P- 2347

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report entitled **"BUCK CONVERTER – DRIVER CIRCUIT BASED ON VHDL"** is the bonafide work of

ANANDH G.	- Register No. 71204105002
SARANYA DEVI S.	- Register No. 71204105042
SINDHUJA C.	- Register No. 71204105051

who carried out the project work under my supervision.

[illegible]
Signature of the Head of the Department
Prof.K.Regupathy Subramaniam

B. [illegible] 17/4/08
Signature of the Guide
Mr.B.Karunamoorthy

Certified that the candidate with university Register No. 71204105002, 71204105042, 71204105051 was examined in project viva voce Examination held on 19.4.08

[illegible]
Internal Examiner

[illegible] 19/4/08
External Examiner

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE 641 006

ABSTRACT:

The strong demand for low-power computing has been driven by a growing variety of portable and battery-operated electronic devices. For instance, in many portable-computing devices such as MP3-players and digital cameras, the full processing capability of a processor is not always required. There are certain times when the operating frequency may be reduced.

In this project the system dynamically monitors circuit performance with a delay line and provides a substantially constant minimum supply voltage for digital processors to properly operate at a given frequency. In addition, the system adjusts the supply voltage to the required minimum under different process, voltage, and temperature and load conditions. Here the buck converter is used as the power delivery system. The gating signal to the buck converter is generated by VHDL coding according to the requirement.

The FPGA logic element for implementing driver Circuit of buck converter is programmed using VHDL language. It is simulated using MODELSIM XE 6.0a and synthesized by Xilinx ISE9.2i Web pack. It is implemented in a Xilinx FPGA device XC3S400 PQ208. The buck converter circuit is designed in MATLAB7.0 and the MATLAB7.0 is linked with MODELSIM6.0a to get PWM signal. MOSFET is used as switching element in buck converter

ACKNOWLEDGEMENT

The completion of our project can be attributed to the combined efforts made by us and the contribution made in one form or the other by the individuals we hereby acknowledge.

We would like to express our deep sense of gratitude and profound thanks to our guide **Mr.B.Karunamoorthy M.E,** Lecturer, Electrical and Electronics Engineering Department, for his valuable guidance, support, constant encouragement and co-operation rendered throughout the project.

We express our heart felt gratitude and thanks to the Dean / HOD of Electrical & Electronics Engineering, **Prof. K.Regupathy Subramanian B.E.(Hons), M.Sc.**, for encouraging us and for being with us right from beginning of the project and guiding us at every step.

We are also thankful to our teaching and non-teaching staffs of EEE and ECE departments for their kind help and encouragement.

We extend our sincere thanks to all our parents and friends who have contributed their ideas and encouraged us for completing the project.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

NO.	SYMBOLS	ABBREVIATIONS
01	FPGA	Field Programmable Gate Array
02	VLSI	Very Large Scale Integrated Circuit
03	LE	Logic Elements
04	IC	Integrated Circuit
05	VHDL	Very High Speed Integrated Circuit Hardware Descriptive Language
06	CLB	Configurable Logic Block
07	I/O	Input / Output
08	IOB	Input Output Block
09	HDL	Hardware Descriptive Language
10	CLK	Clock
11	DCLK	Delay Clock
12	SCLK	Sample Clock
13	ICLK	Input Clock
14	PVT	Process Voltage and Temperature
15	DPWM	Delay Pulse Width Modulation
16	UND	Up Normal Down
17	ECW	Error Compensated Word
18	FCW	Frequency Compensated Word
19	ADC	Analog to Digital Converter
20	DAC	Digital to Analog Converter
21	LUT	Look Up Table

CHAPTER - 1

1. INTRODUCTION

1.1 OBJECTIVE:

This project aims at adjusting the supply voltage to the required minimum constant value under different process, voltage and temperature and load conditions.

1.2 NEED OF THE PROJECT:

In many portable-computing devices such as MP3-players and digital cameras, the full processing capability of a processor is not always required. There are certain times when the operating frequency may be reduced; and a lower frequency means a longer allowable delay. This increased time margin also allows the supply voltage level to be lowered albeit with an increased propagation delay. Since power consumption is quadratic with supply voltage and proportional to operating frequency, reducing both allows excellent energy-efficient operation. This project is done using VLSI architecture. VLSI technology is 30-40 times faster than microcomputer and 10 times faster than signal processing. In addition the VLSI technology is reprogrammable and power consumption is less compared to conventional processing.

1.3 ORGANISATION OF THE REPORT:

Chapter 1: Introduction to the project stating the need for this approach, objective and Methodology

Chapter 2: Driver circuit – An overview

Chapter 3: Buck converter – An overview

Chapter 4: Details about the software implementation of Driver circuit

Chapter 5: gives the details about MATLAB implementation of Buck converter

Chapter 6: FPGA – Describes about the FPGA, its architecture, design flow and applications. Details the architecture and features of the FPGA device

XC3S400 used.

Chapter 7: Gives the details about the hardware implementation

Chapter 8: Conclusion – The project is done and the features of the project are discussed.

Appendices: Gives the features and the specification of the FPGA device used.

1.4 BUCK CONVERTER-DRIVER CIRCUIT:

The required minimum supply voltage is obtained using buck converter. The delay of the gate signal to the buck converter is inversely proportional to the supply voltage. Therefore, the application operates at a reduced clock frequency with a lower supply voltage at the cost of performance reduction to meet the low power requirement. This is achieved by PWM technique.

