors, temperature sensors of various areas of the field, water level sensors in the irrigation system to monitor the proper drip of water along with the fertilizer.

The objective of this research was to develop a monitoring system to obtain measurements of current field conditions in real-time. To identify the suitable pump with facility for maintaining certain recommended pressure in the water pipe. To identify proper sensors and monitoring device required for the farming data like soil moisture, soil temperature, soil fertilizer & chemical constituents.

CHAPTER 2

LITERATURE REVIEW

2.1 Literature survey:

1 FPGA Based Real Time Monitoring System for Agricultural Field, M. Dinesh, P. Saravanan

The greenhouse automation at commercial level is experiencing attention. Also, to achieve competitiveness in the market, the production costs must be kept as low as possible. Low cost automation can be achieved by using VLSI systems so that all category farmers can afford it. The objective of this project was to develop a system to provide autonomous control for temperature and humidity in a closed environment of Greenhouse, which is fulfilled

2 FPGA BASED REAL TIME MONITORING SYSTEM FOR AGRICULTURAL FIELD, Prerana Sarwade¹ , Neeta Shinde² , Shweta Tingre³

Real-time monitoring provides reliable, timely information of crop and soil status, important in taking decisions for crop production improvement. The contribution of this research is the development of a realtime remote monitoring system that acquires data from any kind of sensor to be transmitted by radiofrequency to a computer with an interface module. This allows the sensing of large area fields with a system capable of monitoring crop local environmental or physiological status; the data transmission and storage in the computer is made in realtime. To design this device, the system on a chip approach was followed. Implementation was done in a field programmable gate array, which ensures a low cost.

3 FPGA-based real-time remote monitoring system, Joshua Mendoza-Jasso, Gerardo Ornelas-Vargas, Autonomous University of Zacatecas, Rodrigo Castañeda-Miranda, Eusebio Ventura

Real-time monitoring provides reliable, timely information of crop and soil status, important in taking decisions for crop production improvement. The contribution of this research is the development of a real-time remote monitoring system that acquires data from any kind of sensor to be transmitted by radiofrequency to a computer with an interface module, situated within a 900 m radius. This allows the sensing of large area fields with a system capable of monitoring crop local environmental or physiological status; the data transmission and storage in the computer is made in real-time. To design this device, the system on a chip approach was followed. Implementation was done in a field programmable gate array, which ensures a low cost. The performance of this system was tested using different kinds of sensors and compared with various commercial monitoring systems under greenhouse conditions. The experimental results showed the system to be reliable. For all experiments, the system obtained an R^2 greater than or equal 0.975 in a regression analysis between data acquired from our monitoring system and data obtained from a commercial datalogger with linear fit and second-degree polynomial fit.

4 Real-Time FPGA-Based Systems to Remote Monitoring.

Some industrial and laboratory applications such as control, monitoring, test and measurements, and automation require real-time systems for their development. Embedded systems for acquisition and processing often require the participation of the embedded operating system and therefore are necessary techniques that can accelerate software execution. The latest field-programmable gate arrays' (FPGA) technology has blurred the distinction between hardware and software with embedded processors that allow the development of Systems-on-a-Chip (SoC) running on operating systems. The widespread adoption of wireless technologies such as Bluetooth, ZigBee, and Wi-Fi in the last years has facilitated the use of these technologies to the development of real-time monitoring applications that combined with FPGA devices which has the advantages of low cost, flexibility, and scalability as compared with other commercial systems.

CHAPTER 3

Existing system:

The system monitored different agricultural parameters and transmitted them wirelessly to central unit via Microcontroller. The real time values of different parameters were displayed at both transmitter and receivers side. Sensors used were tested for different environmental condition like changing the temperature across the fields. Also the wireless module was examined against different obstacles. It was found that system works well in transmitting real time values wirelessly to remotely placed receiver. It consists of sensing unit, LCD, relay, water pump and microcontroller. Sensing unit read the different atmospheric conditions. It consists of temperature sensor, relative humidity sensor and soil moisture sensor. The readings are given to microcontroller. Microcontroller will display these reading on LCD as well as transmit it through wireless module.

CHAPTER 4

Proposed system:

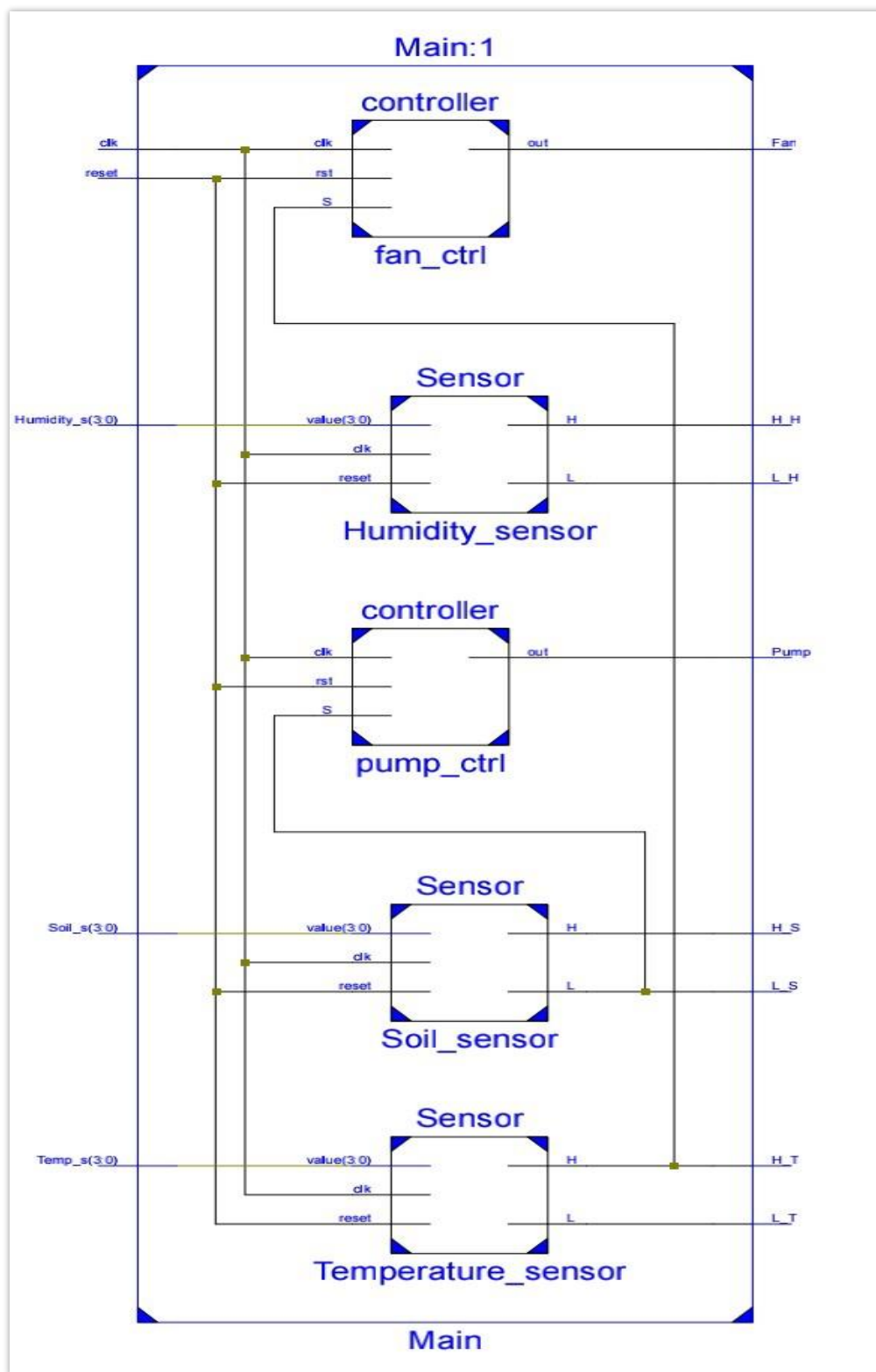
The proposed system is an embedded system which will closely monitor and control the microclimatic parameters of a greenhouse on a regular basis round the clock for cultivation of crops or specific plant species which could maximize their production over the whole crop growth season and to eliminate the difficulties involved in the system by reducing human intervention to the best possible extent.

