FPGA BASED CONTROLLER FOR HYDRO-ELECTRIC UNIT

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

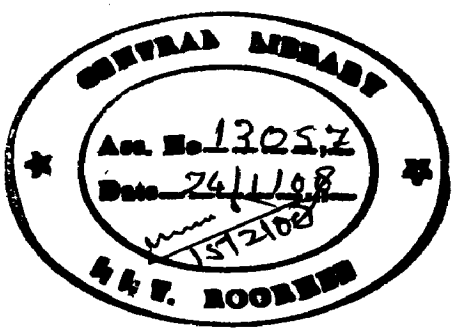
in

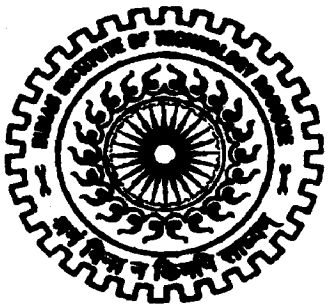
ELECTRICAL ENGINEERING

(With Specialization in Measurement and Instrumentation)

By

DATLA SRINIVASA RAJU





DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)
JUNE, 2007

I.D. No.-M.T.....403/2007-36/HKV

CANDIDATE'S DECLARATION

I here by declare that the work presented in this dissertation entitled, **"FPGA BASED CONTROLLER FOR HYDRO-ELECTRIC UNIT"** submitted in partial fulfillment of the requirement for the award of degree of **MASTER OF TECHNOLOGY** with specialization in **MEASUREMENT AND INSTRUMENTATION** in the department of **Electrical Engineering, Indian Institute of Technology Roorkee, Roorkee** is under the guidance of **Dr. H. K. Verma**, Professor, Department of Electrical Engineering, Indian Institute of Technology, Roorkee.

The matter embodied in this dissertation has not been submitted by me for the award of any other degree.

D S. Raju 29/06/07.

Dated: June 2007 **(DATLA SRINIVASA RAJU)**

Place: Roorkee

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

29/6/07

(Dr.H.K.Verma)
Professor,
Deptt. of Electrical Engg,
I.I.T Roorkee,
Roorkee-247667.

ACKNOWLEDGEMENT

I express my sincere gratitude towards my guide **Dr.H.K.Verma**, Professor, Department of Electrical Engineering, I.I.T. Roorkee, for guidance, advice, support and encouragement during the whole span of the work.

I convey my deep sense of gratitude to the Head of Electrical Engineering department, I.I.T. Roorkee for providing the facilities of the department for this work.

I appreciate and thank the entire laboratory and official staff of Department of Electrical Engineering, who directly or indirectly helped me during the work.

It is difficult for me to express my gratitude to my parents for their affection and encouragement; I continuously received from them, whenever I needed it in crucial and depressing moments.

Special and sincere thanks go to my friends whose support and encouragement has been a constant source strength to me.

(DATLA SRINIVASA RAJU)

ABSTRACT

Active and reactive power demands are never steady and they continually change with the demand. Water input to hydro-generators must, therefore, be continuously regulated to match the active power demand, failing which the machine speed will vary with consequent change in frequency, which may be highly undesirable.

A PID (proportional, integral and derivative) controller has been developed in VHDL to control the speed of the hydroelectric turbine. The Matlab/Simulink environment is used for modeling, simulation and evaluation of the performance of the hydro-electric unit. Modelsim is used to run the VHDL programming for the PID controller. "Link for ModelSim" is used to interface the two simulators to achieve the objective.

CONTENTS

	Page No
CANDIDATE DICLARATION	**i**
ACKNOWLEDGEMENT	**ii**
ABSTRACT	**iv**
CHAPTER-1 INTRODUCTION	**1**
1.1 Field Programmable Gate Array (FPGA)	1
1.2 VHSIC Hardware Description Language (VHDL)	2
1.3 Link for Modelsim	2
1.4 Hydro-Electric Unit	3
1.5 Statement of Problem	3
1.6 Dissertation Layout	4
CHAPTER-2 PID CONTROLLER	**5**
2.1 Introduction	5
2.2 Algorithm for PID Controller	8
2.3 Implementation in V.H.D.L	11
2.3.1 Adder	11
2.3.2 Subtractor	12
2.3.3 Register	12
2.3.4 Multiplier	12
2.3.5 PID Controller	14
2.3.6 Loop Holes in the Original Algorithm	14
CHAPTER-3 INTERFACING THE TWO SIMULATORS (MATLAB Simulink AND MODEL SIM)	**15**
3.1 Introduction	15
3.1.1 Typical Applications	15
3.1.2 Key Features	17
3.1.3 The Cosimulation Environment	17

3.1.4 Modes of Communication 20
3.2 Installation and Setup 20
3.2.1 Deciding on a Configuration 20
3.2.2 Modes of Communication 22
3.2.3 Identifying a Server in a Network Configuration 23
3.2.4 Installing Related Application Software 24
3.2.5 Setting up ModelSim for use with Link for ModelSim 24
3.3 Configuration Procedure for Interfacing the Simulink and ModelSim 26

CHAPTER-4 CONTROLLER FOR HYDRO-ELECTRIC UNIT 29
4.1 Introduction 29
4.2 Hydraulic Channel Model 31
4.3 Governor Model 32
4.3.1 PID governor 33
4.3.2 Procedure for linking the two simulators 33
4.4 Governor Control using Link for Modelsim 39
4.4.1 Setting up ModelSim for Use with Simulink 40
4.4.2 Loading Instances of the VHDL Entity for Cosimulation with Simulink 40
4.4.3 Running the Simulation 40
4.4.4 Shutting Down the Simulation 41

CHAPTER-5 RESULTS AND DISCUSSIONS 42
5.1 Starting from standstill 42
5.2 Loading the Machine 42
5.2.1 Rotor Speed 42
5.2.2 Electric Power Output 42
5.3 Waveforms 43

CHAPTER-6 CONCLUSION AND SCOPE FOR FUTURE WORK **48**

6.1 Conclusion 48

6.2 Scope for Future Work 49

REFERECES **50**

CHAPTER-1

INTRODUCTION

1.1 Field Programmable Gate Array (FPGA):

A **field-programmable gate array** is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to represent the functionality of basic logic gates. These logic gates may be a simple such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components also include memory elements, which may be simple flip-flops or more complete blocks of memories.

The application range of FPGA based designs Increases every day. This is mainly due to the flexibility and capability to perform parallel tasks. The industry is adopting massively the core-based design methodology for system integration using FPGAs, which leads to the appearance of the System-on-Programmable-Chip (SoPC) platforms .These capabilities have been increased with the addition of microprocessors inside the FPGAs, either embedded in specific hardware or synthesized and included in general logic.

FPGA is an integrated circuit that can be configured by the user in order to implement digital logic functions of varying complexities. FPGA can be very effectively used for control purposes in processes demanding very high loop cycle time. The implementation of a digital controller in a FPGA can be parallel, resulting in very high speeds of operation. This fact enables FPGAs to score over general-purpose computing chips like DSP chips, which have a limited number of Multiplier ACcumulator (MAC) units that can be used for the controller design [1].

1.2 VHSIC Hardware Description Language (VHDL):

VHSIC Hardware Description Language is an industry standard language used to describe hardware from the abstract to the concrete level. Moreover, it is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits [2].

VHDL Descriptions consist of primary design units and secondary design units. The primary design units are the entity and the package. The secondary design units are the architecture and the package Body. Secondary design units are always related to a primary design unit. Libraries are collections of primary and secondary design units. A typical design usually contains one or more libraries of design units [3].

1.3 Link for Modelsim [5]:

Link for ModelSim is a cosimulation interface between ModelSim and Matlab SimLink. It integrates MathWorks tools into the Electronic Design Automation (EDA) workflow for field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) development. The interface provides a fast bidirectional link between the Mentor Graphics hardware description language (HDL) simulator, ModelSim SE/PE, and the MathWorks products MATLAB and Simulink for direct hardware design verification and cosimulation. The integration of these tools allows applying each product to the tasks it does best.

The "Link for ModelSim" is a collection of Simulink blocksets that permit interaction between hardware description language, which is implemented in ModelSim and modeled systems in Simulink. The toolboxes include a series of blocks, such as cosimulation block that can be used to build a system. With the help of these blocksets, test bench can be built to estimate the performance of our VHDL code. Moreover, the response in both

the Simulink by using scope block and in ModelSim by using a wave generator can be traced [14].

1.4 Hydro-Electric Unit:

Hydroelectric unit is a generating unit, which generates electric power from Hydropower. In the generating process, active and reactive power demands are never steady and they continually change with the rising or falling trend. water input to hydro-generators must, therefore, be continuously regulated to match the active power demand, failing which the machine speed will vary with consequent change in frequency which may be highly undesirable (maximum permissible change in power frequency is +- 0.5 Hz)[6].

In modern large interconnected systems, manual regulation is not feasible and therefore automatic generation and voltage regulation equipment is installed on each generator. The controllers are set for a particular operating condition and they take care of small changes in load demand without frequency and voltage exceeding the prescribed limits. With the passage of time, as the change in load demand becomes large, the controllers must be reset either manually or automatically [7].

1.5 Statement of Problem:

The objective of this work is to develop a suitable FPGA based controller for hydroelectric unit. The basic PID controller is used to control the valve of the governor system of the hydroelectric unit to maintain the speed as well the active power is constant.

The PID controller is developed in a VHDL, which is a hardware description language and is compiled in ModelSim simulator. In addition, the hydroelectric unit is simulated in MATLAB Simulink environment. The PID controller in ModelSim is incorporated in governor control in Simulink by using powerful cosimulation environment "Link for ModelSim".

1.6 Dissertation Layout:

This dissertation is organized into six chapters. They are briefly introduced here.

In Chapter-1, a brief description of FPGA, VHDL, Link for ModelSim and Hydroelectric unit including the objective of this dissertation work.
Chapter-2 presents the algorithm and implementation in VHDL of the controller, which is developed for hydroelectric unit.
Chapter-3 describes the procedure for the interfacing the two simulators ModelSim and Matlab Simulink.
Chapter-4 presents the development and working of the controller for hydroelectric unit.
Chapter-5 gives the results and discussion of the dissertation work
Finally, Chapter-6 presents the conclusion and scope for the future work

CHAPTER-2

PID CONTROLLER

2.1 Introduction:

PID (Proportional-Integral-Derivative) is the most common control methodology in industrial control. It is a continuous feedback loop that keeps the process flowing by taking corrective action whenever there is any deviation from the desired value ("set point") of the process variable (rate of flow, temperature, voltage, etc.). The deviation from the desired value is called the "error". An "error" occurs when an operator manually changes the set point or when an event (valve opened, closed, etc.) or a disturbance changes the load, thus causing a change in the process variable [7].

The PID controller receives signals from sensors and computes corrective action to the actuators from a computation based on the error (proportional), the sum of all previous errors (integral) and the rate of change of the error (derivative).

The controller takes a measured value from a process or other apparatus and compares it with a reference setpoint value. The difference (or "error" signal) is then used to adjust some input to the process in order to bring the process's measured value to its desired setpoint. Unlike simpler controllers, the PID can adjust process outputs based on the history and rate of change of the error signal, which gives more accurate and stable control. In contrast to more complex algorithms such as optimal control theory, PID controllers can often be adjusted without advanced mathematics.