1.5 CONVENTIONAL TECHNIQUE FOR PWM:

The pulse-width modulator circuit consists of a saw-tooth generator, an error amplifier, and a comparator. The frequency of saw-tooth generator can usually be set by choosing proper values of an RC network. The error amplifier compares the reference voltage and the feedback signal. The feedback signal is obtained using a voltage divider network across the output of the buck converter circuit. The output of the error amplifier is compared with the saw-tooth waveform and when this voltage is greater than the output of saw tooth generator, the output of the comparator would be at logic '1'.

When the output of comparator is at logic '1', the switch in the buck converter circuit can be kept in the ON state. When the comparator is at logic '0', the switch in the buck converter circuit can be kept in the OFF state.

If the output voltage tends to be greater than the desired value, the output voltage of the error amplifier would fall and the duration for which the output of comparator remains at logic '1' would decrease. Thus the duty cycle of the switch reduces and the output of the buck converter would fall. When control by frequency modulation is desired, the ON-period is kept constant, but the frequency is varied in order to bring about regulation. Such a technique is necessary if the load on the regulator tends to be come very low. It is difficult to make the ON-period below certain time duration and when this limit is reached, control by pulse width modulation becomes impossible. Then the duty cycle is reduced by keeping the ON period fixed and increasing the cycle period. The value of minimum ON period depends on the transistor switch.

1.6 LIMITATIONS AND SOLUTIONS:

The disadvantage of analog circuits is that they tend to drift with time and they are difficult to tune. Analog circuits are usually hot and are sensible to noise. If we look at an analog signal, we can expect any real number from it and we can't identify for sure what's noise and what's signal. The analog system fails to maintain the voltage at constant level under different load conditions, temperature and process.

To overcome these limitations we are going for digital pulse width modulation. There is no problem of noise interference and heating of the circuit. It is very compact compared to conventional circuits. The digital PWM is achieved by means of VHDL coding.

1.7 METHODOLOGY:

The sequence of work carried out is been listed below,

VHDL coding:	Describes the circuit structure
Simulation:	To check the functionality of the code
MATLAB:	For designing Buck converter and simulation
Synthesis:	RTL is turned into design implementation in terms of logic gates creates a net list
Mapping:	Fits the design into available resources in the tool
Place & Route:	Determining the position & interconnecting the sub blocks
Timing simulation:	For timing analysis of CLBs and net delays
Bit stream:	A binary file in terms of 0s and 1s to program FPGA
Hardware Implementation:	Downloading the program to XILINX device and verifying the output through LED

CHAPTER-2

2. DRIVER CIRCUIT

2.1 BLOCK DIAGRAM:

A block diagram of the all digital self-adjusting minimum power supply system is presented in Fig. 1. The system consists of a closed-loop controller, a fixed frequency clock signal (CLK), frequency information (FI), a dc–dc buck-converter, and a processor as a load current source. The loop controller consists of a slacktime detector, a voltage adjuster, and a PWM modulator. The presented controller performs the following discrete-timecompensation:

$$D(n+1)=F(n)+\alpha E(n)+A(n) \tag{1}$$

Where D(n+1) is the next value of the duty ratio,and F(n),E(n)are the frequency compensation, the current values of the detected error, and A(n) the accumulated compensation, respectively, and α is error scaling factor. As in (1), the control approach does not request any previous values, while an accumulated compensation factor is needed. The variations of PVT and load are dynamically updated in the compensation factor. During updating, the duty ratio is controlled by a finite-state machine (FSM) which maintains a substantially constant supply voltage. The scaling coefficient is used to increase the error resolution and it is implemented by the shift-right function instead of multiplication.

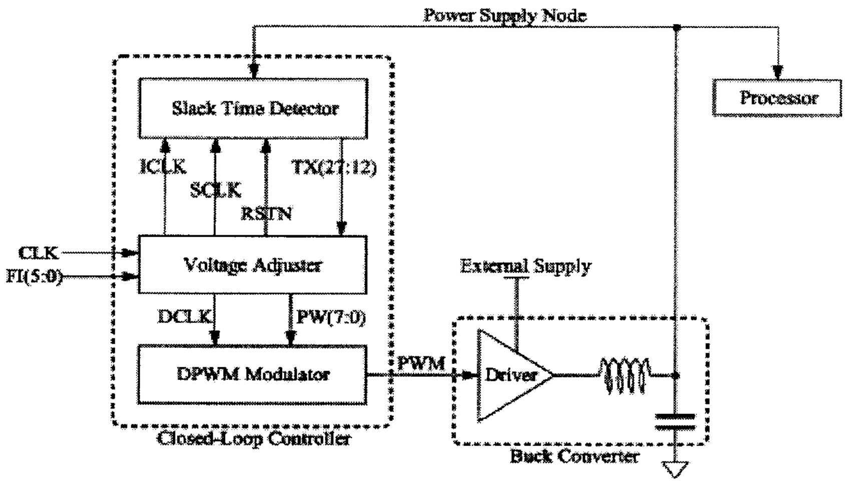


FIG 1:BLOCK DIAGRAM OF DRIVER CIRCUIT.