The system comprises of

- **Sensors**
- **FPGA**

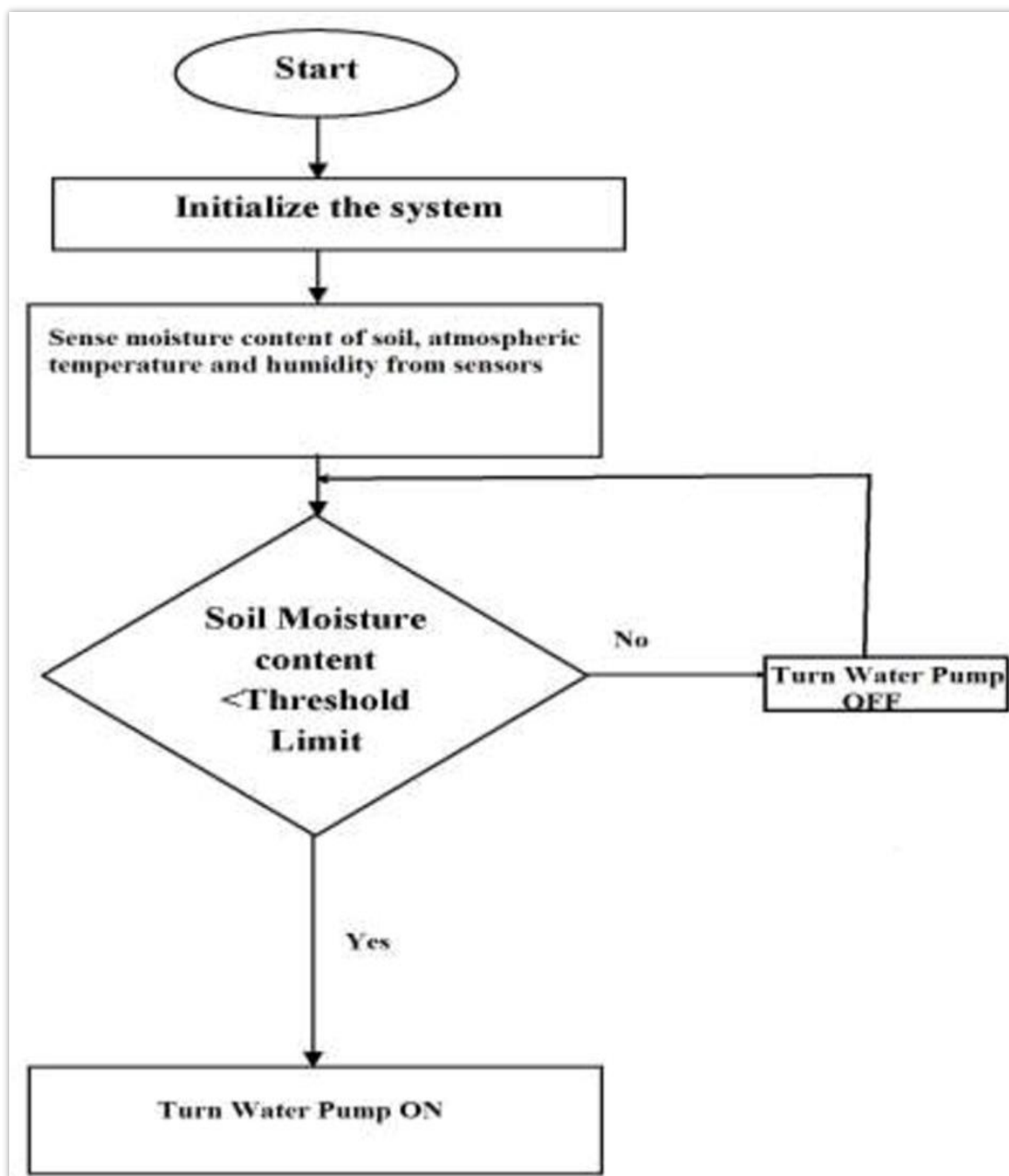
When any of the above mentioned climatic parameters cross a safety threshold which has to be maintained to protect the crops, the sensors sense the change and the fpga reads this from the data at its input ports after being converted to a digital form by the ADC. The fpga then performs the needed actions by employing relays until the strayed-out parameter has been brought back to its optimum level. Since a fpga is used as the heart of the system, it makes the set-up low-cost and effective nevertheless. As the system also employs an LCD display for continuously alerting the user about the condition inside the greenhouse, the entire set-up becomes user friendly.

4.1 Block diagram:



RTL
schematic
diagram:

4.2 Flow chart:



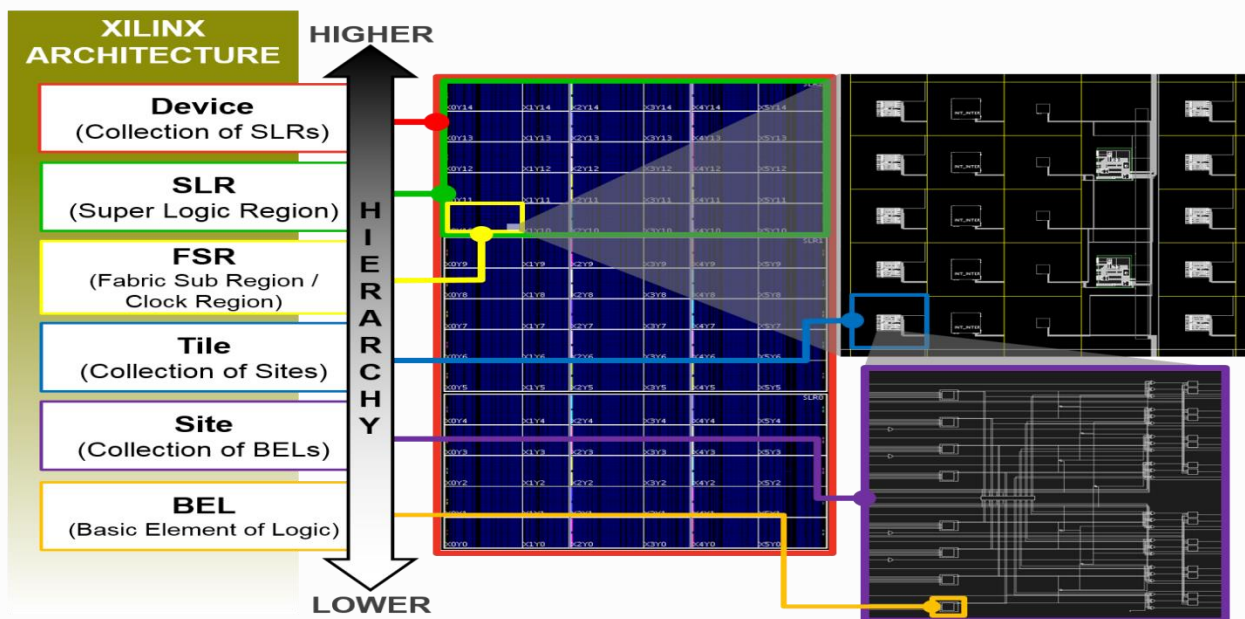
CHAPTER -5

System design:

• XILINX ARCHITECTURE TERMINOLOGY

- BEL (Basic Element of Logic)
- Site
- Tile
- FSR (Fabric Sub Region or Clock Region)
- SLR (Super Logic Region)
- Device

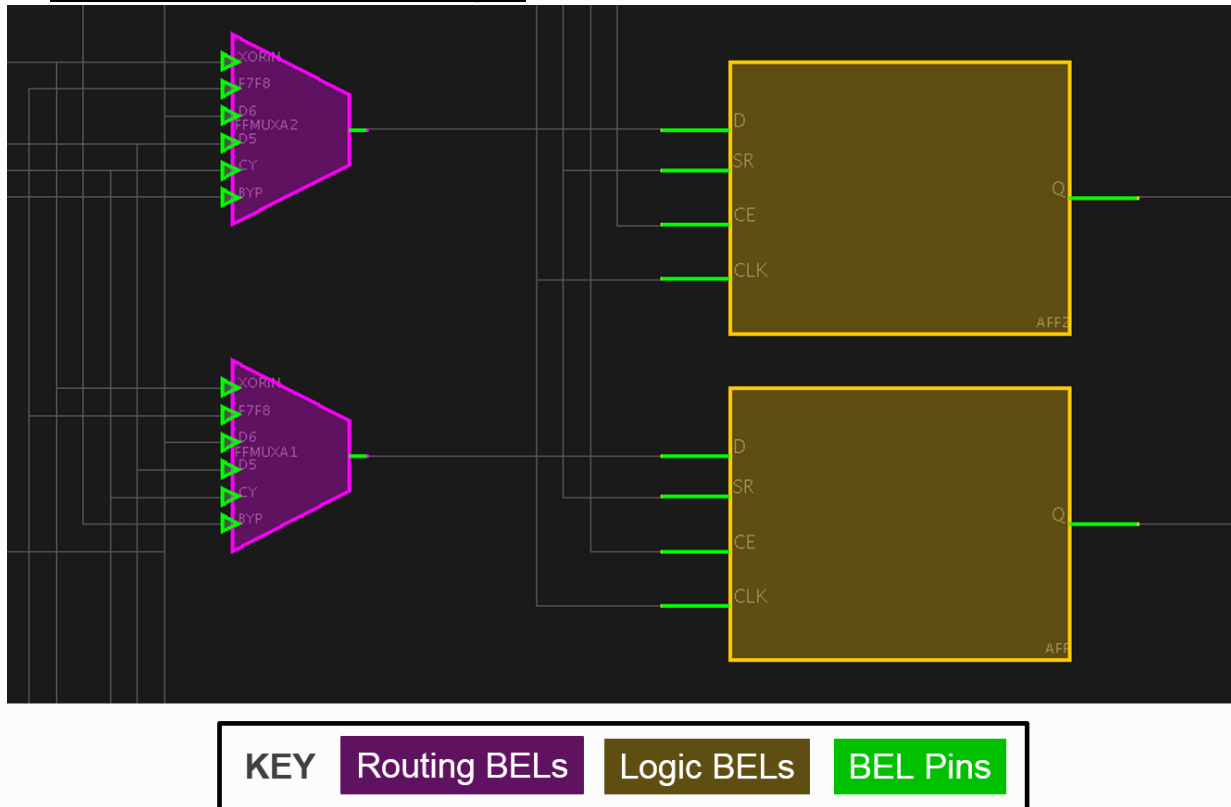
In order to use RapidWright, an understanding of Xilinx FPGA architecture and hierarchy will be necessary in navigating your way around the device APIs. In Xilinx FPGAs, there are six major levels of hierarchy that describe basic components all the way up to the entire device. This hierarchy can be seen in the figure below:



Levels of architectural hierarchy in Xilinx FPGAs.

We begin our discussion with a bottom-up approach starting with the lowest level of hierarchy, the basic element of logic.

5.1 BEL (Basic Element of Logic)



Vivado representation of two routing muxes (routing BELs) and two flip flops (logic BELs).

At the lowest level, the atomic unit of Xilinx FPGAs is a BEL. BELs are the smallest, indivisible, representable component in the fabric of an FPGA. There are two kinds of BELs, Logic BELs (Basic Element of Logic) and Routing BELs. A Logic BEL is a configurable logic-based site that can support the implementation of a design cell. Each BEL can support one or more types of UNISIM cells (UNISIM cells are described in Libraries Guides [UG953](#) for Series 7 devices and [UG974](#) for UltraScale™ devices). The mapping between a leaf cell (non-leaf cells do not represent implementable hardware, just hierarchy) in the netlist and a BEL site is referred to as the ‘placement’ of the cell. Thus, when one runs the Vivado command `place_design`, it is essentially mapping all leaf cells in the netlist to compatible and legal BEL sites.

Routing BELs are programmable routing muxes used to route signals between BELs. Routing BELs do not support any design elements (logic cells from the netlist do not

occupy routing BEL sites), they are used only for routing. However, some routing BELs do have optional inversions.

BELs have input and output pins. BELs also have configurable connections that connect an input pin to an output pin. These BEL-based configurable connections are called site PIPs (where PIP stands for Programmable Interconnect Point). Both logic BELs and routing BELs can have site PIPs.

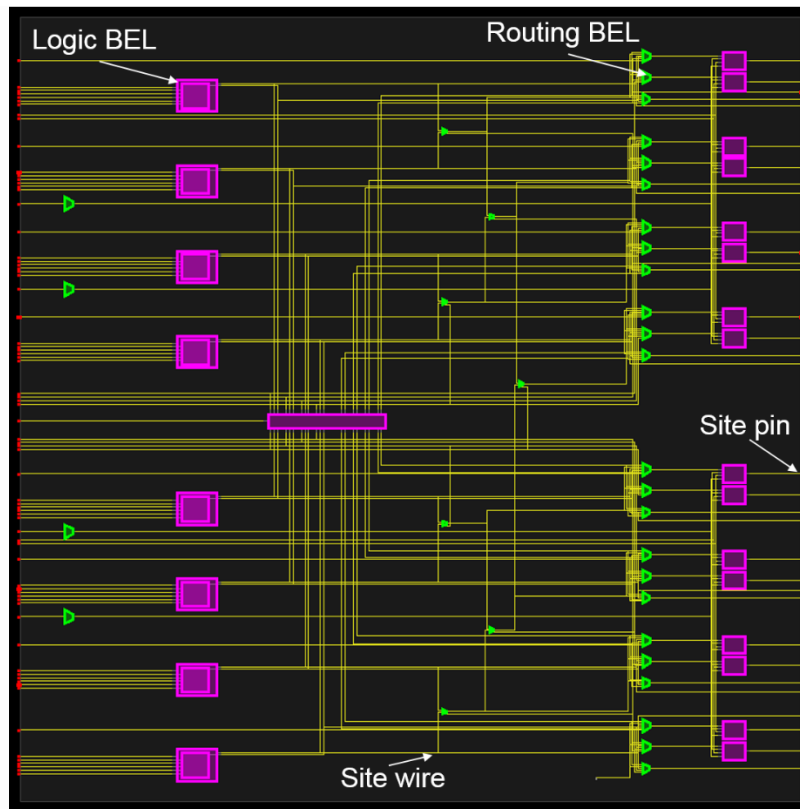
However, in the case of a logic BEL, the site must be unoccupied by a cell in order for the route through to be usable. Often, these site PIPs, when implemented in logic BELs (a LUT is a common example), are referred to as a “route through” or “route-thru.”

When routing a design, in order to physically route a net it is sometimes necessary to route through unused LUTs or other logic BELs with site PIPs.

5.2 Site

A group of related elements and their connectivity is referred to as a site. Inside of a site, one can find three major categories of objects:

1. BELs (Logic BELs and/or Routing BELs)
2. Site Pins (External input and output pins to the site)
3. Site wires (connecting elements to each other and site pins)



An UltraScale+ SLICEL site, where logic BELs are magenta, routing BELs are green, site pins are red and site wires are yellow.

Sites are instances of a type and each site has a unique name with an `_X#Y#` suffix denoting its location in the site type grid. Each site type will have its own XY coordinate grid, independent of others. The only exception to this is that SLICEL and SLICEM types share the same grid space. SLICEL and SLICEM are the most common site type and are the basic configurable logic building blocks that contain LUTs and flip flops that form the backbone of the FPGA fabric.

Site Type

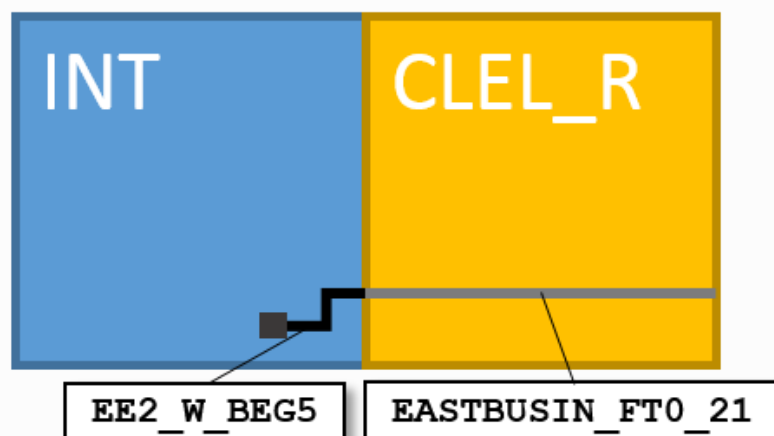
Sites are heavily replicated across the device and each instance of a site corresponds to a site type of that device's architecture family. Additionally, sites found in an FPGA device are sometimes capable of hosting different types, however, when a tile is queried, a 'primary' site type is designated.

5.3 Tile

At an abstract level, Xilinx devices are created by assembling a grid of tiles. Similar to sites, each tile is an instance of a type and each tile has a unique name with

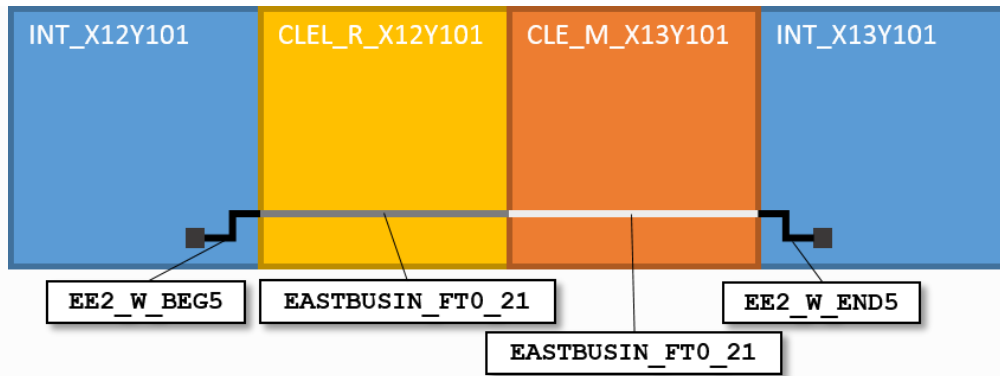
an `_X#Y#` suffix. Tiles are the building blocks used when constructing an FPGA device. Tiles are designed to abut one another when laid down to construct an FPGA device.