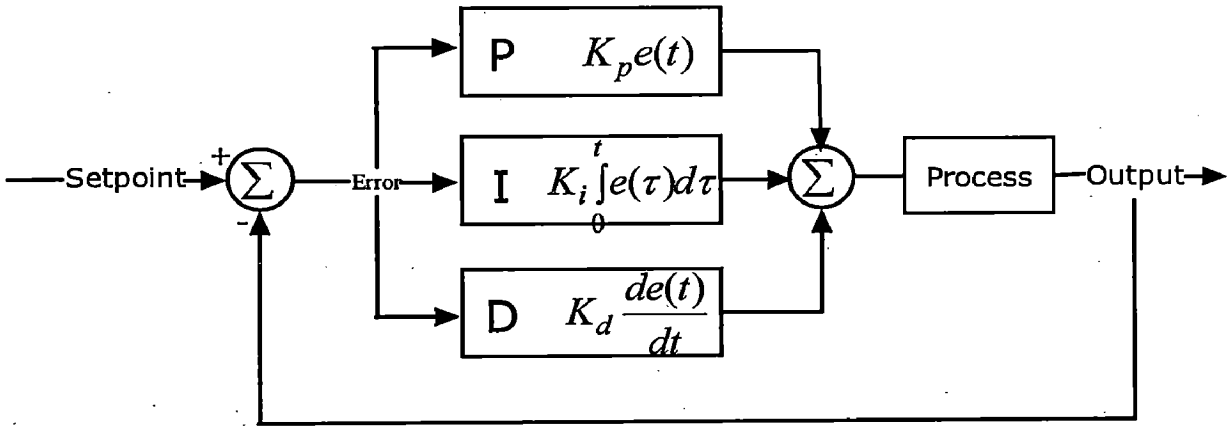


Fig 2.1 Basic Block Diagram of a PID controller [7]

The **PID** control scheme is named after its three correcting terms, whose sum constitutes the output.

Proportional - To handle the immediate error, the error is multiplied by a constant K_p. Note that when the error is zero, a proportional controller's output is zero. However, the proportional controller will not reach the setpoint if a non-zero output is required to maintain the setpoint. This is called a "steady state error". To eliminate this error an Integral component must be added to the controller.

Integral - To find out from the past, the error is integrated and multiplied by a constant K_i. The integral term allows a controller to eliminate a steady state error if the process requires a non-zero input to produce the desired setpoint. An integral controller will react to the error by accumulating a value that is added to the output value. While this will force the controller to approach the setpoint faster than a proportional controller alone and eliminate steady state error, it also guarantees that the process will overshoot the setpoint since the integral value will continue to be added to the output value.

Derivative - To predict the future, the first derivative of the error is multiplied by a constant K_d. This can be used to reduce the magnitude of the overshoot produced by the integral component, but the controller will be a bit slower to reach the setpoint initially.

The output of the controller (i.e. the input to the process) is given by

Output (t) $=P_{contrib}+I_{contrib}+D_{contrib}$ 2.1

Where $P_{contrib}$, $I_{contrib}$, $D_{contrib}$ are:

$$P_{contrib}=K_p e(t)$$

$$I_{contrib}=\frac{1}{T_i}\int_0^t e(\tau)d\tau$$

$$D_{contrib}=T_d\frac{de}{dt}$$

Where e(t) = Setpoint - Measurement(t), is the error signal, and K_p, T_i, T_d are constants that are used to tune the PID control loop:

K_p: Proportional Gain - Larger Kp typically means faster response since the larger the error, the larger the feedback to compensate.

T_i: Integral Time - Smaller Ti implies steady state errors are eliminated quicker. The tradeoff is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before reaching the steady state.

T_d: Derivative Time - Larger T_d decreases overshoot, but slows down transient response.

Normally the controller is implemented with the K_p gain applied to the $I_{contrib}$, and $D_{contrib}$ terms as well in the following form, also called the standard form.

$$u(t) = K_p[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt}] \quad \text{.....2.2}$$

In the ideal parallel form, the standard parameters K, Ti and Td are replaced with (K_p, K_i and K_d).

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \quad \text{.....2.3}$$

In this notation the gain parameters are related to the parameters of the standard form through $K_p = K$, and $K_d = K * Td$.

2.2 Algorithm for PID Controller:

In this project, the PID algorithm is applied for closed-loop control. This is the most commonly used control law and has been demonstrated to be effective for FPGA implementation. The PID controller is described in a differential equation from equation 2.3 as [8]:

$$u(t) = K_p[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d \frac{de(t)}{dt}] \quad \text{.....2.3}$$

Where K_p is the proportional gain, T_i is the integral time constant and T_d is the derivative time constant.

For a small sample interval T, this equation can be changed into a difference equation by discretization. A difference equation can be implemented by a digital system, either in hardware or software easily. The

derivative term is simply replaced by a first-order difference expression and the integral by a sum, thus the difference equation 2.3 can be written as:

$$u(n) = K_p[e(n) + \frac{1}{T_i}\sum_{j=0}^{n} e(j) + \frac{T_d}{T}(e(n) - e(n-1))] \quad \text{.....2.4}$$

Equation (2.4) can be rewritten as:

$$u(n) = K_p e(n) + K_i \sum_{j=0}^{n} e(j) + K_d(e(n) - e(n-1)) \quad \text{.....2.5}$$

Where $K_i = K_pT/T_i$ is the integral coefficient,

$K_d = K_pT_d/T$ is the derivative coefficient.

To compute the sum, all past errors, e (0)...e (n), have to be stored. This algorithm is called the "position algorithm". An alternative recursive algorithm can be derived from the calculation of the control output, u (n), based on u(n−1) and the correction term Δu(n). To derive the recursive algorithm, first calculate u (n − 1) based on Eq. (2.5):

$$u(n-1) = K_p e(n-1) + K_i \sum_{j=0}^{n} e(j) + K_d(e(n-1) - e(n-2)) \quad \text{.....2.6}$$

Then calculate the correction term as:

$$\Delta u(n) = u(n) - u(n-1)$$

$$= K_0 e(n) + K_1 e(n-1) + K_2 e(n-2) \quad \text{.....2.7}$$

Where $K_0 = K_p + K_i + K_d, K_1 = -K_p - 2K_d, K_2 = K_d$.

Equation (2.7) is called the "incremental algorithm". The control output is calculated as:

$$u(n) = u(n-1) + K_0 e(n) + K_1 e(n-1) + K_2 e(n-2) \quad \text{.....2.8}$$

In the software implementation, the incremental algorithm (Eq.2.8) can avoid accumulation of all past errors e (n) and can realize smooth switching from manual to automatic operation, compared with the position algorithm (Eq.2.5).

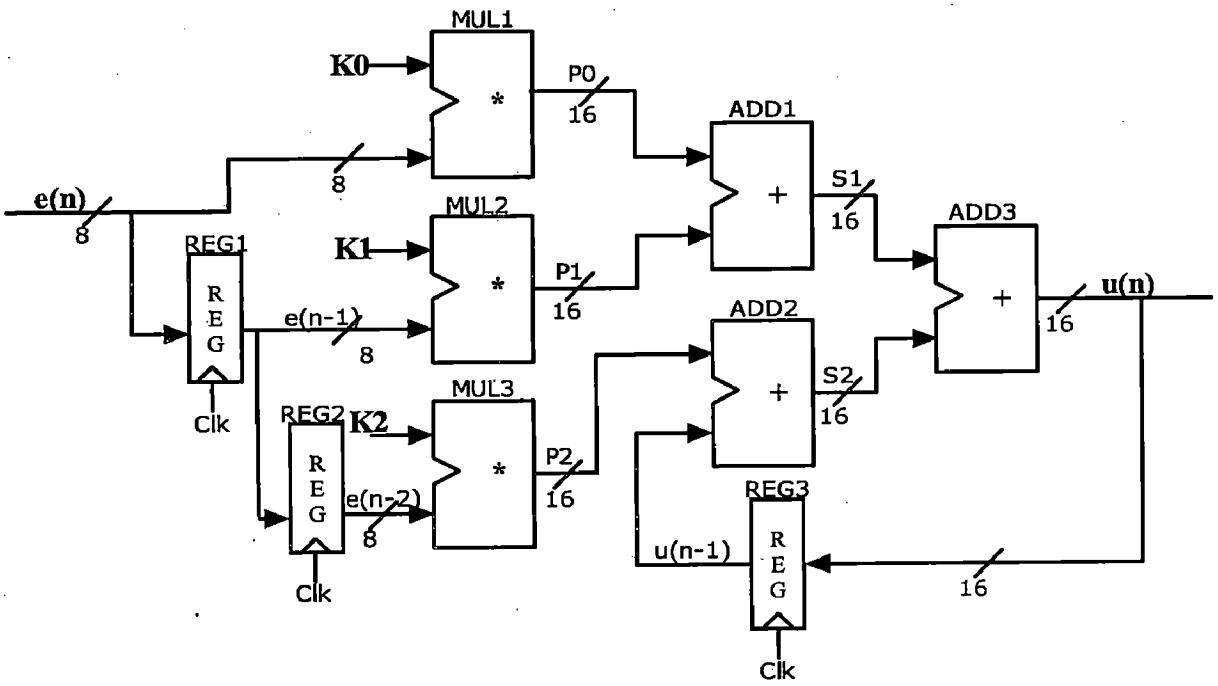


Fig 2.2 Algorithm for PID controller [8]

Where e(n) =error signal

$$p_0 = K_0 \times e(n)$$

$$p_1 = K_1 \times e(n-1)$$

$$p_2 = K_2 \times e(n-2)$$

$$s_1 = P_0 + P_1$$

$$s_2 = P_2 + u(n-1)$$

$$u(n) = s_1 + s_2$$

Figure 2.2 shows parallel design of the PID control algorithm. The design requires 4 adders and 3 multipliers, corresponding to the basic operations. In the figure bold signals are I/O ports, while others are internal signals. The clock signal clk is used to control sampling frequency. At the rising edge of control, signal e (n) of the last cycle is latched at register REG1, thus becomes e (n−1) of this cycle. In the same manner, e (n−2) and u (n − 1) are recorded at REG3 and REG4 by latching e (n − 1) and u (n) respectively.

2.3 Implementation in VHDL:

VHDL, or **V**HSIC **H**ardware **D**escription **L**anguage, is commonly used as a design-entry language for field-programmable gate arrays and application-specific integrated circuits in electronic design automation of digital circuits.

The figure 2.2 which is implemented from the algorithm consists of three 8-bit multipliers, three 16-bit adders, three registers in which two are 8-bit, one is 16-bit and one subtractor of 16-bit. So, implementation of adder, multiplier, register and subtractor is described in brief as follows.

2.3.1 Adder:

Implementation of n-bit adder in Hardware Description Language is different implementing in digital design. In digital design n-bit adder can be constructed from a full adder by input carry, augend and addend bits. In this way n number of full adders has to be added for n-bit adder. But in VHDL, n-bit adder can be designed in a simple manner with a simple logic as follows.

```
isum(i) := a(i) xor b(i) xor carry;
carry  := (a(i) and b(i)) or (a(i) and carry) or (b(i) and carry);
```

These two relations used in the loop of n for n-bit adder for getting the n-bit sum and carry. For further clarification see Appendix-source code for PID controller.

2.3.2 Subtractor:

The subtractor can be designed in V.H.D.L by modifying the adder. In digital system subtraction can be replaced by the adding of minuend to the 2's complement of subtrahend. By changing the augend of the adder as the NOT of subtrahend and input carry as '1' adder itself acts as subtractor.

2.3.3 Register:

The register can be designed with a simple logic as if clock input is '1' then output is equal to input.

2.3.4 Multiplier:

With simple logic in a loop of n, which is the order, multiplier can be designed. From the figure 2.3 and figure 2.4 gives the clear idea of the multiplier and logic design.