2.2 SLACKTIME DETECTOR:

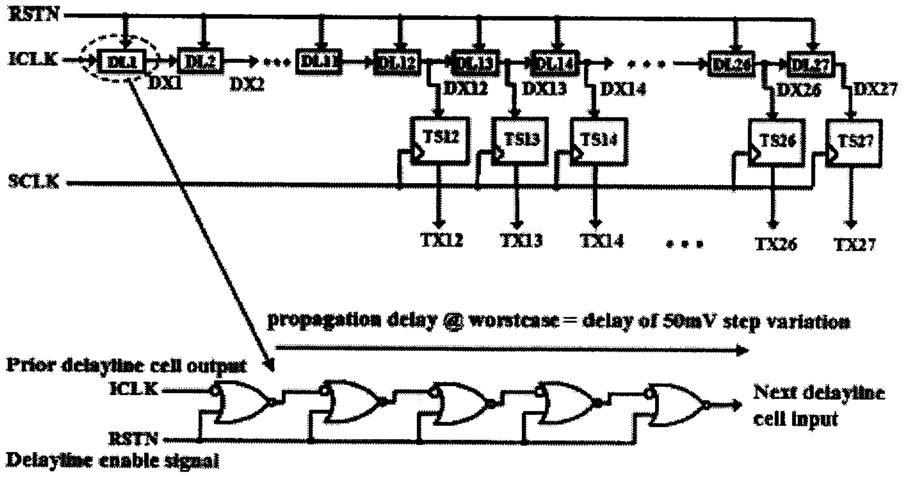


FIG 2:CIRCUIT DIAGRAM OF SLACK TIME DETECTOR

Slack time detection is the ability to determine the minimum voltage required for a given operating frequency. It requires continuous monitoring of a critical path delay through the digital circuitry with respect to PVT, load, and frequency. The proposed slacktime detector consists of a chain of delay cells and a tap register as shown in Fig. 2. The principle of a delayline as an ADC, is based on the relation between supply voltage and propagation delay. The slacktime detector determines the voltage level of the delayline's supply based on the propagation delay through the delayline. In other words, the delay, along with the supply voltage, is converted to a digital value by sampling the delayline.

There are three considerations for designing a delayline to monitor the critical path timing over PVT variations. The first requirement is to avoid the worst case crossover effect on the nonlinear characteristics between delay and voltage. The crossover becomes worse when the delay increases longer or the voltage decreases. Therefore, the margin for proper circuit operation is applied for high-crossover ratio. Determination of the resolution of delayline is the second consideration. A fine step size results in very slowsettling while a coarse step size can cause hysteretic oscillation.

The last requirement is to minimize the hardware burden for delayline. As semiconductor fabrication technology improves, circuit delay is shorter and, in turn, the number of needed cells to implement a critical path delay is larger. From the last delayline design requirement, the cell of a delayline should have low performance. A NOR structure slowly transits at high-to-low transition compared to a NAND circuit's transition. However, low-to-high transition time of a NOR gate is the same as that of a NOT gate and is faster than a NAND gate. Therefore, a pair of NOR and NOT gates is selected as the unit delay cell. The number of delay cells between taps is determined by increasing a step voltage so the accumulated voltage steps require one more tap active than the prior accumulated steps voltage at the worst case.

The number of delay cells between taps varies because the propagation delay is not a linear function of supply voltage. The inverted input of the delay cell (NOR-INV) receives the input clock signal or the output of prior delay cell; and the noninverted input is connected to a delayline enable signal. The sampling clock signal lags the input clock signal by a 1/4 of a period. The tap register samples the values of the delayline at 1/4 period intervals after the input clock pulse begins to propagate through the delayline. The magnitude of the supply voltage is inferred by determining how far along the delayline the input clock pulse propagatesin a 1/4 period. Therefore, a delayline in the negative feedback path of a closed loop reflects variations in circuit performance in response to variations of PVT, load, and frequency, and adaptively scales the regulated voltage of a buck converter via a loop controller. The delayline is characterized at the worst case with regard to fixed-frequency input sources. Delay of the delayline implies its process corner, junction temperature, and supply voltage at a given work load. From this measurement, a desirable constant supply voltage is determined in response to PVT and load variations. This guarantees the propagation delays just less than the critical path delay limitation and assures proper operation.

2.3 TIMING DIAGRAMS:

2.3.1 TIMING DIAGRAM OF SLACKTIME DETECTOR:

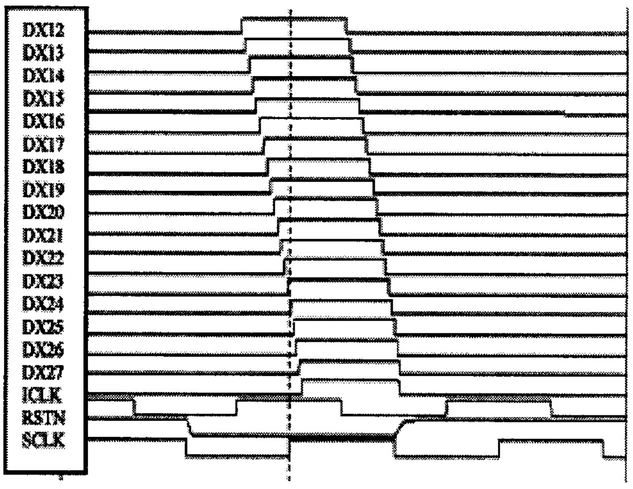


FIG 3:TIMING DIAGRAM OF SLACK TIME DETECTOR

The figure shows the timing of the inputs DX12- DX27 at each tap, the input clock signal (ICLK), the sample clock signal (SCLK), and delayline enable signal (RSTN).