Not all tiles contain sites and those that do, can have more than one. Unlike sites and BELs, tiles do not have user visible pins. Instead, tiles contain uniquely-named wires that can connect to site pins or other wires through a programmable interconnect point (PIP). PIPs are programmable muxes that connect two wires together in the same tile. Most PIPs are present in switch box tiles (those with the “INT” prefix). Columns of switch box tiles are designed to connect to all fabric resources such as CLBs, DSPs, and BRAMs. When tiles abut, they are designed such that certain wires in the adjoining tiles line up and connect as shown in the figure below:



Node

As there are no pins on tiles, the notion of a node is used to describe the connectivity of wires in between tiles. A node is a collection of electrically connected wires that spans one or more tiles. The figure below shows how four wires that abut among four tiles form a node:



Nodes and wires exist as first class Tcl objects in Vivado and the example above can be queried as follows:

```
%get_wires-of[get_nodeINT_X12Y101/EE2_W_BEG5]
INT_X12Y101/EE2_W_BEG5INT_X13Y101/EE2_W_END5CLEL_R_X12Y101/EAS
TBUSIN_FT0_21CLE_M_X13Y101/EASTBUSIN_FT0_21
%
```

For additional resources regarding Vivado objects, see [UG912: Vivado Design Suite Properties Reference Guide](#).

Tile Type

Each tile belongs to a type or definition. A tile type will contain the inventory list of all wires, PIPs and site types. Vivado does not directly represent the tile type as an object, but is listed as a property value under each tile.

Xilinx traditionally has leveraged a columnar-based architectural approach to tile layout. That is, with a few exceptions, all tiles within a column are of the same type but tiles occupying the same row are typically different types.

5.4 FSR (Fabric Sub Region or Clock Region)

A fabric sub region, also known as a clock region, is a replicated 2D array of tiles in the fabric. In the UltraScale architecture, all FSRs are 60 CLBs tall, but their width will vary depending on the mix of tile types used in its construction.

Clock routing and distribution lines are represented as the same granularity as FSRs. In UltraScale architectures, there are 24 horizontal routing tracks, 24 vertical routing tracks, 24 horizontal distribution tracks and 24 vertical distribution tracks. These routing and distribution tracks abut to tracks in neighboring FSRs to form the device clock network resource set. For more information specific to clocking resources.

5.5 SLR (Super Logic Region)

This level of hierarchy is only present on devices that use the stacked silicon interconnect technology (SSIT) or also known as 2.5D packaging using a silicon interposer. As multiple dies (or dice) are packaged together, each die becomes a super logic region or SLR. SLRs contain a 2D array of FSRs and are typically identical as each die is fabricated from the same mask set.

In order for logic to communicate between SLRs, the UltraScale architecture employ special tiles in the FSRs neighbouring the abutment of two SLRs. A column of CLBs is removed and replaced with special tiles called Laguna tiles that have dedicated flip flop sites to aid in crossing the SLR divide.

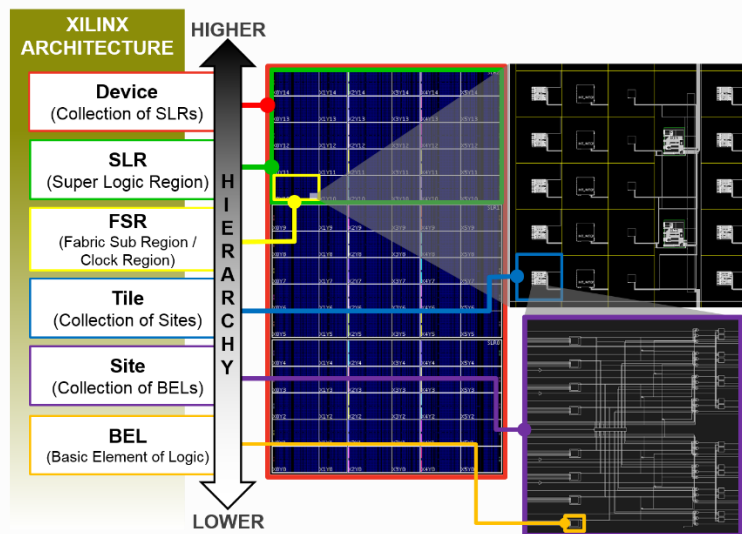
5.6 Device

At the highest level of Xilinx architecture is the device. This is generally a 2D array of FSRs for single die products or two or more SLRs abutted vertically.

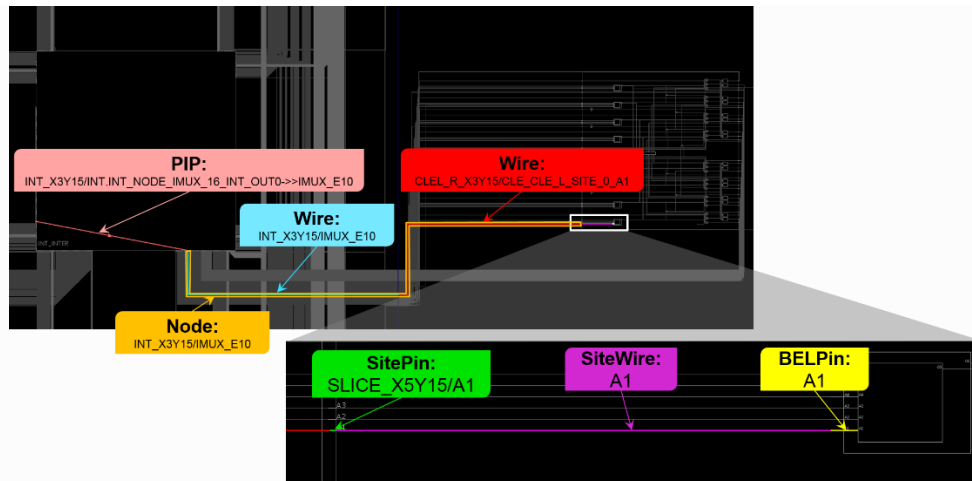
Most of the details of managing speed grades, packages, temperature are most commonly dealt with by using a string to uniquely identify a part is by using a String of the part name. RapidWright automatically interprets all valid and supported Xilinx devices by part name.

These levels of hierarchy are available in RapidWright and the table below shows basic getters in both RapidWright and Vivado.

RapidWright Class	RapidWright Java API	Vivado Object	VivadoTcl API
SLR	Device.getSLR(int id)	SLR	get_slrs -filter SLR_INDEX==\$idx
ClockRegion	Device.getClockRegion(int row,int col)	Clock Region	get_clock_regions -filter NAME==\$name
Tile	Device.getTile(String name)	Tile	get_tiles -filter NAME==\$name
Site	Device.getSite(String name)	Site	get_sites -filter NAME==\$name
BEL	Site.getBEL(String name)	BEL	get_bels -of \$site -filter NAME==\$name



Levels of architectural hierarchy in Xilinx FPGAs.



Examples of different routing resources Xilinx FPGAs.

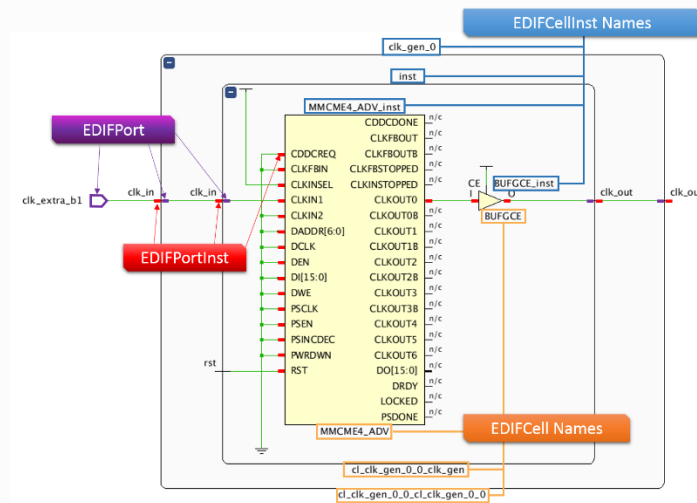
A. EDIF Package (Logical Netlist)

In Vivado, all designs post synthesis have a logical netlist that can be exported in the EDIF netlist format. EDIF (Electronic Design Interchange Format) 2 0 0 is the netlist format used in RapidWright. This is due to its inclusion in Vivado's design checkpoint file format and that Vivado has facilities to read and write it (`read_edif` and `write_edif`).