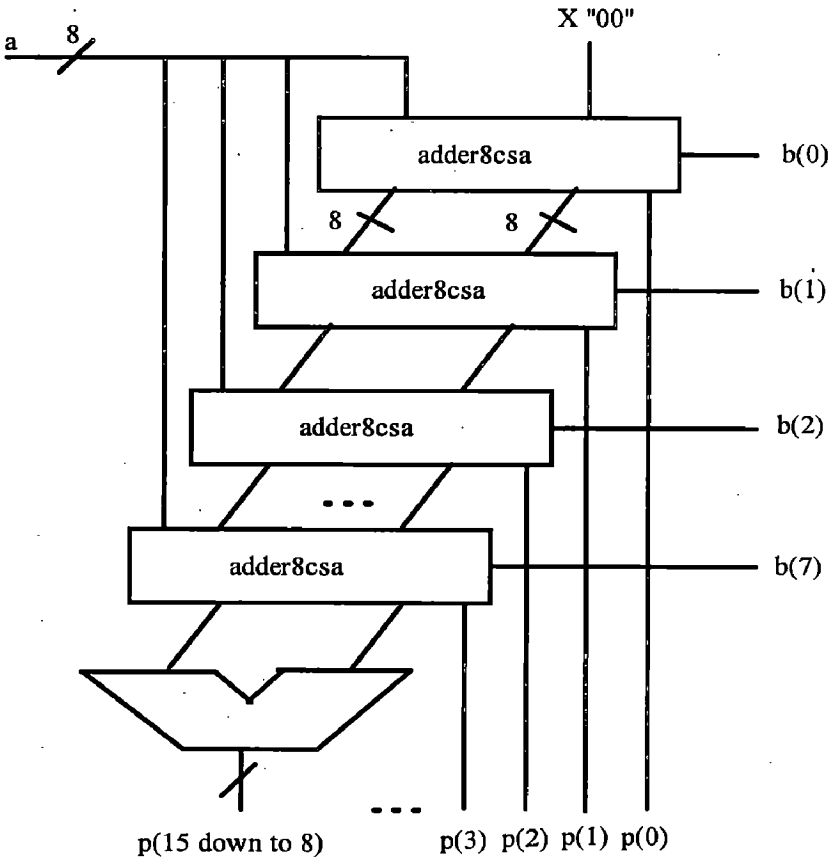


Fig 2.3 Algorithm for multiplier [4]

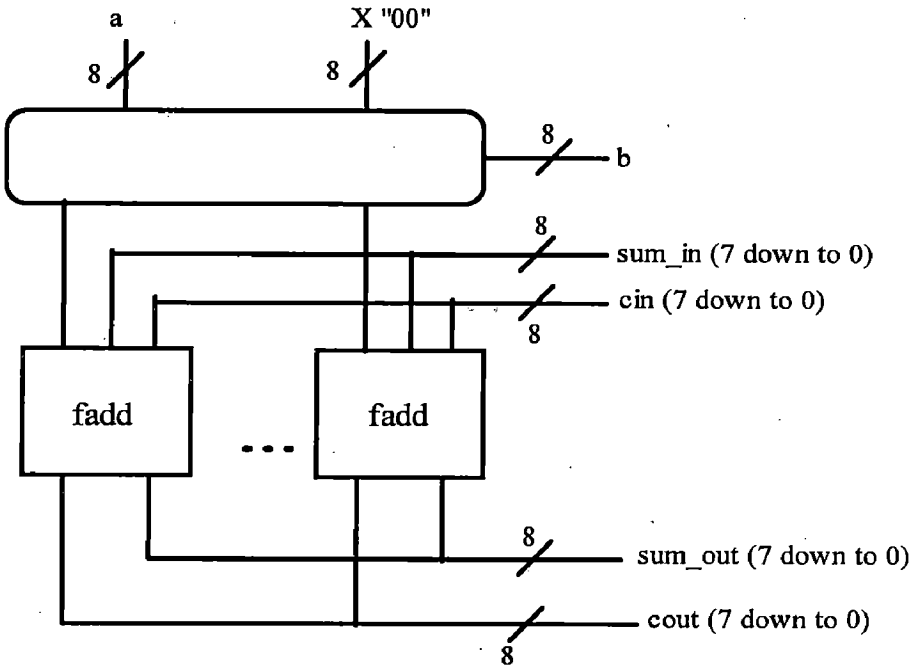


Fig 2.4 schematic for fadd [4]

2.3.5 PID Controller:

By connecting the each component as in the figure 2.2 with the help of signals pid controller can get the final design.

2.3.6 Loop Holes in the Original Algorithm:

In the actual process the multiplier3 output is added to u(n-1). For the first clock pulse input to the register is 'U', which is undefined then the output also becomes undefined 'U'. If both the values are added the ultimate output is undefined 'U'.

To avoid this problem, different ways have been tried such as initializing the output as '0' and usage of multiplexer. In VHDL, it cannot be possible to initialize the output port. And then attempt has been made with multiplexer, where inputs are '0' and u(n). If the output port is 'U' then '0' input has been selected otherwise u(n). But this also gives the same undefined output as 'U'.

After so many attempts, finally the architecture of the adder has been changed such that if any of the inputs is undefined 'U' then the output is forced to '0'. Due to change in the architecture, output of the adder for the first clock pulse becomes '0'. Then onwards, as the u(n-1) is defined correctly output becomes actual value.

CHAPTER-3

INTERFACING THE SIMULATORS (MODELSIM AND MATLAB SIMULINK)

3.1 Introduction:

Link for ModelSim is a cosimulation interface between ModelSim and Matlab SimLink. It integrates MathWorks tools into the Electronic Design Automation (EDA) workflow for field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) development. The interface provides a fast bidirectional link between the Mentor Graphics hardware description language (HDL) simulator, ModelSim SE/PE, and the MathWorks products MATLAB and Simulink for direct hardware design verification and cosimulation. The integration of these tools allows applying each product to the tasks, it does best [5]:

a) ModelSim — hardware modeling in HDL and simulation
b) MATLAB — numerical computing, algorithm development, and visualization
c) Simulink — simulation of system-level designs and complex models

The Link for ModelSim interface consists of MATLAB functions and ModelSim commands for establishing the communication links between ModelSim and the MathWorks products. In addition, a library of Simulink blocks is available for including ModelSim HDL designs in Simulink models for cosimulation.

3.1.1 Typical Applications:

Link for ModelSim streamlines FPGA and ASIC development by integrating tools available for

1. Developing specifications for hardware design reference models

2. Implementing a hardware design in HDL based on a reference model
3. Verifying the design against the reference design

The figure 3.1 shows how ModelSim and MathWorks products fit into this hardware design scenario.

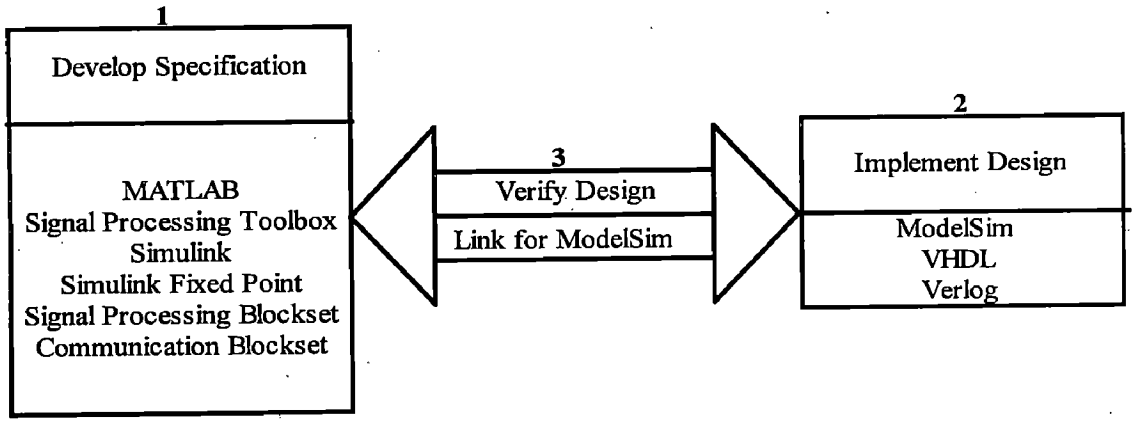


Fig 3.1 Basic Block Diagram for Link for ModelSim [5]

As the figure shows, Link for ModelSim connects tools that traditionally have been used discretely to accomplish specific steps in the design process. By connecting the tools, Link for ModelSim simplifies verification by allowing to cosimulate the implementation and original specification directly. The end result is significant time savings and the elimination of errors inherent to manual comparison and inspection.

In addition to the preceding design scenario, Link for ModelSim allows to use

1. MATLAB or Simulink to create test signals and software test benches for HDL code
2. MATLAB or Simulink to provide a behavioral model for an HDL simulation
3. MATLAB analysis and visualization capabilities for real-time insight into an HDL implementation
4. Simulink to translate legacy HDL descriptions into system-level views

3.1.2 Key Features:

Key features of Link for ModelSim include

1. Ability to link ModelSim to MATLAB and Simulink for bidirectional cosimulation, verification, and visualization
2. Support for PE and SE versions of ModelSim
3. Support for Window and Unix platforms (see the MathWorks Link for ModelSim requirements page for specific platforms supported)
4. Support for shared memory and TCP/IP socket modes of communication between MATLAB and Simulink and ModelSim
5. A Simulink block for cosimulating HDL models (VHDL or Verilog) in Simulink
6. A Simulink block for exporting test vectors and results as value change dump (VCD) files
7. Support for multiple simultaneous ModelSim instances, and multiple HDL entities from within one Simulink model or MATLAB function
8. Interactive or batch mode cosimulation, debugging, testing, and verification of HDL code (VHDL or Verilog) from within MATLAB
9. MATLAB test bench functions that support verification of the performance of a VHDL or Verilog model, or of components within the model
10. MATLAB component functions that simulate the behavior of entities in a VHDL or Verilog model

3.1.3 The Cosimulation Environment [5]:

Link for ModelSim is a client/server test bench and cosimulation application. The role that ModelSim plays in a Link for ModelSim simulation environment depends on whether ModelSim links to MATLAB or Simulink.

i. MATLAB and ModelSim Links

When linked with MATLAB, a ModelSim function as the client, as the figure 3.2 shows.

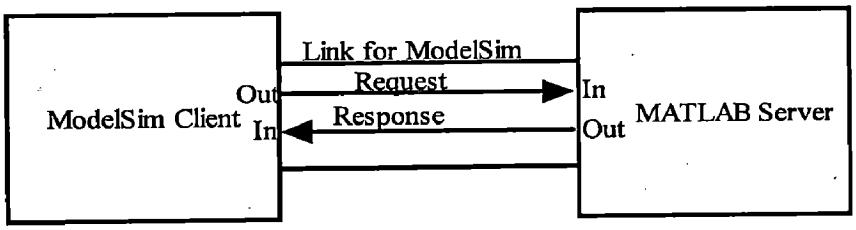


Fig 3.2 MATLAB and ModelSim Links [5]

In this scenario, a MATLAB server function waits for service requests that it receives from a ModelSim simulator session. After receiving a request, the server establishes a communication link, and invokes a specified MATLAB function wrapper that computes data for, verifies, or visualizes the HDL model (coded in VHDL or Verilog) that is under simulation in ModelSim.

The MATLAB server can service multiple simultaneous ModelSim sessions and HDL entities. However, The figure 3.3 shows a multiple-client scenario connecting to the server at TCP/IP socket port 4449.

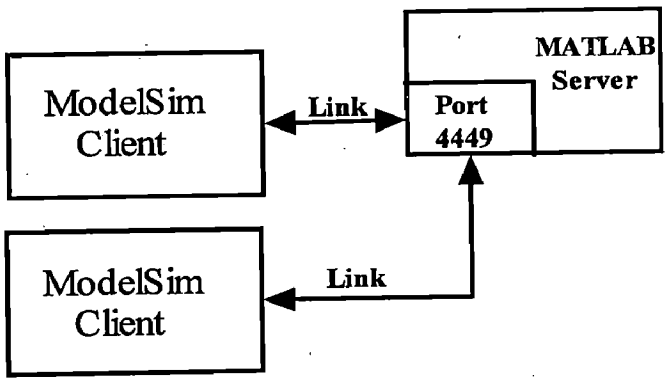


Fig 3.3 MATLAB and ModelSim Links in a Network Environment [5]

ii. Simulink and ModelSim Links

When linked with Simulink, ModelSim functions as the server, as shown in the figure 3.4.