2.3.2 TIMING DIAGRAM OF CLOCKS:

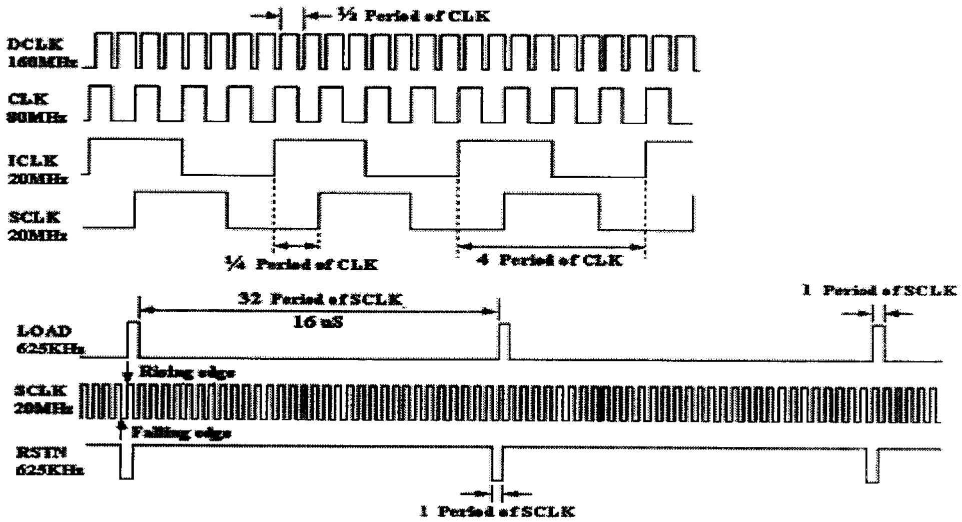


FIG 4: TIMING DIAGRAM OF CLOCKS

2.4 VOLTAGE ADJUSTER:

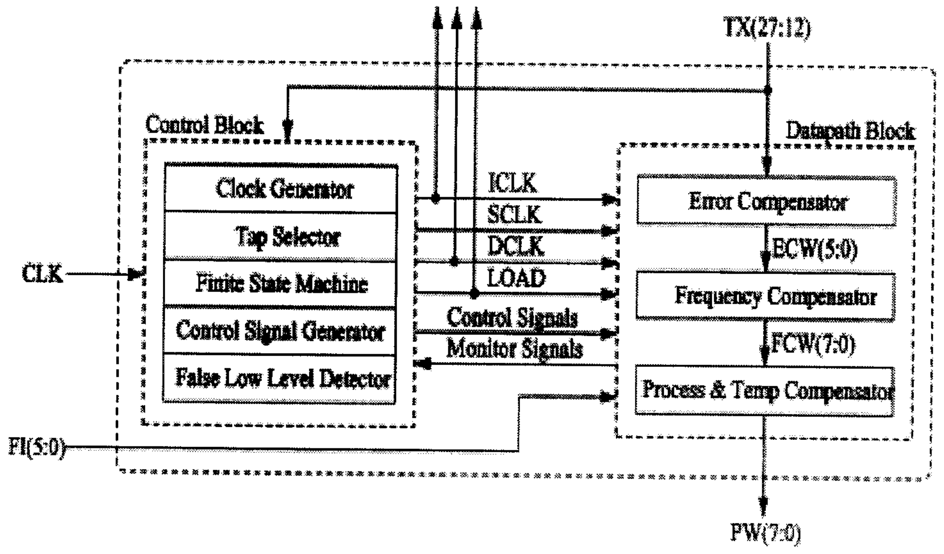


FIG 5:BLOCK DIAGRAM OF VOLTAGE ADJUSTER

The voltage adjuster consists of an error compensator, a frequency compensator, a process, voltage, and temperature compensator, and a control block as shown in Fig. 5. The major role of the voltage adjuster is to compensate a supply voltage error at a given frequency from the measurement of the slack time detector and to provide a desirable constant voltage level against variations of frequency as well as PVT. In addition, for high-speed and low-overshoot/undershoot start-up, it controls soft-start operation.

2.4.1 ERROR COMPENSATOR:

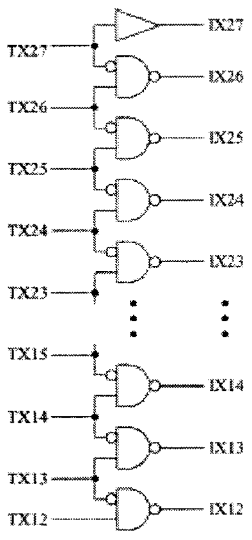


FIG 6:CIRCUIT DIAGRAM OF LOW LEVEL DETECTOR

The role of the error compensator is to detect the voltage error,E(n) as in (2), and to generate a proportionally compensated value. It receives the propagation delay word TX(27:12) from the slacktime detector and detects the position of one and zero pair of taps as shown in Fig 6.