RapidWright reads, represents and writes logical netlist information in the EDIF format and the EDIF package is written to explicitly accommodate this need. It was written with Vivado-generated EDIF in mind and may not support every corner case of the EDIF 2 0 0 specification.

The `EDIFNetlist` is the top level class that contains the netlist and cell libraries. All EDIF-related objects have EDIF as a class name prefix. The `EDIFNetlist` keeps a reference to the top cell which is wrapped in the `EDIFDesign` class. It also maintains a top cell instance reference that is generated when the file is loaded.

Another distinction is that an `EDIFPort` can be a bussed-based object whereas an `EDIFPortInst` can only represent a single bit. An `EDIFNet` defines connectivity inside an `EDIFCell` by connecting `EDIFPortInst` objects together (port references on cell instances inside the cell or to external port references entering/leaving the cell).



Snapshot of the Vivado netlist viewer with references to RapidWright EDIF classes

Several classes also inherit from `EDIFPropertyObject` (which also inherits from `EDIFName`). `EDIFPropertyObject` endows objects with the ability to store properties which are key/value pairs. Properties are a mapping between an `EDIFName` object and a `EDIFPropertyObject`. These properties can contain key programmable information such as LUT equations or attributes specific to BEL sites.

B. Design Package (Physical Netlist)

The design package is the collection of objects used to describe how a logical netlist maps to the device netlist. The design is also referred to as the physical netlist or implementation. It contains all of the primitive logical cell mappings to hardware, specifically the cells to BEL placements and physical net mapping to programmable interconnect or routing.

The `Design` class in RapidWright is the central hub of information for a design. It keeps track of the logical netlist, physical netlist, constraints, the device and part references among other things. The `Design` class is most similar to a design checkpoint in that it contains all the information necessary to create a DCP file.

Since a design programs a device, there are some one-to-one mappings between the device and design representation in RapidWright. For example:

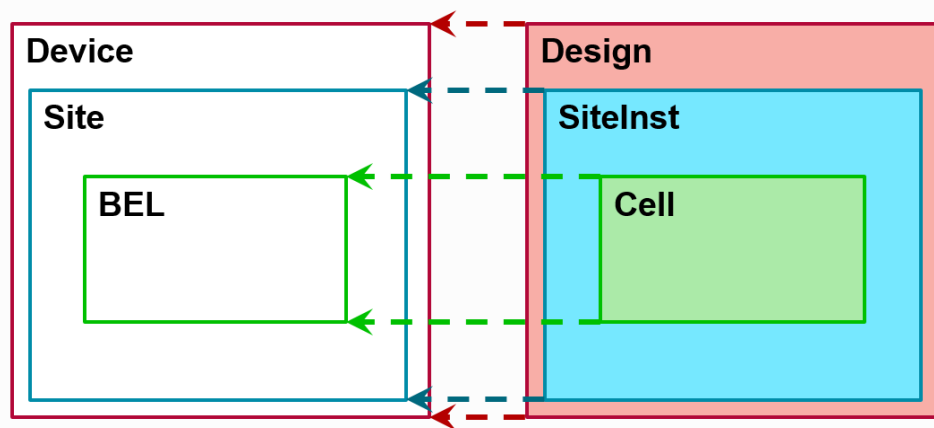


Illustration representing how a `Cell`, `SiteInst` and `Design` map to `BEL`, `Site` and `Device` respectively

C. `SiteInst`

Design representation and implementation in Vivado is BEL-centric (BELs and cells). The `SiteInst` keeps track of the cells placed onto its BELs, the site PIPs used in routing and how routing resources map to nets.

Each `SiteInst` maps to a specific compatible site within a device. The `SiteInst` has a type using a `SiteTypeEnum` as the designator. It also maintains a map of named leaf cells from the logical netlist that are physically placed onto the BEL sites within the site. RapidWright also preserves the same Vivado “fixed” flag that is used in certain situations by Vivado to prevent components inside the site from being moved.

Routing nets inside of a site (intra-site) is different from routing outside of sites (inter-site). Routing nets outside of sites consists of finding a path of `Node` objects from a source site pin to a sink site pin by turning on a set of PIPs. In contrast, routing inside of a site can be a bit more complex as it must also account for site context and consider which BELs are occupied. In general, Vivado attempts to automate the intra-site routing task. RapidWright also strives to do the same (see `SiteInst.routeSite()`), however it may not always fully automate tasks as expected and the user may be required to call additional APIs when placing/routing design elements.

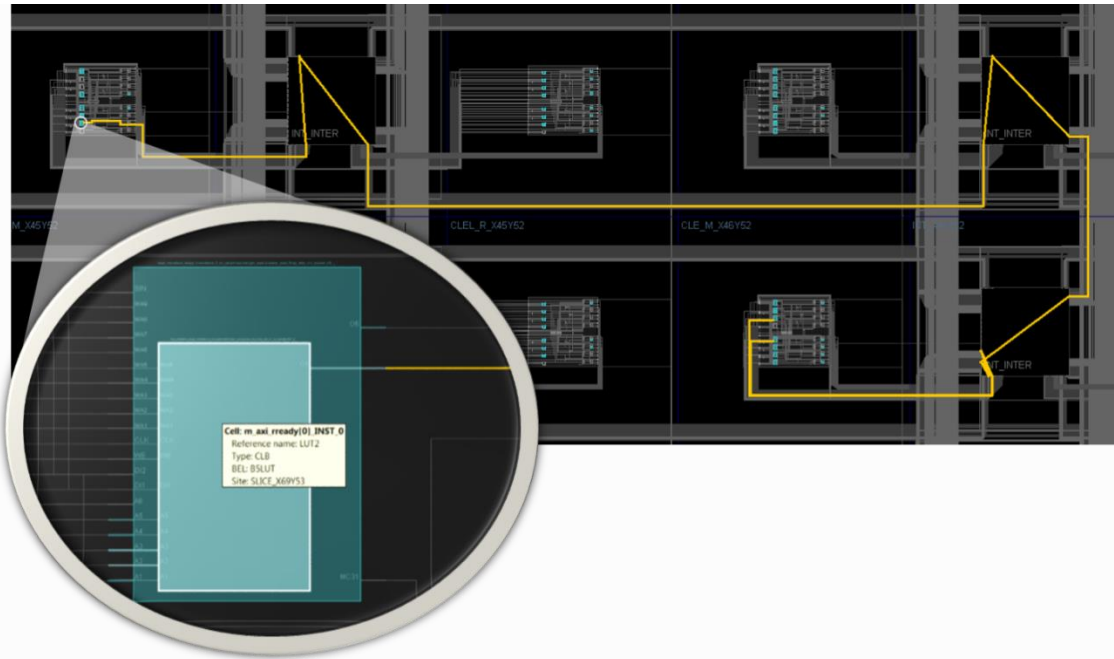
Generally, a `SitePIP` will establish a connection through a routing BEL or, in some cases, a logic BEL from an element input pin to an element output pin, thus connecting two separate site wires.

.

D. Net

Routing outside of a site is represented by the `Net` class. A `Net` in RapidWright is typically named after the logical driver source pin and represents the entire set of logically equivalent nets that map to the same electrically equivalent net.

A `Net` is a physical net that implements a route using PIPs (programmable interconnect points) that, when combined together connect nodes into a path from a source site pin to one or more sink site pins. A `Net` starts and stops at site pins represented by `SitePinInst` objects (design instances of `SitePin` objects). The physical implementation of the 11 logical nets above is shown in the figure below:



The net is also referenced when routing inside a site, but the site routing implementation is captured in the `SiteInst` object.

E. Cell (A BEL Instance)

At the lowest level, a RapidWright `Cell` maps a logical leaf cell from the EDIF netlist (`EDIFCellInst`) to a `BEL`. The cell name is typically the full hierarchical logical name of the leaf cell it maps to and also maintains the library cell type name (FDRE, for example for a reset flip flop). A cell also maintains the logical cell pin mappings to the physical cell pin mappings (pins on the `BEL`).

F. Module

A module is a physical netlist container construct available in RapidWright. A RapidWright module is represented by the `Module` class in the `design` package. A module contains both a logical and physical netlist that provides all the details necessary for a full implementation.

It is most similar to a placed and routed out-of-context DCP, however RapidWright enables the implementation to be replicated or relocated to multiple compatible areas of the fabric—capabilities that are not yet available in Vivado. A module is a definition object in that the `SiteInst` and `Net` objects it contains are a prototype or blueprint for a pre-implemented block that can potentially be ‘stamped’ out and relocated in valid locations around a device. The `ModuleInst` represents the instance object of a `Module` and is part of the implemented portion of a physical netlist.