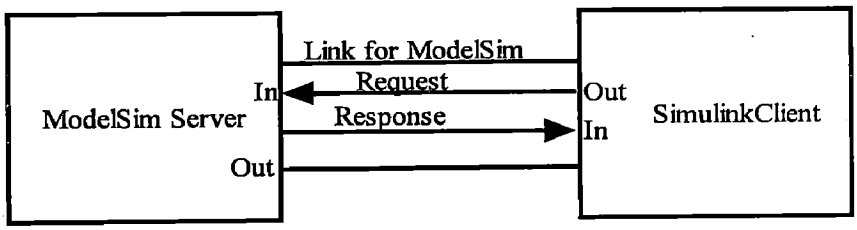


Fig 3.4 Simulink and ModelSim Links [5]

In this case, ModelSim responds to simulation requests it receives from cosimulation blocks in a Simulink model. Once the cosimulation session is initiated, it can be used Simulink and ModelSim to monitor simulation progress and results. For example, to monitor simulating timing diagrams, the signal has to be added to a ModelSim Wave window.

As the figure 3.5 shows, multiple cosimulation blocks in a Simulink model can request the service of multiple instances of ModelSim, using unique TCP/IP socket ports.

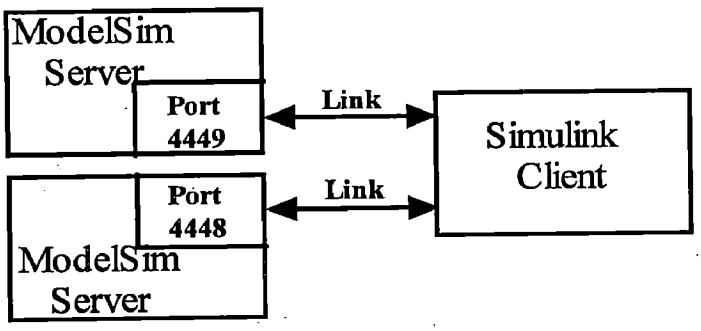


Fig 3.5 Simulink and ModelSim Links in a Network Environment [5]

3.1.4 Modes of Communication:

The mode of communication that Link for ModelSim uses for a link between ModelSim and MATLAB or Simulink somewhat depends on whether the simulation application runs in a local, single-system configuration or in a network configuration. If ModelSim and the MathWorks products can run locally on the same system and the application requires only one communication channel, it has the option of choosing between shared memory and TCP/IP socket communication. Shared memory communication provides optimal performance and is the default mode of communication.

3.2 Installation and Setup:

This section explains how to define the Link for ModelSim application environment.

Environment Requirements:

1. Configurations
2. Mode of Communication
3. Network Configurations
4. Related Software
5. ModelSim Setup

3.2.1 Deciding on a Configuration:

For various configurations of an application:

i. Shared memory communication is an option for configurations that require only one communication link on a single computing system.

ii. TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.

iii. In any configuration, an instance of MATLAB can run only one instance of the Link for ModelSim MATLAB server (hdldaemon) at a time.

iv. In a TCP/IP configuration, the MATLAB server can handle multiple client connections to one or more ModelSim sessions.

v. HDL Cosimulation blocks in a Simulink model can connect to the same or different ModelSim sessions.

vi. When using both MATLAB and Simulink, different TCP/IP ports must be used for links between these products and ModelSim.

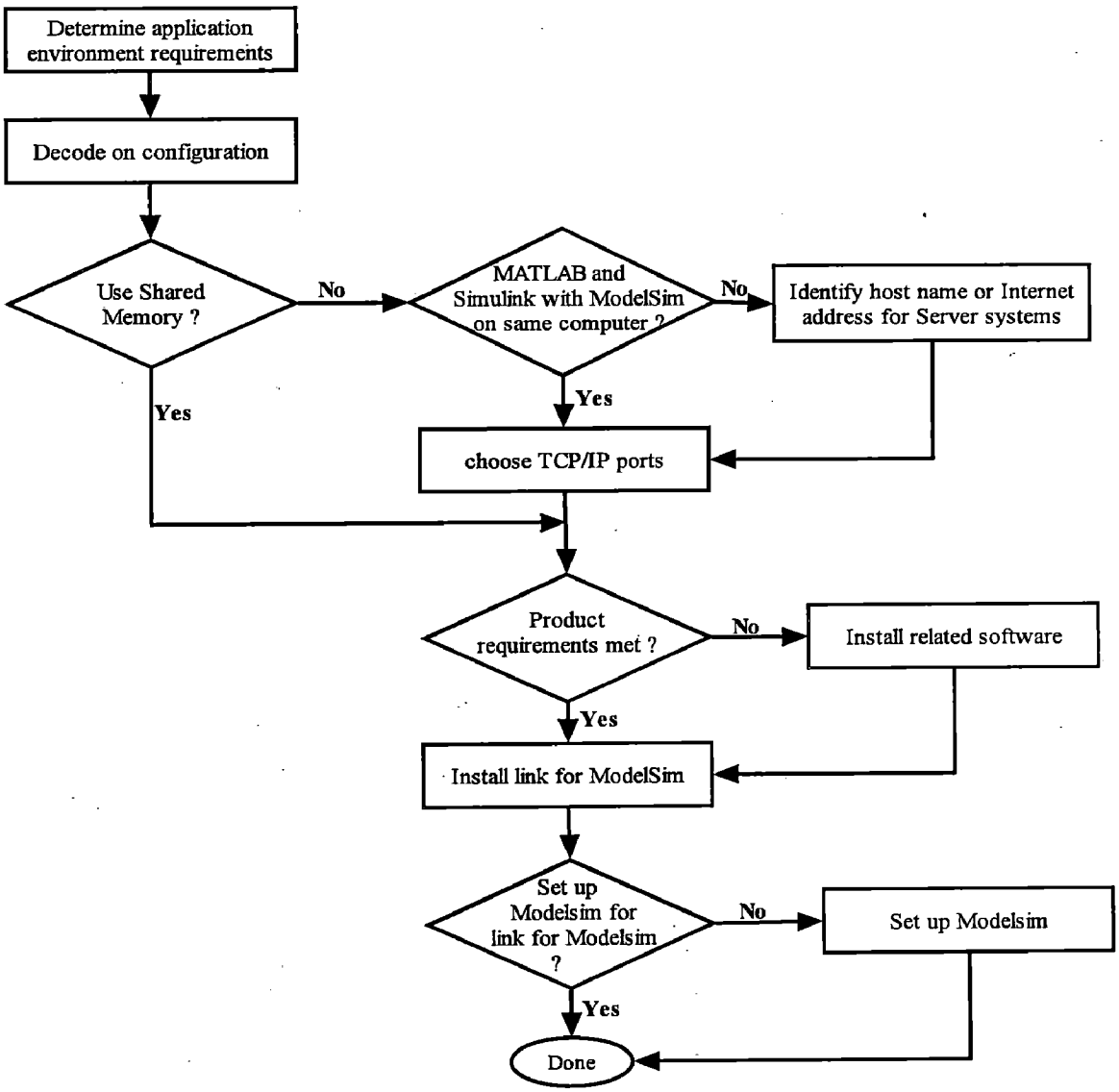


Fig 3.6 flow chart for installation and setup [5]

The scenarios apply whether ModelSim is running on the same or different computing system as MATLAB or Simulink. In a network configuration, an Internet address in addition to a TCP/IP socket port to identify the servers in an application environment.

3.2.2 Modes of communication [5]:

The mode of communication that the Link for ModelSim uses for a link between ModelSim and MATLAB or Simulink somewhat depends on whether

the simulation application runs in a local, single-system configuration or in a network configuration. If ModelSim and the MathWorks products can run locally on the same system and the application requires only one communication channel, it has the option of choosing between shared memory and TCP/IP socket communication. Shared memory communication provides optimal performance and is the default mode of communication.

To use the TCP/IP socket communication, it has to choose a TCP/IP socket port number that is available in computing environment for use by the Link for ModelSim client and server components. The two components use the port number to establish a TCP/IP connection. Port numbers are particularly important for applications that implement multiple clients and servers and use TCP/IP socket communication on a single node. The port numbers uniquely identify each client and server and enable connections only between components sharing the same port number. For remote network configurations, the Internet address helps distinguish multiple connections.

In MATLAB, checking the server status at this point indicates that the server is running with no connections:

```
x=hdldaemon ('status')
HDLDaemon server is running with 0 connections
x= 4449
```

3.2.3 Identifying a Server in a Network Configuration:

If there is a need to set up a Link for ModelSim application such that ModelSim and the MathWorks products reside on different systems, it has to set up the systems to use

i. TCP/IP networking protocol
ii. Link for ModelSim TCP/IP socket mode of communication

As part of the application setup, it has to identify

i. The Internet address or host name of the computer running the server component of our application
ii. The TCP/IP socket port number or service name (alias) to be used for Link for ModelSim connections

3.2.4 Installing Related Application Software:

Based on the configuration decisions and the software required for our Link for ModelSim application, identify software the need to install and where is the requirement to install it. For details on how to install ModelSim, see the installation instructions for that product. For information on installing MathWorks products, see the MATLAB installation instructions [5].

Based on the configuration decisions, identify systems on which it has to install Link for ModelSim. Install Link for ModelSim on each system running MATLAB that requires a communication channel for ModelSim and MATLAB or Simulink cosimulation.

For details on how to install Link for ModelSim, see the MATLAB installation instructions.

3.2.5 Setting Up ModelSim for Use with Link for ModelSim:

There is a choice to have ModelSim run on the same machine as MATLAB or on a separate machine:

i. If the same machine has been chosen, then no additional installation instructions are necessary. However, when ModelSim is run on the same machine as MATLAB, it has the option to configure ModelSim to be able to work with Link for ModelSim when invoked from outside of MATLAB. To enable this feature, follow the instructions in Setting Up ModelSim on the Same Machine as MATLAB for configuring ModelSim for use with MATLAB.

ii. If different machine has been chosen, follow the instructions in Setting Up ModelSim on a Separate Machine from MATLAB.

After all the required software is installed, it has to set up ModelSim so that it is always ready for use with MATLAB and Simulink. It can be complete this setup immediately after installing the software (or later), either interactively or programmatically from scripts.

To configure ModelSim for use with Link for ModelSim when ModelSim is invoked outside of MATLAB, use the MATLAB function configuremodelsim:

Configuremodelsim

Identify the ModelSim installation to be configured for MATLAB and Simulink

Do you want configuremodelsim to locate installed ModelSim executables [y]/n? n

Please enter the path to your ModelSim executable file (modelsim.exe or vsim.exe): C:\Modeltech_6.0b\win32

Modelsim successfully configured to be used with MATLAB and Simulink

The configuremodelsim function registers new MATLAB- and Simulink-related Tcl commands for the ModelSim simulator by creating the file ...\tcl\ModelSimTclFunctionsForMATLAB.tcl within in the ModelSim installation directory. This command does *not* select the configured ModelSim executable as the default simulator to be used by the vsim command (that selection is done instead by the vsimdir property of the vsim command).

If ModelSim is running on a machine that does not have MATLAB, it has to provide ModelSim with the libraries and configuration information it

needs to communicate with MATLAB. Every time ModelSim is started, and want it to communicate with MATLAB, it has to run vsim with the startup file that created as part of this setup.

3.3 Configuration Procedure for Interfacing the Simulink and ModelSim [5]:

The basic steps for setting up a Link for ModelSim session that uses Simulink and the HDL Cosimulation block to verify an HDL model. The HDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from an HDL model under simulation in ModelSim. The HDL Cosimulation block supports simulation of either VHDL or Verilog models. These are the following steps:

i. Developing the VHDL Code:

A typical Simulink and ModelSim scenario is to create a model for a specific hardware component in ModelSim that need to integrate into a larger Simulink model. The first step is to design and develop a VHDL model in ModelSim.

ii. Compiling the VHDL File:

Set up a design library and compile pid.vhd by giving the command in ModelSim

```
ModelSim> vcom pid.vhd
```

iii. Creating the Simulink Model:

1. Configure the Constant block, which is the model's input source:

I. change the parameter values in the **Main** pane:

II. Click the **Signal data types** tab. The dialog box now displays the **Output data type mode** menu. Select appropriate **Output data type mode** menu.