2.4.2 DATA USED FOR ERROR COMPENSATOR:

	One-shot Pulse	Detected Voltage	Binary Detected Voltage	Voltage Error	Binary Voltage Error (2's C)	Proportionally Compensated Voltage
	IX27	1.35	011011	+0.15	000011	010101
	IX26	1.30	011010	+0.10	000010	010110
	IX25	1.25	011001	+0.05	000001	010111
Reference	IX24	1.20	011000	0.00	000000	011000
	IX23	1.15	010111	-0.05	111111	011001
	IX22	1.10	010110	-0.10	111110	011010
	IX21	1.05	010101	-0.15	111101	011011
	IX20	1.00	010100	-0.20	111100	011100
	IX19	0.95	010011	-0.25	111011	011101
	IX18	0.90	010010	-0.30	111010	011110
	IX17	0.85	010001	-0.35	111001	011111
	IX16	0.80	010000	-0.40	111000	100000
	IX15	0.75	001111	-0.45	110111	100001
	IX14	0.70	001110	-0.50	110110	100010
	IX13	0.65	001101	-0.55	110101	100011
	IX12	0.60	000000	-0.60	110100	100100
						ECW(5:0)

The compensator converts the propagation delay position to an error voltage by comparing it to the reference delay position (shown in the table given above) along with supply voltage under worst case conditions. In turn, it generates a proportionally compensated propagation delay word ECW (5:0) that represents a reference value at a default frequency plus a compensated error value.

2.4.3 FREQUENCY COMPENSATOR:

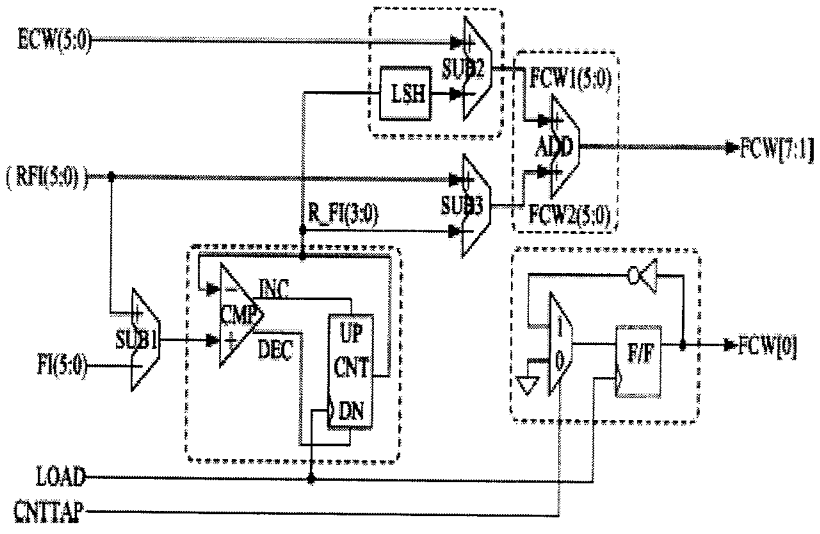


FIG 7: CIRCUIT DIAGRAM OF FREQUENCY COMPENSATOR

The frequency compensator adjusts the duty cycle of the PWM pulse based on the desired supply voltage at a given frequency. The first subtractor, SUB1, in Fig. 7 generates a difference between the frequency information, FI(5:0), and the internal reference voltage level, RFI(5:0). The difference implies the desirable voltage variation in response to a given frequency. The up/down counter (CNT) receives the difference and counts up or down at the load signal (LOAD) until the output of the counter is equal to the difference. This prevents the supply level and frequency variation from abruptly changing and reduces ringing. The second subtractor, SUB2, receives the proportionally compensated propagation delay word ECW(5-0) from the error compensator and the shift-lefted counter number for the subtrahend, and generates a frequency compensated propagation delay word FCW1(5-0).

The compensated error-step from the delayline is the same as the resolution of the delayline (6 bits). However, the resolution of theDAC is higher than that of the ADC, and the error step also has a higher resolution. To increase the control resolution, the proportionally compensated error-value should be scaled close to the resolution of the DAC. The third subtractor, SUB3, provides
a reference supply voltage word FCW2(5:0) at a given frequency FI(5:0). The one-bit higher resolution compensated word FCW(7:1) is generated by adding the frequency compensated word to the reference supply voltage word. However, a 7-bit DAC for a 6-bit ADC is insufficient to avoid limit-cycling (Section II). Therefore, dither logic generates a least significant bit (LSB) of the frequency compensated word FCW(0).

2.4.4 PROCESS, VOLTAGE, AND TEMPERATURE COMPENSATOR:

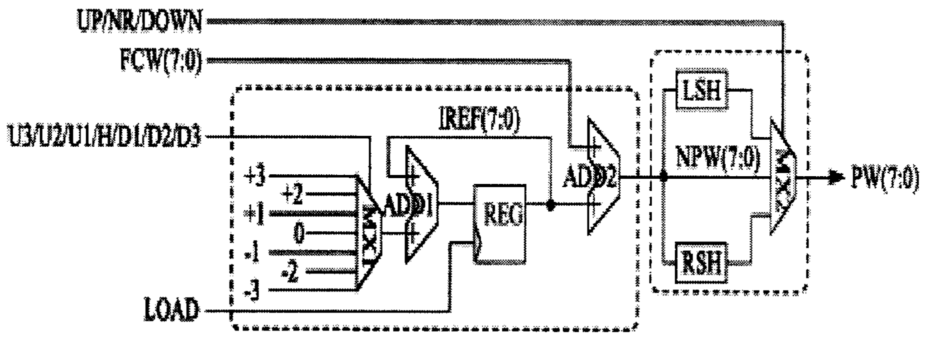


FIG 8: CIRCUIT DIAGRAM OF PVT COMPENSATOR

It consists of an internal dynamic voltage reference source, a pulse width generator, and a ringing stopper as shown in Fig. 8. The internal dynamic voltage reference source adds or subtracts one, two, or three steps according to the increment/decrement indicators U1, U2, U3, D1, D2, D3, and generates an internal dynamic voltage reference, IREF (7-0). The reference value compensates the fluctuations due to process and temperature variations as well as the quantization error of the external supply voltage. Fig. 12 illustrates the equivalent supply voltages which ensure the same propagation delay at different operational and intrinsic parameters. The pulse width generator, ADD1, receives the frequency compensated word