G. Module Instance

A module instance quite simply is an instance of a module, instance name. For example, if a module had a `SiteInst` named “SLICE_X2Y2” and `Net` named `data_ready`, a newly created module instance named “fred” would have counterpart `SiteInst` and `Net` objects called “fred/SLICE_X2Y2” and “fred/data_ready”.

A module instance will typically have one of its site instances selected as what is called an ‘anchor’. The anchor site instance is a common reference point by which all other site instances and nets in the instance can be referenced. This is useful for determining if a potential location on the fabric is compatible with the module instance for placement.

CHAPTER 6

Implementation Basics

- Implementation Basics
 - Placement
 - Routing

Implementation, in the context of RapidWright and compiling designs for FPGAs, is defined as the placement and routing of a synthesized/mapped netlist to a specific FPGA device. This section will describe the detailed mechanics of how placement and routing can be achieved in RapidWright.

6.1 Placement

As opposed to Vivado, RapidWright enables three layers or levels of placement in its design abstraction: BEL level, site level and module level. Vivado primarily only enables BEL placement (previously in ISE, sites were the major unit of placement). This section details how RapidWright represents and interacts with design elements at the three levels of placement mentioned.

A. BEL Placement

A SitePIP configures a routing BEL to propagate a signal from one of its inputs to its output pin. SitePIPs must be turned on in the respective `SiteInst` when a cell is placed onto a BEL as the common convention in Vivado is to always leave the site in a legally routed state.

B. Site Placement

Within RapidWright, it can be straightforward to move a `SiteInst` from one site to another. An example of how to relocate a site instance from one location to another is shown below:

```
Design d = Design.readCheckpoint("example.dcp");  
SiteInstance si = d.getSiteInstanceFromSiteName("SLICE_X0Y0");  
si.place(d.getDevice().getSite("SLICE_X1Y1"));
```

The user is responsible for changing any existing routing resources that previously routed to the old site.

C. Module Placement

One of RapidWright's unique capabilities is providing another level of hierarchy in implementation. Through the `Module` and `ModuleInstance` classes, a complex cell can be replicated and/or relocated across the device. When a pre-implemented module is created for a device, all valid locations are pre-calculated and stored for the anchor site within the `Module`. Therefore, placement of a `ModuleInstance` is simply selecting one of the valid anchor sites and applying it.

6.2 Routing

In Vivado, there is roughly three different types of routing: intra-site, inter-site and clock routing. This section provides a brief overview of each.

A. Site (Intra-site) Routing

When a cell is placed onto a BEL, typical Vivado convention is to route the intra-site net portions immediately after. Routing a site implies mapping the physical net to site wires and site PIPs. In RapidWright, some of this intra-site routing happens when the cell is placed and there are a few methods that can also help finish intra-site routing in special cases. `SiteInst.routeIntraSiteNet()` will attempt to route one BELPin to another for intra-site nets. `SiteInst.routeSite()` will attempt to route all the nets that pertain to the site.

B. Interconnect (Inter-site) Routing

The majority of work in routing a design is in inter-site routing. This is the task of selecting a set of routing resources that enable a path between a source site pin and one or more sink site pins. The physical routing of a net in RapidWright is simply described by a list of PIPs. RapidWright comes with a rudimentary router for UltraScale architectures, but it is still a work in progress. It doesn't fully resolve congestion, but provides a working example for more specialized tasks.

C. Clock Routing

Clock routing is very architecture specific and is similar to inter-site routing in that it is also implemented by a list of PIPs. However, there are key steps and constraints that must be satisfied beyond typical inter-site routing.

D. RWRoute

RWRoute is a full design router that has been developed in the RapidWright framework leveraging its lightweight timing model. It is capable of routing designs in both wirelength-driven and timing-driven modes, enabling the open source community to innovate and develop new algorithms. The open source aspect enables creation of domain-specific algorithms such as bundle routing in customized cost functions for the desired figure of merit. It also supports a partial routing mode, which is an essential capability for a future library-based customized flow.

RWRoute has some limitations:

1. It currently only supports UltraScale+ devices.
2. The timing model in RapidWright does not estimate hold time and thus RWRoute cannot address hold time violations.
3. For the most accurate clock routing in timing-driven mode, certain files will be needed to be created (see `tcl/rwroute/README` for more information).

4. When attempting to route designs in timing-driven mode, for the most accurate timing estimates on hard blocks (such as DSPs), the design must be pre-analyzed and a set of files must be created to feed into RWRRoute (see tcl/rwroute/README for more information).

.

Bitstream Manipulation

RapidWright cannot generate bitstreams on its own. It is necessary to create bitstreams using Vivado. RapidWright does not contain the information needed to translate a placed and routed design into a bitstream. RapidWright also has no encryption/decryption capabilities and will not be able to parse any bitstreams successfully that are encrypted. As with any files generated by RapidWright, they are not warranted and it is intended as an experimental platform only.

CHAPTER 7

A Pre-implemented Module Flow

This section describes a pre-implemented module flow that can operate in two ways:

1. Target high performance implementations by reusing high quality, customized solutions.
2. A rapid prototyping demonstration vehicle that hints at a future of fast compile times.

Background and Flow Comparison

	High Performance Flow	Rapid Prototyping Flow
PBlock Selection	Application Architect (Manual)	PBlock Generator
Block Placement	Application Architect (Manual)	Block Placer
Global Routing	Vivado	RapidWright Router OR Vivado

The high performance flow (as described in more detail in the High Performance Flow section below) requires input from the application architect of the design. This does involve extra effort, but leads to potentially the highest implementation results.

7.1 Rapid Prototyping Flow

It is optimized more for fast compile times by automating the tasks of pblock selection for each block/IP involved and also placement of the blocks.

A. Module Cache

In order to better facilitate fast loading performance of modules, RapidWright has a fast and efficient file format for storing modules in a directory called a cache. The facilities for reading and writing these module storage files are found in the `BlockCreator` class found in the `ipi` package. As each IP to be implemented in a design might have different physical contexts or placement pblocks, multiple implementations of the

same `Module` are stored in a `ModuleImpls` object which is simply an extended `ArrayList<Module>`. This allows all the implementations to reside in the same object and file and to reference each unique implementation with an index.

Each RapidWright module entry has three relevant files:

1. Input: A metadata text file generated from Vivado to communicate information about the IP, its ports, clocks, constraints and approximate delays on inputs and outputs. This file is read into RapidWright during the module file creation process.
2. Output: To store the physical implementation data of each module implementation, a '.dat' file is created from `BlockCreator`.
3. Output: The logical netlist is shared among all implementations and is stored in a compressed EDIF file format with a '.kryo' extension.

B. Block Stitcher

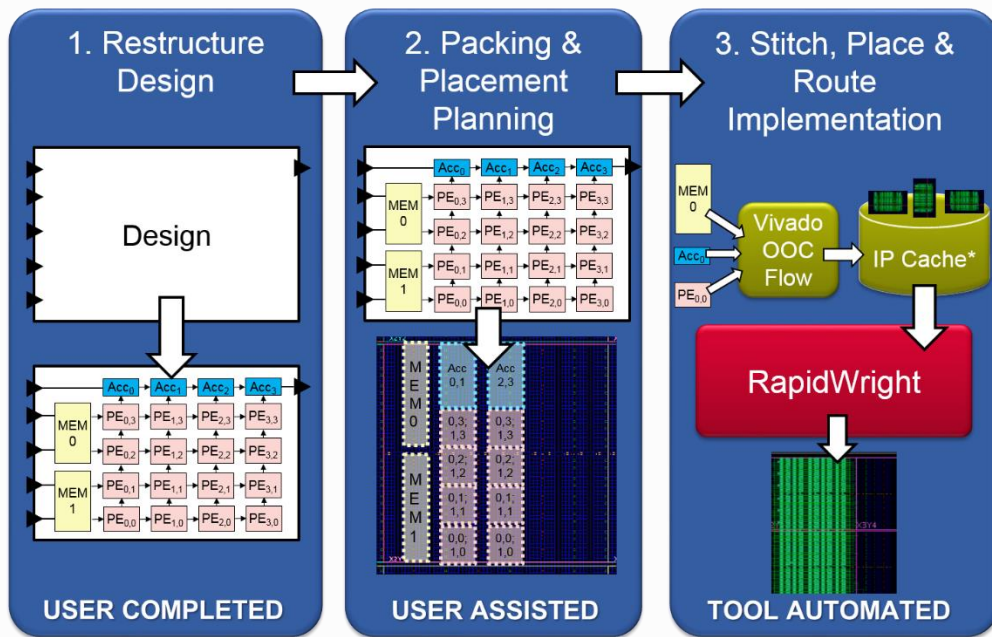
The block stitcher (found in the class `BlockStitcher` of the `ipi` package) is the heart of the pre-implemented design flow. It manages the flow progress and ensures that all blocks have been cached and retrieved appropriately. It also reads in the IP Integrator netlist file (EDIF) that describes the block connectivity and stitches together the block implementations in the physical netlist. It also reads and parses the implementation guide file (if provided) and creates the block implementations accordingly.