2. Configure the HDL Cosimulation block, which represents the pid model written in VHDL.

i. In the **Ports** pane, give signal data path in the full HDL name edit field.
ii. In the **Connection** pane, select the shared memory.
iii. Configure the **Clocks** pane by adding clock properties.
iv. Enter some simple Tcl commands to be executed before and after simulation in the **Tcl** tab:
v. View the **Timescales** pane to make sure it is set to its default parameters.
vi. The final step is to connect the blocks, configure model-wide parameters, and save the model it has been shown in Fig 3.7

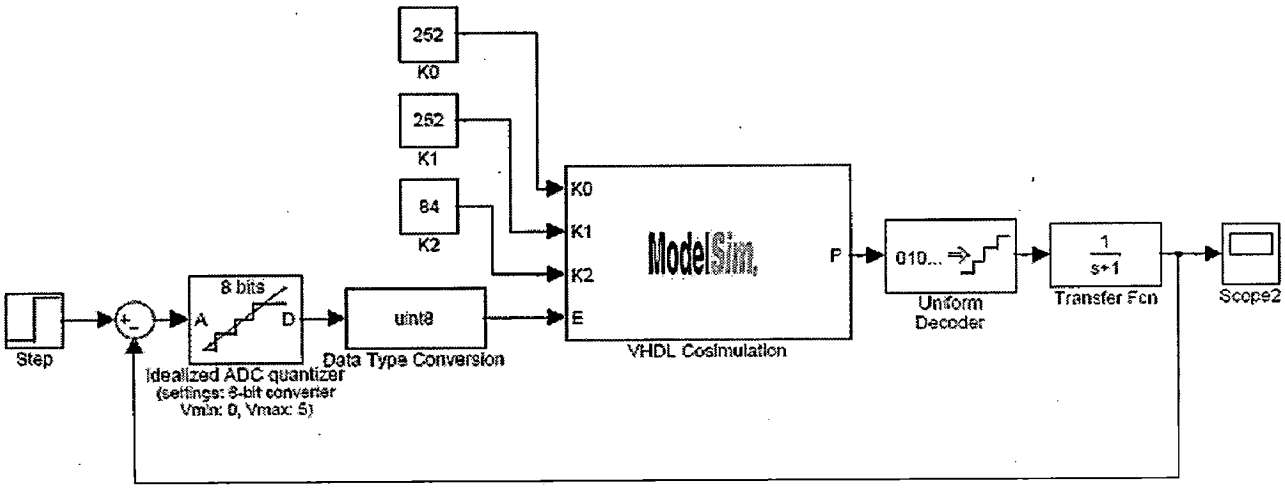


Fig 3.7 Schematic of PID controller in Simulink

3. Configure the Simulink solver options for a fixed-step, discrete simulation; this is required for correct cosimulation operation.

iv. Setting Up ModelSim for Use with Simulink:

Now it has a VHDL representation of a PID and a Simulink model that applies the PID. To start ModelSim such that it is ready for use with Simulink, enter the following command line in the MATLAB Command Window:

>>vsim

v. Loading Instances of the VHDL Entity for Cosimulation with Simulink:

The vsimulink command is a Link for ModelSim variant of the ModelSim vsim command. It is made available as part of the ModelSim configuration. To load an instant enter the following vsimulink command:

ModelSim> vsimulink work.pid

vi. Running the Simulation:

Running and monitoring a cosimulation session is achieved by adding a wave window by entering the following ModelSim command:

VSIM *n*> add wave /pid/*

vii. Shutting down the Simulation:

Shut down a simulation in an orderly way by selecting **Simulate** > **End Simulation**.

CHAPTER-4

CONTROLLER FOR HYDRO-ELECTRIC UNIT

4.1 Introduction:

Active and reactive power demands are never steady and they continually change with the rising or falling trend. water input to hydro-generators must, therefore, be continuously regulated to match the active power demand, failing which the machine speed will vary with consequent change In frequency which may be highly undesirable (maximum permissible change in power frequency is +- 0.5 Hz). In addition, the excitation of generators must be continuously regulated to match the reactive power demand with reactive generation; otherwise, the voltages at various system buses may go beyond the prescribed limits [9].

In modern large interconnected systems, manual regulation is not feasible and therefore automatic generation and voltage regulation equipment is installed on each generator. Fig 4.1 gives the schematic diagram of load frequency and excitation voltage regulators of a turbo-generator. The controllers are set for a particular operating condition and they take care of small changes in load demand without frequency and voltage exceeding the prescribed limits. With the passage of time, as the change in load demand becomes large, the controllers must be reset either manually or automatically [6].

For small changes, active power is dependent on internal machine angle δ and is independent of bus voltage. While bus voltage is dependent on machine excitation (therefore on reactive generation Q) and is independent of machine angle δ change in angle δ is caused by momentary change in generator speed. Therefore, load frequency and excitation voltage controls are non-interactive for small changes and can be modeled and analyzed independently. Furthermore, excitation voltage control is fast acting in which the major time constant encountered is that of the generator field; While the

power frequency control is slow acting with major time constant contributed by the turbine and generator moment of inertia. This time constant is much larger than that of the generator field. Thus, the transients in excitation voltage control vanish much faster and do not affect the dynamics of power frequency control.

Changes in load demand can be identified as: (i) slow varying changes in mean demand and (ii) fast random variations around the mean. The regulators must be designed to be insensitive to fast random changes, otherwise the system will be prone to hunting resulting in excessive wear and tear of rotating machines and control equipment.

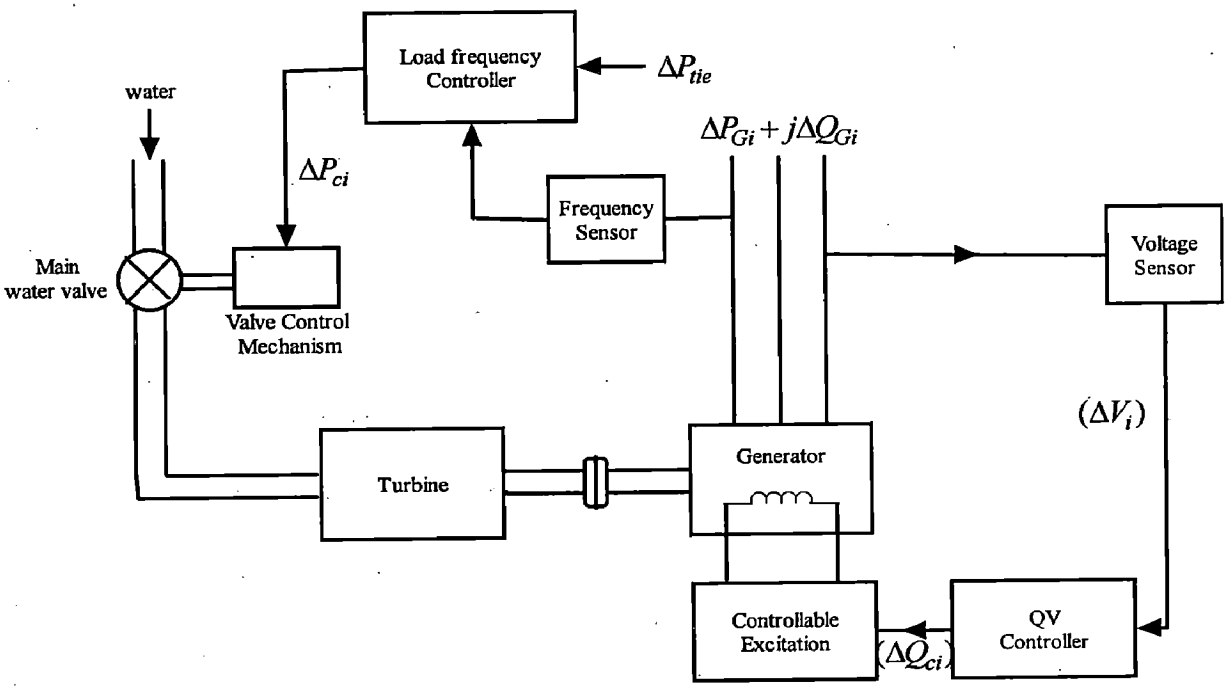


Fig 4.1 Basic Block Diagram for the Excitation and Governor control for an Hydro-Electric unit [6]

4.2 Hydraulic Channel Model:

Hydraulic turbines are of two basic types: impulse turbine and reaction turbine. The impulse turbine (also known as pelton wheel) is used for high heads 300m or more and the runner is at atmospheric pressure. In a reaction turbine, the pressure within the turbine is above the atmospheric pressure. The Francis turbine is used for heads up to 360m and the propeller turbine is for low heads up to 45meters [9].

Model without Surge Tank:

In order to simplify the model following assumptions are made

i. The hydraulic resistance is negligible.
ii. The penstock pipe is inelastic and water is incompressible.
iii. The velocity of water varies directly with the gate opening and with the square root of the net head.
iv. The turbine output is proportional to the product of the head and volume flow.

The Simulink diagram represents a simple nonlinear hydraulic channel configuration with unrestricted head, tailrace and without surge tank. Moreover, the model is achieved from the following relations.

Ideal gate opening is given by,

$$G = A_t * g$$

Where A_t is the gain of the turbine,

$$A_t = \frac{1}{g_{fl} - g_{nl}}$$

No-load water velocity,

$$U_{nl} = A_t g_{nl} \sqrt{H_0}$$

Hydraulic head at Gate,

$$H = \left[\frac{U}{G}\right] * \left[\frac{U}{G}\right]$$

Water starting time, $$T_w = \frac{LU_r}{a_g H_r}$$

Mechanical power output is given by, $$P_m = P - PL$$

Where PL represents the fixed power loss of the turbine, $$PL = U_{nl} * H$$

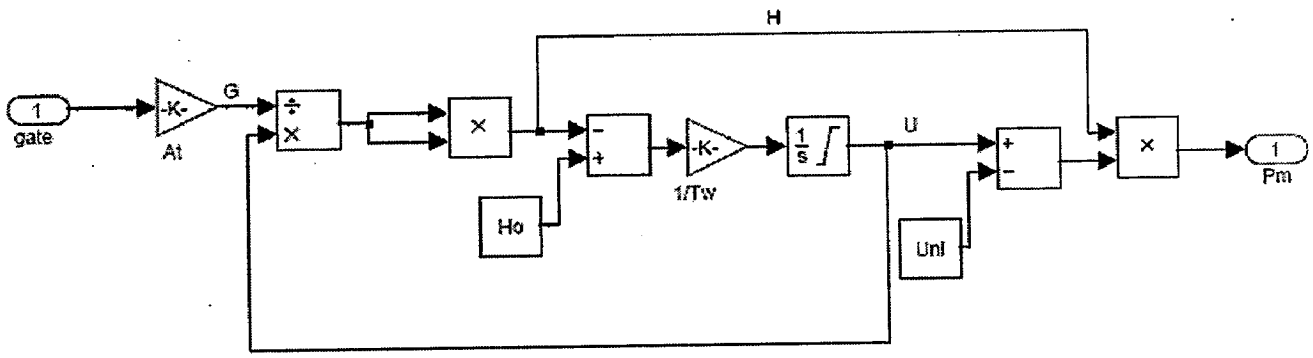


Fig 4.2 Hydraulic Channel Model in Simulink

Per unit conversion factor, $$P_r = \frac{turbineMWrating}{baseMVA}$$

So, per unit turbine power on generator MVA base is expressed as:

$$P_m = (U - U_{nl})P_r$$

4.3 Governor Model:

Water turbine governors are supplied for the basic purpose of governing the speed. In modern power plant, the role of governor is to accept and react to external signals and to combine them to achieve a desired operating mode. This governor can be either (i) electro-hydraulic type or (ii) PID type [9].