FCW(7:0) and the accumulated compensation [as in (2)] IREF(7-0), and generates a normal PWM pulse width NPW(7-0). The ringing stopper receives three inputs: a shift-lefted PWM pulsewidth, a normal PWM pulse width, and a shift-righted PWM pulsewidth. It outputs a pulsewidth word PW(7-0) in response to selection signals UP, NR, and DOWN. Since the high-valued derivative direction of supply voltage during frequency-changing or starting-up is unchanged by the step-size compensated value NPW(7:0), emphasized activation (double or half size of PWM pulse) is needed.

2.5 DPWM MODULATOR:

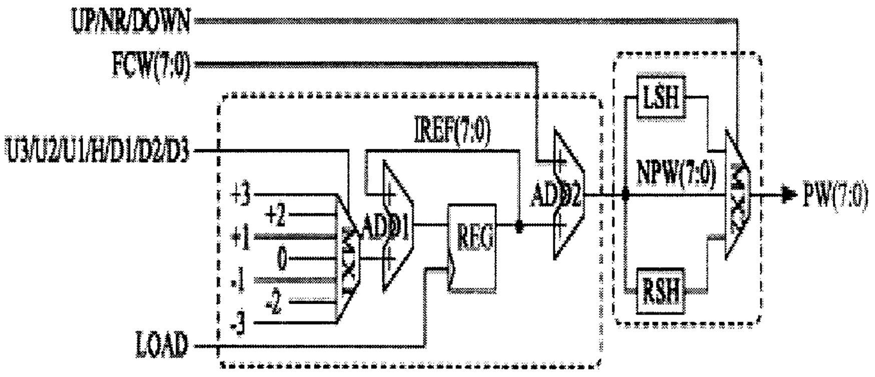


FIG 9:CIRCUIT DIAGRAM OF DPWM MODULATOR

The DPWM pulse modulator in Fig. 9 consists of a loadable down counter and a pulse generator. Counter loads the PWM pulsewidth PW(7-0) from the voltage adjuster by the LOAD,changing a binary output of counter as a count in response to the DCLK. DPWM pulse modulator outputs a pulse modulated signal PWM defined by the binary input value PW(7-0) of counter.

CHAPTER-3

3. BUCK CONVERTER

3.1 CIRCUIT:

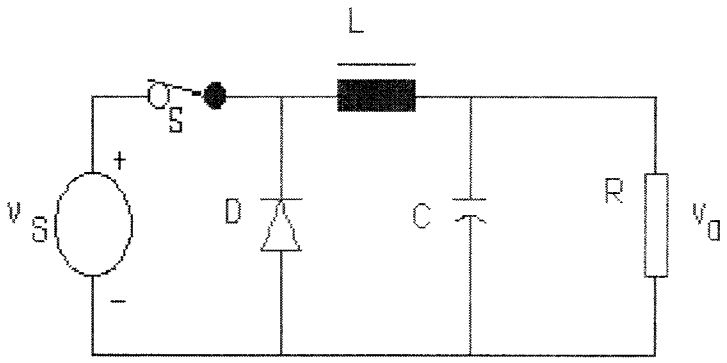


FIG 10: BASIC CIRCUIT DIAGRAM OF BUCK CONVERTER

3.2 STATES OF OPERATION:

The operation of the buck converter is explained first. This circuit can operate in any of the three states as explained below.

3.2.1 FIRST STATE :

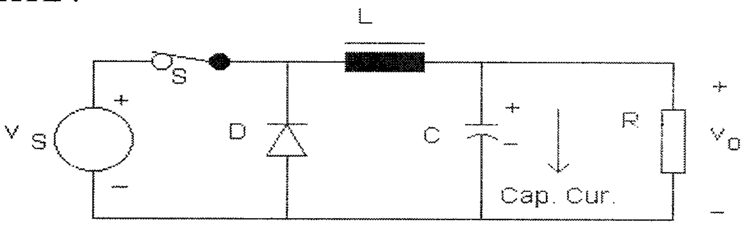


FIG 11: BUCK CONVERTER IN STATE1(a)

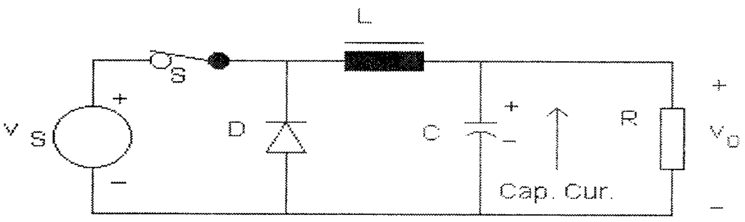


FIG 12: BUCK CONVERTER IN STATE1 (b)

The first state corresponds to the case when the switch is ON. In this state, the current through the inductor rises, as the source voltage would be greater than the output voltage, whereas the capacitor current may be in either direction, depending on the inductor current and the load current. When the inductor current rises, the energy stored in it increases. During this state, the inductor acquires energy.