7.2 High Performance Flow

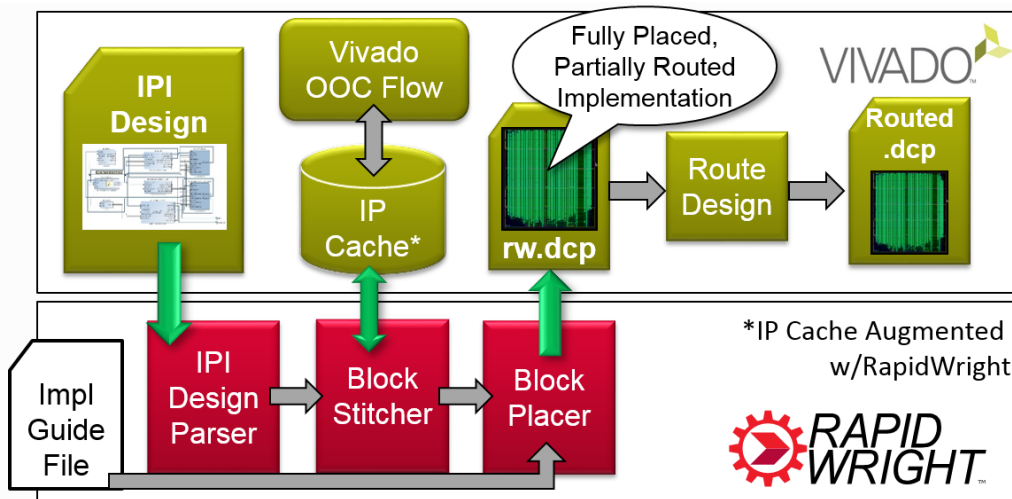
One of the key attributes of RapidWright is the ability to capture optimized placement and routing solutions for a module and reuse them in multiple contexts or locations on a device. Vivado often provides good results for small implementation problems (smaller than 10k LUTs within a clock region). However, when those same modules are combined into a large system, total compile time increases and the probability of timing closure is reduced. This phenomenon limits achievable performance and timing closure predictability of larger designs.

RapidWright endows users with a new design vocabulary by caching, reusing and relocating pre-implemented blocks. We believe this to be an enabling concept and offer a three-step high performance design strategy:

1. Restructure the Design: Expose all modular pieces and replication in an IP Integrator design.
2. Packing & Placement Planning: Craft custom pblocks and placement patterns to match architecture layout and resources.
3. Stitch, Place & Route Implementation: Run the automated flow to create a final implementation.



High level visual of the three step process for the high performance module-based design strategy



High level view of the pre-implemented flow process and interactions between Vivado and RapidWright

A. BLOCK (IP Cache Entry)

The block construct describes all of the potential implementations for a particular block/IP. For each uniquely configured IP (entry in the IP cache), there exists a block. Multiple instances of the same block/IP can exist and this construct allows the application architect to map instances by name to a specific implementation.

B. IMPL (Implementation)

Each block has one or more IMPLs. Each implementation carries a pblock and potentially some SUB_IMPL which allows for sub pblocks to be applied to portions of the logic inside the block. Each IMPL is indexed so that it can be referenced and applied to specific instances of the block. The application architect takes special care in selecting implementations and their pblocks to maximize their potential performance, architectural footprint and placement packing efficiency.

C. SUB_IMPL (Sub Implementation)

This is an optional construct that allows for more fine-grained pblocks being applied to a partial subset of the block/IP in an implementation. One field requires a Tcl command that returns a subset of cells that should be included in the sub implementation and associated pblock. Multiple sub implementation entries can exist for each implementation. As an example, if a particular IP is tall and narrow and there are specific

cells that need to be placed at the top and/or bottom, the SUB_IMPL construct can be used to pblock the top and bottom specific cells in sub pblock of the overall implementation.

D. *INST (Instance)*

In each design, there will be one or more instances of a block/IP. Each instance has a unique name and must be assigned to an implementation. Each instance also requires a placement which is provided by denoting a specific site onto which the lower left corner of the pblock of the respective implementation could be placed.

E. *CLOCK (Clock Input)*

The clock construct describes a clock input to the block or IP and allows it to apply a clock period constraint in nanoseconds. It also requires the BUFGCE site from which the clock will be driven so that during placement and routing, the clock skew can be estimated.

CHAPTER 8

Xilinx ISE Overview

8.1 OVERVIEW

The Xilinx ISE system is an integrated design environment that consists of a set of programs to create (capture), simulate and implement digital designs in a FPGA or CPLD target device. All the tools use a graphical user interface (GUI) that allows all programs to be executed from toolbars, menus or icons. On-line help is available from most windows.

8.2 Design Flow Overview –The following steps are involved in the realization of a digital system using Xilinx FPGAs, as illustrated by the following figure.

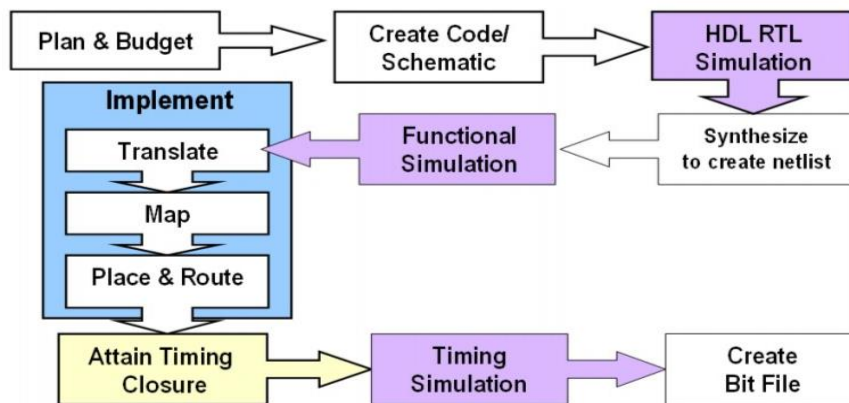


Figure: Overview of the various steps involved in the design flow of a digital system

A. Design Entry

The first step is to enter y our design. This can be done by creating “Source” files. Source files can be created in different formats such as a schematic, or a Hardware Description Language (HDL) such as VHDL, Verilog or ABEL. A project design will consist of a top-level source file and various lower-level source files. Any of these files can be either a schematic or a HDL file

B. Design Synthesis

The synthesis step creates netlist files from the various source files. The netlist files can serve as input to the implementation module.

C. Design Verification (simulation)

This is an important step that should be done at various stages of the design. The simulator is used to verify the functionality of a design (functional simulation), the behavior and the timing (timing simulation) of your circuit. Timing simulation is run after implementing your circuit in the FPGA since it needs to know the actual placement and routing to find out the exact speed and timing of the circuit.

D. Design Implementation

After generating the netlist file (synthesis step), the implementation will convert the logic design into a physical file that can be downloaded on the target device (e.g. Virtex FPGA). This steps involves three sub-steps: Translating the netlist, Mapping and Place&Route.

E. Device Configuration

This refers to the actual programming of the target FPGA by downloading the programming file to the Xilinx FPGA.