In the following section mathematical treatments for these are given [5].

4.3.1 PID governor:

The power frequency control is slow acting with major time constant contributed by the turbine and generator moment of inertia. This time constant is much larger than that of the generator field. Some electro-hydraulic governors are provided with three-term controllers proportional-integral-derivative (PID) actions. This allows the possibility of higher response speeds by providing both transient gain reduction and transient gain increase.

The derivative actions beneficial for isolated operation and particularly for plants with large water starting time. However, the use of a high derivative gain or transient gain increase will result in excessive oscillations and possibly instability when the generation unit is strongly connected to an interconnected system. Therefore derivative gain is usually set to zero. Without the derivative action, the transfer function of a PID (now PI) governor is equivalent to that of the mechanical hydraulic governor. Fig shows PID governor model with PID block in ModelSim.

4.3.2 Procedure for linking the two simulators [5]:

Procedure for linking the two simulators is explained as follows:

a. Developing the VHDL Code

The VHDL entity for the PID controller model will represent 8-bit streams of input and 16-bit output signal values with an IN port and OUT port of type STD_LOGIC_VECTOR. An input clock signal of type STD_LOGIC will trigger the registers when set:

1. Start ModelSim
2. Change to the writable directory "project", which have been created earlier.

```
ModelSim>cd C:/project
```

3. Open a new VHDL source edit window.
4. Add the VHDL code for PID controller
5. Save the file to project.vhd.

b. Compiling the VHDL File

This section explains how to set up a design library and compile project.vhd:

1. Verify that the file project.vhd is in the current directory by entering the ls command at the ModelSim command prompt.
2. Create a design library to hold the compilation results. To create the library and required _info file, enter the vlib and vmap commands as follows:

```
ModelSim> vlib work
ModelSim> vmap work work
```

If the design library work already exists, ModelSim *does not* overwrite the current library, but displays the following warning:

```
# ** Warning: (vlib-34) Library already exists at "work".
```

3. Compile the VHDL file by giving the tcl command

```
ModelSim> vcom project.vhd
```

c. Creating the Simulink Model

Now creating our Simulink model ie PID governor control. For this, first create a simple Simulink model that drives input into a block representing the VHDL project that coded in Developing the VHDL Code.

Start MATLAB, and Open a new model window. Then, open the Simulink Library Browser. Drag the required blocks from the Simulink Library Browser to our model window.

Next, configure the Constant block, which is the model's one of the input source:

i. Double-click the Constant block icon to open the Constant block parameters dialog. Enter the following parameter values in the **Main** pane:
   a. **Constant value**: 0
   b. **Sample time**: 10

   Later it can be possible to change these initial values to see the effect various sample times have on different simulation runs.

ii. Click the **Signal data types** tab. The dialog box now displays the **Output data type mode** menu.

   Select uint8 from the **Output data type mode** menu. This data type specification is supported by Link for ModelSim without the need for a type conversion. It maps directly to the VHDL type for the VHDL port, STD_LOGIC_VECTOR(7 DOWNTO 0).

iii. Click **OK**. The Constant block parameters dialog closes and the value in the Constant block icon changes to zero.

Next, configure the HDL Cosimulation block, which represents the pid controller model written in VHDL. Start with the **Ports** pane:

1. Double-click the HDL Cosimulation block icon. The Block Parameters dialog for the HDL Cosimulation block appears. Click the **Ports** tab.
2. In the **Ports** pane, select the sample signal /top/sig1 from the signal list in the center of the pane.

3. In the **Full HDL Name** edit field, replace the sample signal pathname /top/sig1 with /project/e. Then click the **Update** button. The signal name in the selected list entry changes. Similarly change the other signals as per in the VHDL top entity.
4. Select the sample signal /top/sig3. Click the **Delete** button. The signal is now removed from the list.

Now configure the parameters of the **Connection** pane:

1. Click the **Connection** tab.
2. Select socket from the **Connection method** list. This option specifies that Simulink and ModelSim will communicate via a designated TCP/IP socket port. Observe that two additional fields, **Port number or service** and **Host name**, are now visible.

Note that, because the **ModelSim running on this computer option** is selected by default, the **Host name** field is disabled. In this configuration, both Simulink and ModelSim execute on the same computer, no need to enter a remote host system name.

3. In the **Port number or service** text box, enter socket port number 4449 or, if this port is not available on the system, another valid port number or service name. The model will use TCP/IP socket communication to link with ModelSim. Note the entered parameter and that parameter has to be specified at the same socket port information when setting up the ModelSim for linking with Simulink.
4. Leave **Connection Mode** as **Full Simulation**.
5. Click **Apply**.

Now configure the **Clocks** pane:

1. Click the **Clocks** tab.

2. Click the **New** button. A new clock signal with an empty signal name is added to the signal list; the new signal is selected for editing.
3. In the **Full HDL Name** text box, enter the signal path /project/clk. Then select Rising from the **Edge** list. Set the **Period** parameter to 10.
4. Click the **Update** button.
5. Click **Apply.**

Next, enter some simple Tcl commands to be executed before and after simulation:

1. Click the **Tcl** tab.
2. In the **Pre-simulation commands** text box, enter the following Tcl command:

   echo "Running governor control in Simulink!"

3. In the **Post-simulation commands** text box, enter

   echo "Done"

4. Click **Apply**.

Next, view the **Timescales** pane to make sure it is set to its default parameters.

1. Click the **Timescales** tab.
2. The default settings of the **Timescales** pane are shown below. These settings are required for correct operation of this example. See Representation of Simulation Time for further information.
3. Click **OK** to close the Function Block Parameters dialog box.

In the same way configure the A/D converter and decoder according to the system.

The final step is to connect the blocks, configure model-wide parameters, and save the model:

1. Connect the blocks as shown in fig 4.3.

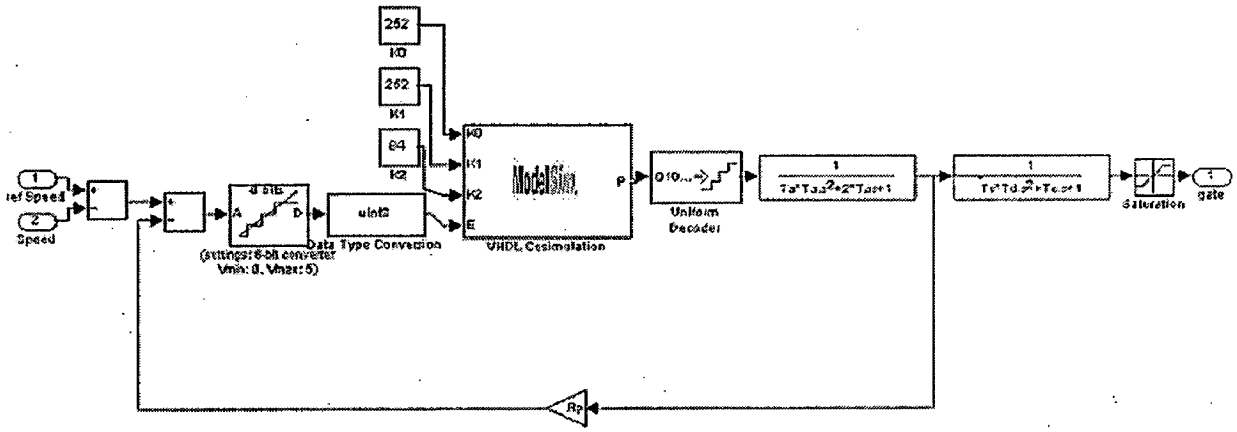


Fig 4.3 PID Governor Model in Simulink

2. Configure the Simulink solver options for a fixed-step, discrete simulation; this is required for correct cosimulation operation.
   a. Select **Configuration Parameters** from the **Simulation** menu in the model window. The Configuration Parameters dialog box opens, displaying the **Solver options** pane.
   b. Select Fixed-step from the **Type** menu.
   c. Select discrete (no continuous states) from the **Solver** menu.
   d. Click **Apply**. The **Solver options** pane should appear as shown below.
   e. Click **OK** to close the Configuration Parameters dialog box.
3. Save the model.

4.4 GOVERNOR CONTROL:

The governor controller for hydroelectric unit is achieved by using the developed models of hydraulic channel model without surge tank and PID governor model. It is shown in the fig 4.4 in the bus bar mode.

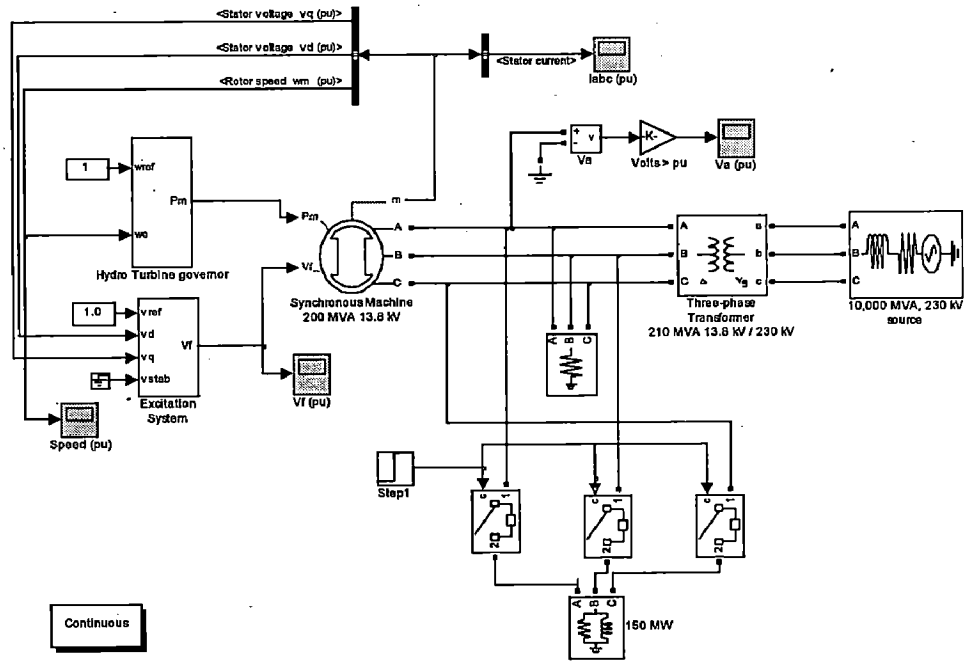


Fig 4.4 Governor Control of a Hydro-electric Unit

4.4.1 Setting Up ModelSim for Use with Simulink:

To start ModelSim such that it is ready for use with Simulink, enter the following command line in the MATLAB Command Window:

```
>>vsim
```

4.4.2 Loading Instances of the VHDL Entity for Cosimulation with Simulink:

The vsimulink command is a Link for ModelSim variant of the ModelSim vsim command. It is made available as part of the ModelSim configuration.

To load an instance of the project entity,

1. Change the input focus to the ModelSim window.
2. If necessary, change the directory to the location of the project.vhd file. For example:
3. ModelSim> cd C:/project
4. Enter the following vsimulink command:
5. ModelSim> vsimulink work.pid

ModelSim starts the vsim simulator such that it is ready to simulate entity project in the context of the Simulink model.

4.4.3 Running the Simulation:

This section guides scenario of running and monitoring a cosimulation session.

1. Open and add the project signals to a **wave** window by entering the following ModelSim command:

2. VSIM *n*> add wave /project/*
3. Change the input focus to the Simulink model window.
4. Start a Simulink simulation. Also, note the changes that occur in the ModelSim **wave** window. Zoom can be used to get a better view of the signal data. Note the change in the sample time in the **wave** window.

4.4.4 Shutting Down the Simulation:

This section explains how to shut down a simulation in an orderly way:

1. In ModelSim, stop the simulation by selecting **Simulate** > **End Simulation**.
2. Quit ModelSim.
3. Close the Simulink model window.