When the switch is closed, the elements carrying current are shown in red colour in Fig. 11, whereas the diode is in gray, indicting that it is in the off state. In Fig. 11, the capacitor is getting charged, whereas it is discharging in Fig. 12.

The equations that govern the operation of the circuit in the first state are shown below.

$$\frac{di_L}{dt} = \frac{v_S - v_O}{L} \qquad (1)$$

$$\frac{dv_O}{dt} = \frac{i_L - {}^{v_O}\!/_R}{C} \qquad (2)$$

3.2.2 SECOND STATE:

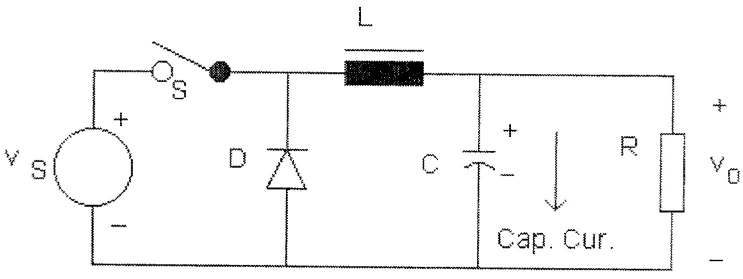


FIG 13:BUCK CONVERTER IN STATE2(a)

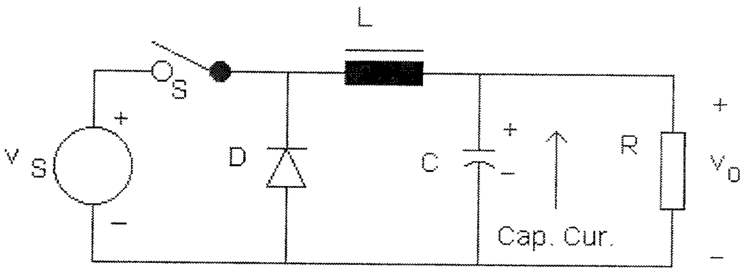


FIG 14:BUCK CONVERTER IN STATE2(b)

The second state relates to the condition when the switch is off and the diode is ON. In this state, the inductor current free-wheels through the diode and the inductor supplies energy to the RC network at the output. The energy stored in the inductor falls in this state. In this state, the inductor discharges its energy and the capacitor current may be in either direction, depending on the inductor current and the load current .The equations that govern the operation of the circuit in the second state are shown below.

$$\frac{di_L}{dt} = -\frac{v_O}{L} \quad (3)$$

$$\frac{dv_O}{dt} = \frac{i_L - {}^{v_O}\!/_{R}}{C} \quad (4)$$

3.3.3 THIRD STATE:

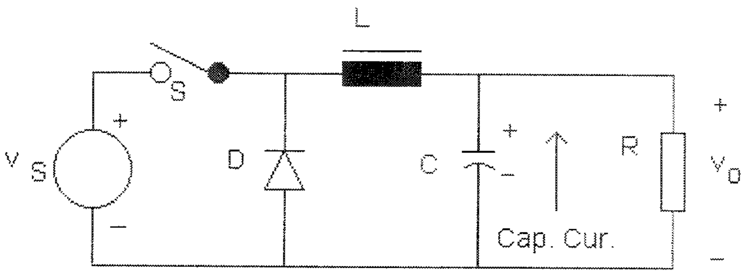
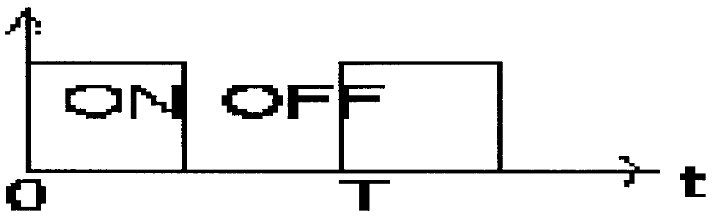


FIG 15:BUCK CONVERTER IN STATE3

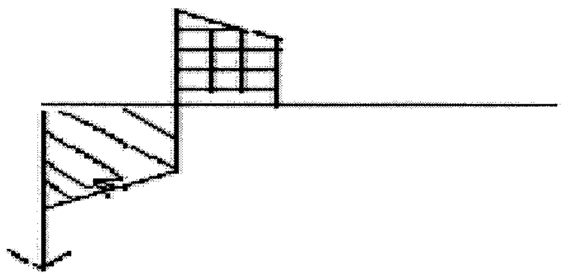
When the switch is open, the inductor discharges its energy. When it has discharged all its energy, its current falls to zero and tends to reverse, but the diode blocks conduction in the reverse direction. In the third state, both the diode and the switch are OFF and Fig.15 illustrates the third state. During this state, the capacitor discharges its energy and the inductor is at rest, with no energy stored in it. The inductor does not acquire energy or discharge energy in this state. The equation that governs the operation of the circuit in the third state is shown below.