8.3 Project Navigator Window: The above steps are managed through a central ISE Project Navigator window, shown below

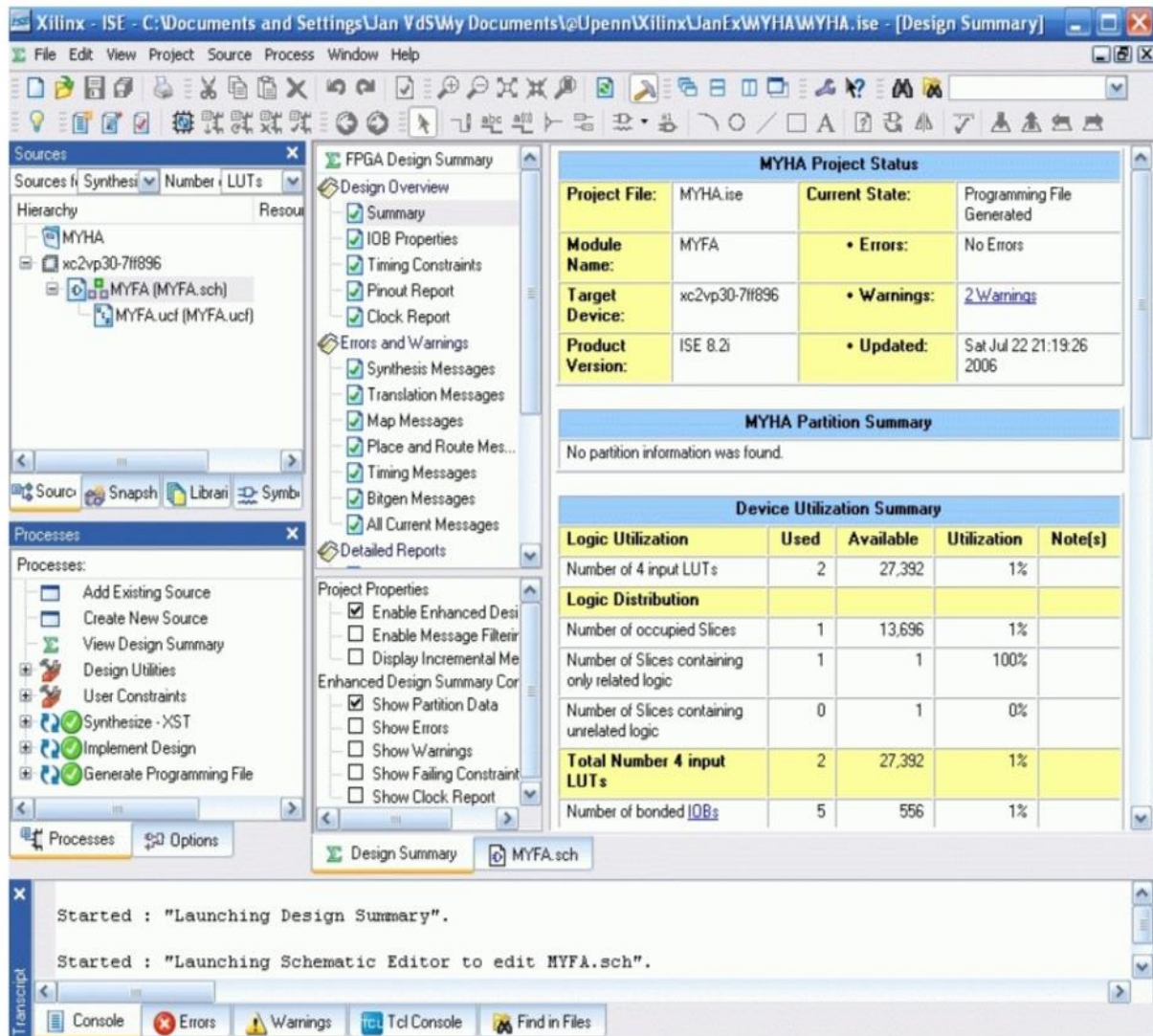


Figure: ISE Project Navigator Window (Screen clip from Xilinx (TM) ISE software)

A. Sources Window

This window contains the design source files for a project. These are the source files that you created or added to the project (see later on). A drop down list at the top of

sources window allows you to select source files that are associated with a particular design aspect such as Synthesis/Implementation or Simulation.

B. Processes Window

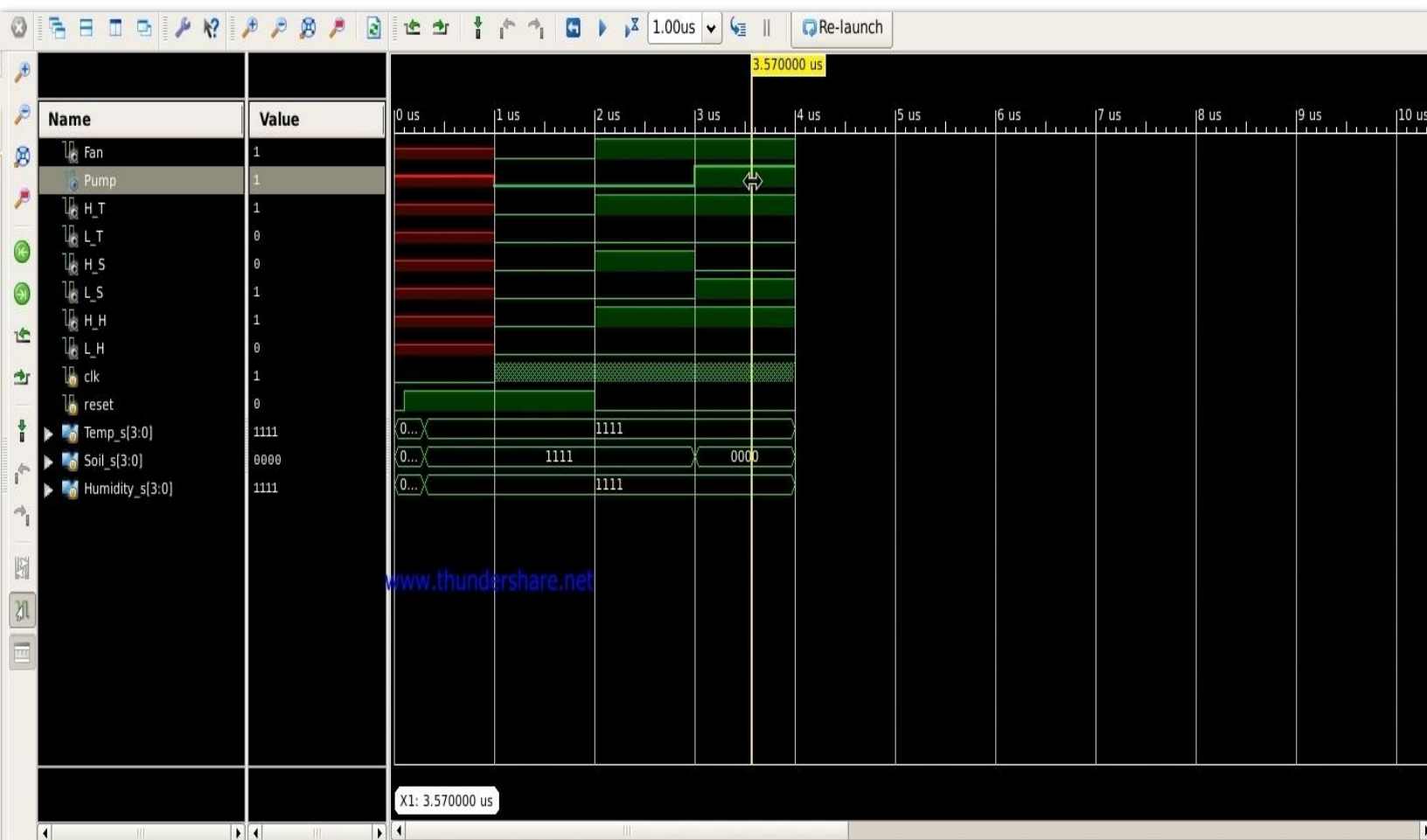
The processes windows list the available processes (corresponding to the process selected in the processes window). Typically you will select a particular process that you want to perform on the selected source file. This can include a simulation, implementation, etc. To run a process you can double click on the process. When a process has been successfully executed a red tick-off icon appears. When you run a high-level process, the Project Navigator will automatically run all the associated lower-level processes.

8.4 Device Types:

FPGA and CPLD There are two types of programmable logic devices. One is called a field programmable gate array (FPGA) and the other a complex logic device (CPLD). The CPLD XC9500 device has a PAL-like architecture and is non-volatile. It gives relatively good performance and is well suited for combinational logic circuits and control logic of medium complexity. The FPGA device (ex. Vertex series) has an array-like architecture and is volatile (SRAM based). It makes use of lookup tables (stored in the SRAM memory) to implement logic functions. It is good to realize complex logic functions that contain both combinational and sequential circuits. Its capacity is usually limited by the number of input/output pins and not by its complexity.

Chapter -9

OUTPUT SCREEN SHOTS:



Chapter 10

Conclusion

The objective of this project was to develop a system to provide autonomous control for temperature and humidity in closed environment, which is fulfilled. Also, to achieve competitiveness in the market, the production cost must be kept as low as possible. Low cost automation can be achieved by using VLSI systems so that all category farmers can afford it.

References:

1. Aniket.H.Hade, Dr. M.K. Sengupta-Automatic Control of Drip Irrigation System & Monitoring Of Soil by Wireless and SU SU YI MON,CHAW MYAT NEW- IMPLEMENTATION OF THE PRECISION AGRICULTURE USING LEACH PROTOCOL OF WIRELESS SENSOR NETWORK(Apr,2014).
2. Priyanka J. Ranade¹ , Prof. S. B. Takale - FPGA based Smart Water Distribution System using Wireless Sensor Network (june -2016)
3. FPGA implementation of humidity and temperature remote sensing system1- 4244-2396-5/08/\$20.00 ©2008 IEEE
4. O. Korner , H. Challaw, "Temperature integration and process-based humidity (March- 2016)