CHAPTER-5

RESULTS AND DISCUSSIONS

In this chapter, results of controller for hydroelectric unit, which is developed in chapter-4, are presented. The rotor speed and output electric power of the machine for different loads are presented.

5.1 Starting from standstill:

The machine is started from standstill position with a initial load of 0.05 p.u. The oscillations in rotor speed and electric power due to sudden change are settled in 15 seconds can be observed from the fig 5.1 to fig 5.6.

5.2 Loading the Machine:

The Machine is loaded at 20 seconds after starting the machine from standstill position. Because the machine is already settled at reference point, the oscillations due to sudden loading settled in 10 seconds, which is less than the case in 5.1.

5.2.1 Rotor Speed:

As the load on the machine is increased, the peak value of the oscillations in the rotor speed is also increasing. It has been observed from fig 5.1, fig 5.3 and fig 5.5.

5.2.2 Electric Power Output:

As the load on the machine is increased, the peak value of the oscillations in the Electric power is also increasing. At the time where the load is applied, it suddenly goes to that peak value and settles at the load. It has been observed from fig 5.2, fig 5.4 and fig 5.6.

5.3 Waveforms:

The waveforms for rotor speed and electric power output for different loads are shown below:

For 0.25 p.u load:

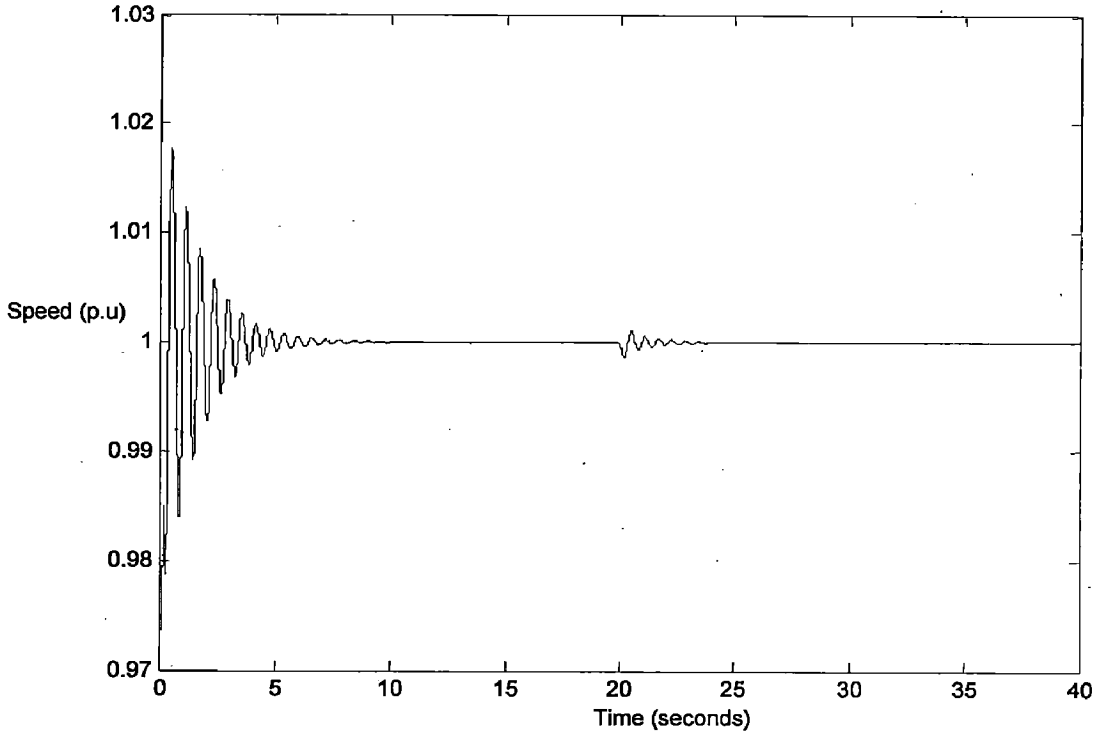


Fig 5.1 speed Vs time for 0.25 pu load

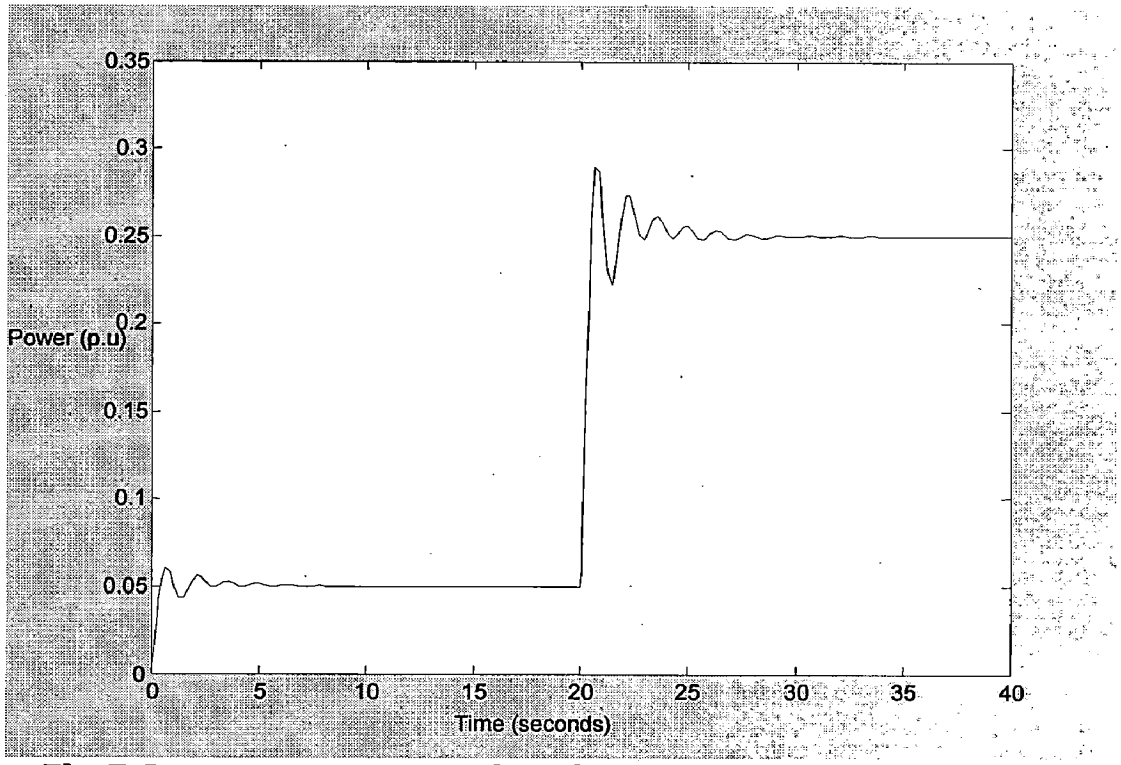


Fig 5.2 output power Vs time for 0.25 pu load

For 0.5 p.u load:

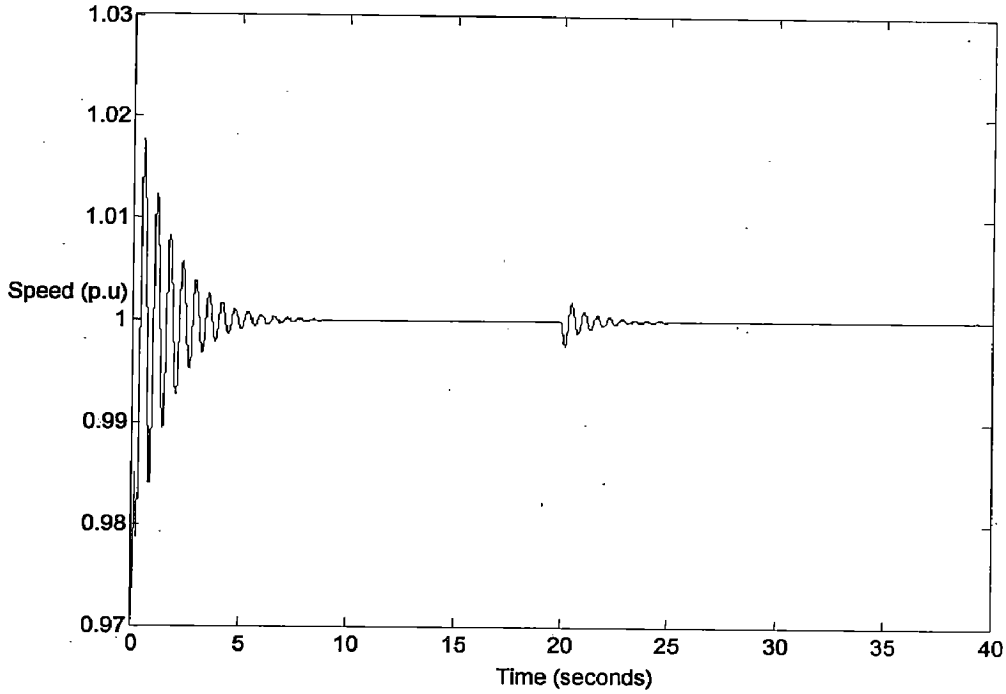


Fig 5.3 speed Vs time for 0.50 pu load

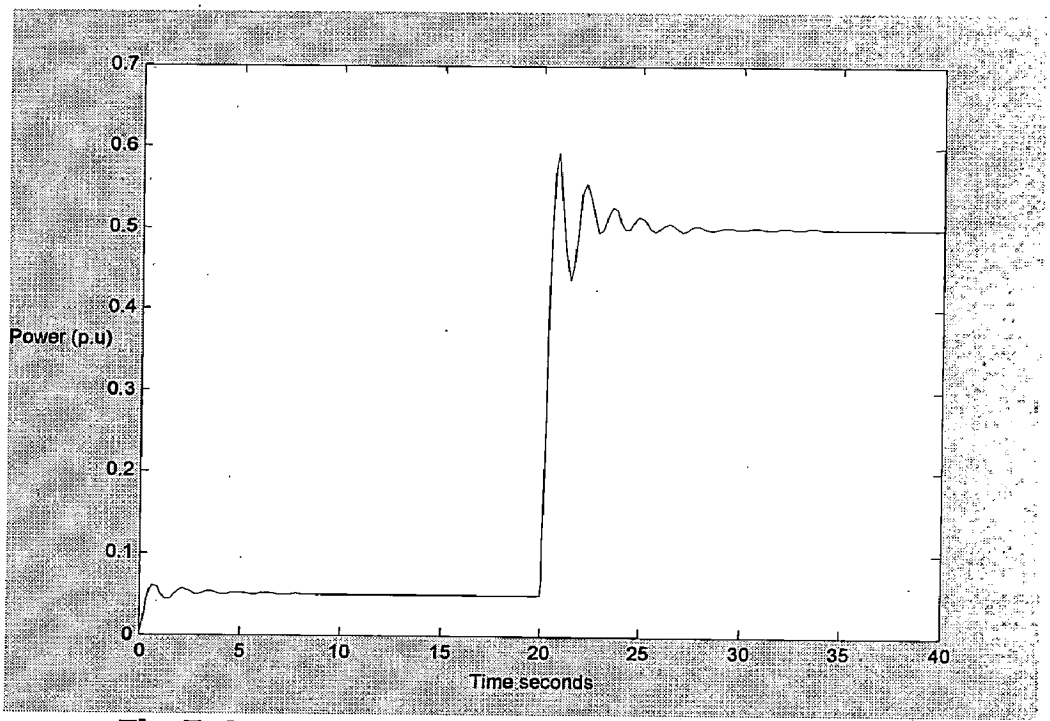


Fig 5.4 output power Vs time for 0.50 pu load

For 0.75 p.u load:

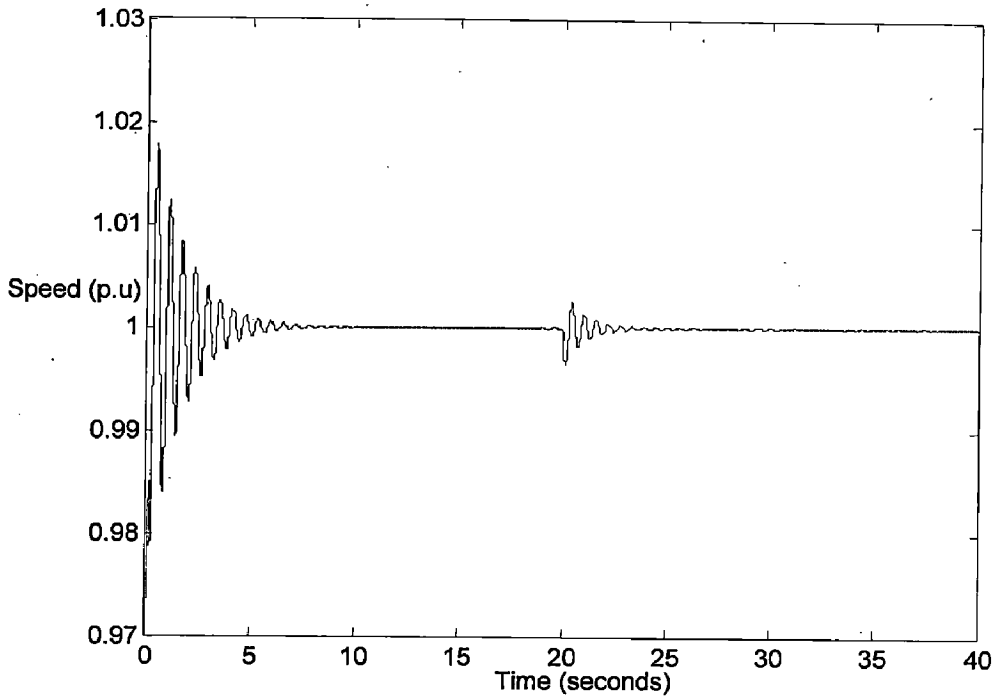


Fig 5.5 speed Vs time for 0.75 pu load

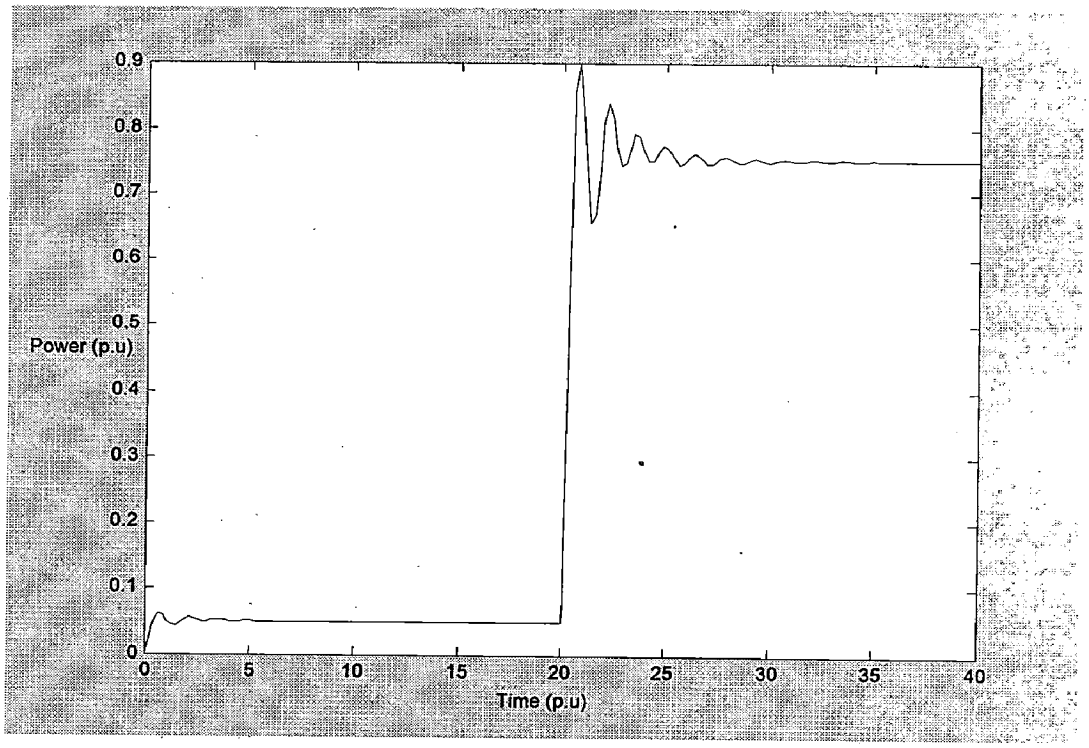


Fig 5.6 output power Vs time for 0.75 pu load

From the fig 5.7, it has been observed how the error signal is tends to zero and giving the appropriate output value in the Modelsim.

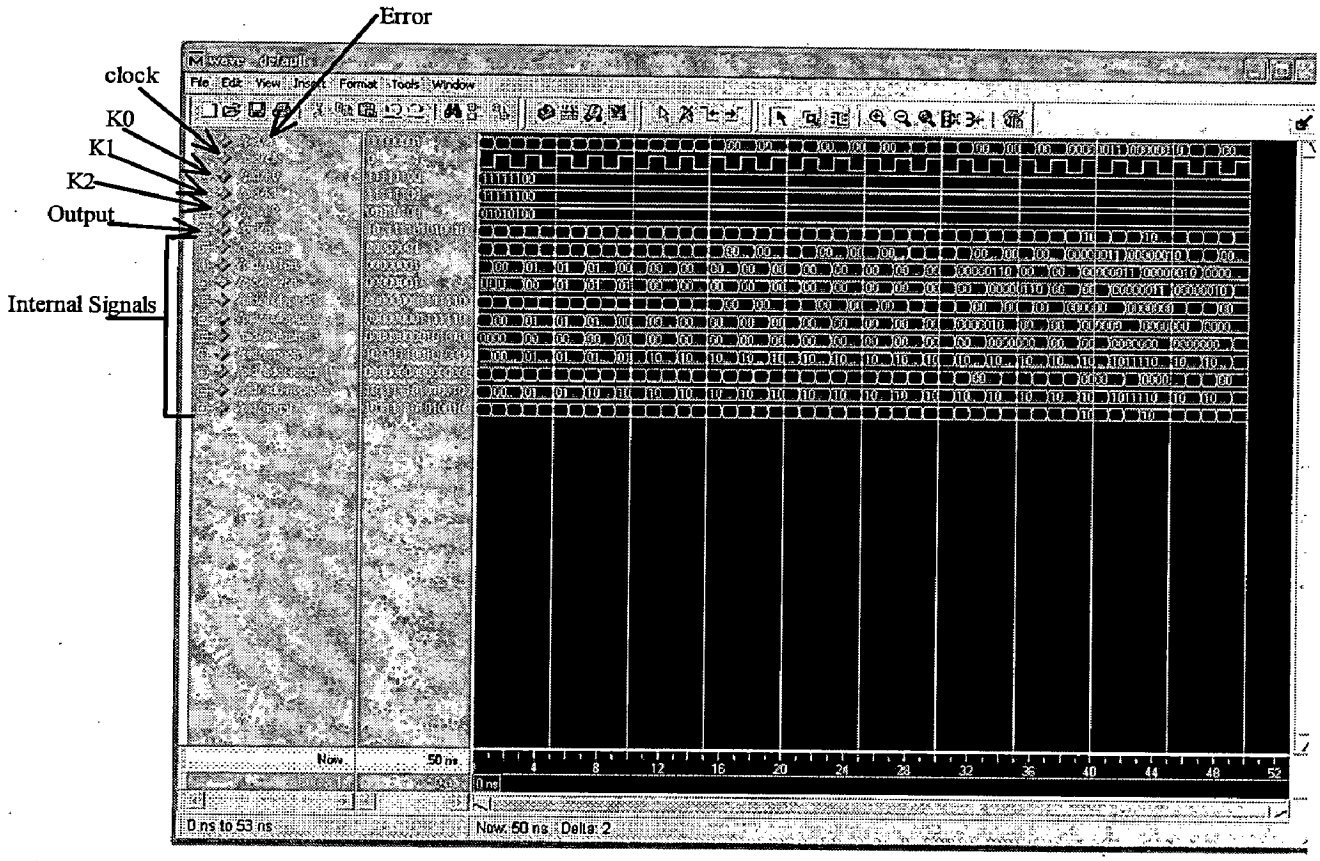


Fig 5.5 Waveform for the PID controller in ModelSim

CHAPTER-6

CONCLUSION AND SCOPE FOR FUTURE WORK

6.1 Conclusion:

The basic objective of developing a suitable FPGA based controller for hydroelectric unit is achieved. The basic PID controller is used to control the valve of the governor system of the hydroelectric unit to maintain the speed as well the active power at a constant value.

The PID controller is developed in VHDL, which is a hardware description language and is compiled in ModelSim simulator. In addition, the hydroelectric unit is simulated in MATLAB Simulink environment. The PID controller in ModelSim is incorporated in governor control in Simulink by using powerful cosimulation environment "Link for ModelSim".

Due to above said integration both the high performance of a hardware implementation and the validation and testing capabilities of a Simulink design has been achieved.

By proper tuning the PID controller the error signal which is the input for the PID in VHDL is forced to zero has been observed in waveform. In addition, by sudden application of load on the system it has been observed that there is a fluctuation in the speed of the system and it settles in around 10 seconds.

As per the expectations, all objectives and results are achieved. Anyhow, some of the aspects, which could not be included in the present development, can be taken up in the future work.

6.2 Scope for Future Work:

Two suggestions are made below for future work:

1. Further work can be carried out by Implementing above developed design for the system on a FPGA kit. On successful implementation, the same can be burnt on to a FPGA chip.

2. Presently the PID controller has been tuned by trial and error method. An auto-tuning algorithm can be developed in VHDL.

REFERENCES

1. Xilinx Corp. Spartan-3 Complete Datasheet Documentation. Xilinx, http://www.xilinx.com, August 2005.

2. D. L. Perry, "VHDL Programming by Example", TATA McGraw Hill Publications, 2004

3. J. Bhasker, "A VHDL Primer", PEARSON Education, 2005

4. M. MORRIS MANO, "DIGITAL DESIGN", PEARSON Education, 2004

5. http://www.mathworks.com/products/modelsim/

6. I J NAGARATH, D P KOTHARI, "Modern Power System Analysis", TATA McGraw Hill Publication.

7. I.J. Nagarath and M. Gopal, control systems Engineering, New age International, Third edition,1999

8. W. Zhao, B. H. Kim, A. C. Larson, and R.M. Voyles. "FPGA Implementation of Closed-Loop Control System for Small-Scale Robot". In Proceedings of the 12th International Conference on Advanced Robotics ICAR, 2005.

9. P. Kundur.,"Power System Stability and Control", McGraw Hill Publication. New York

10. João Lima, Ricardo Menotti, João M. P. Cardoso, and Eduardo Marques "A Methodology to Design FPGA-based PID Controllers"

11. Working Group on Prime Mover and Energy Supply Models for System Dynamic Performance Studies, "Hydraulic Turbine and Turbine Control Models for System Dynamic Studies", IEEE Transaction on Power System, Vol.-7, No. 1, pp. 167-179, February 1992.

12. L. Samet, N. Masmoudi, M.W. Kharrat, and L. Kamoun, "A Digital PID Controller for Real Time and Multi Loop Control: a comparative study", in Proceedings of 1998 IEEE International Conference on Electronics, Circuits and Systems, Vol.1, Sep. 7-10, 1998, pp. 291-296.

13. Jes´us L´azaro, Armando Astarloa, Jagoba Arias, Unai Bidarte, Aitzol Zuloaga Simulink/Modelsim Simulable VHDL PID Core for Industrial SoPC Multiaxis Controllers 1-4244-0136-4/06/2006 IEEE pages from 3007-3011

14. Mentor Graphics. ModelSim.