$$\frac{dv_O}{dt} = -\frac{{}^{v_O}\!/_{R}}{C} \quad (5)$$

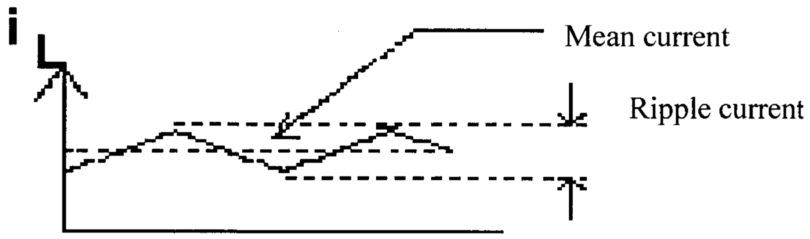
3.3 WAVE FORMS OF THE BUCK CONVERTER:



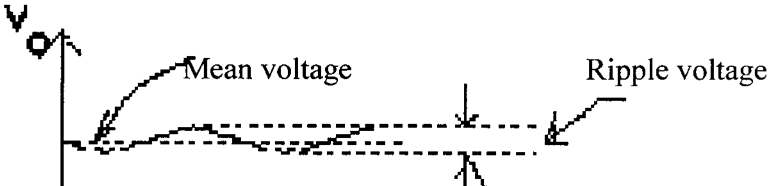
Votage Across Inductor:



Inductor Current:



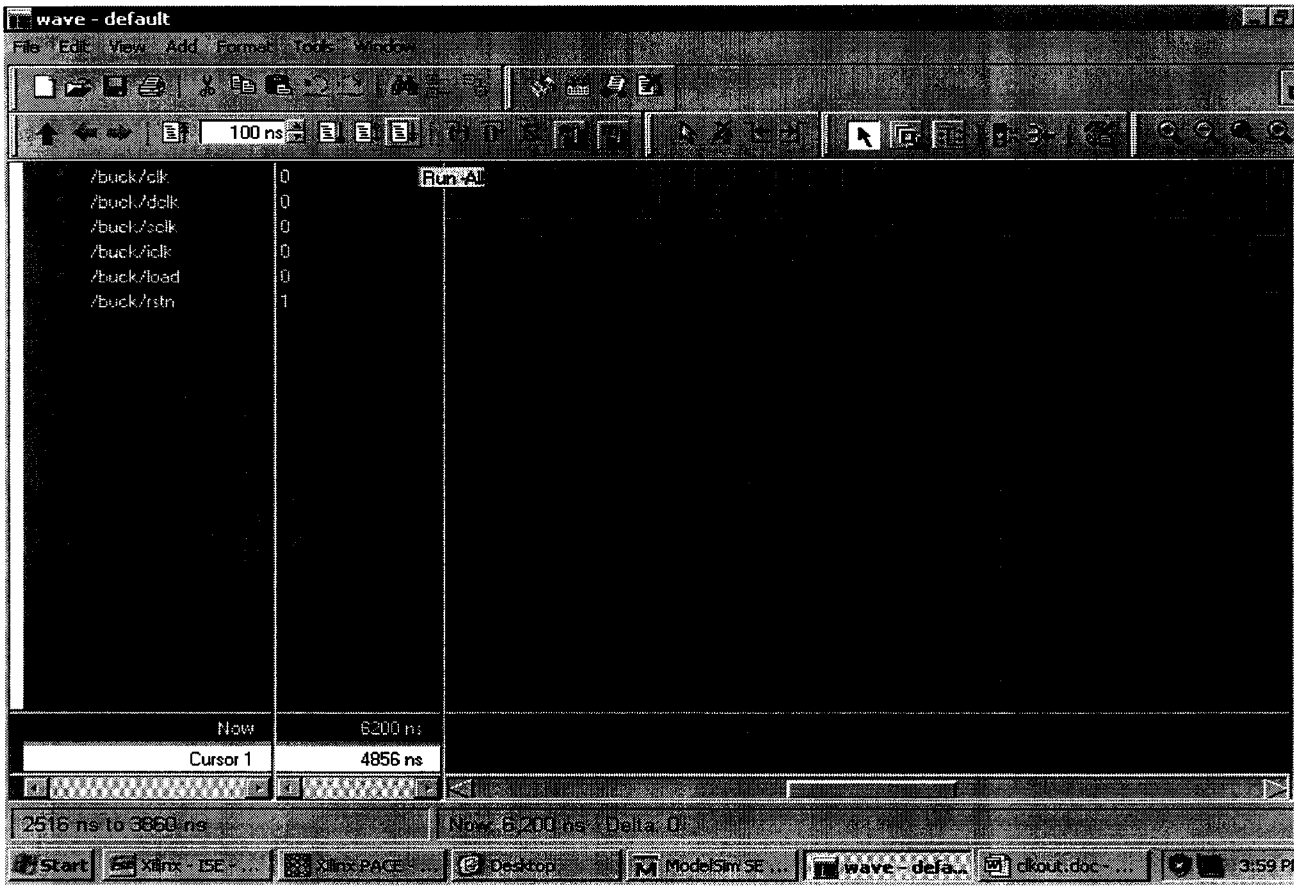
Output Voltage:



CHAPTER-4

4. SOFTWARE IMPLEMENTATION

4.1 SIMULATION RESULT OF CLOCK GENERATOR:



```
/buck/clk     0
/buck/dclk    0
/buck/sclk    0
/buck/iclk    0
/buck/load    0
/buck/rstn    1
Now           6200 ns
Cursor 1      4856 ns
2516 ns to 3860 ns
Now: 6,200 ns  Delta: 0
```

The figure shows the simulation result of the clock generator.

The total time period of CLK=12.5ns.

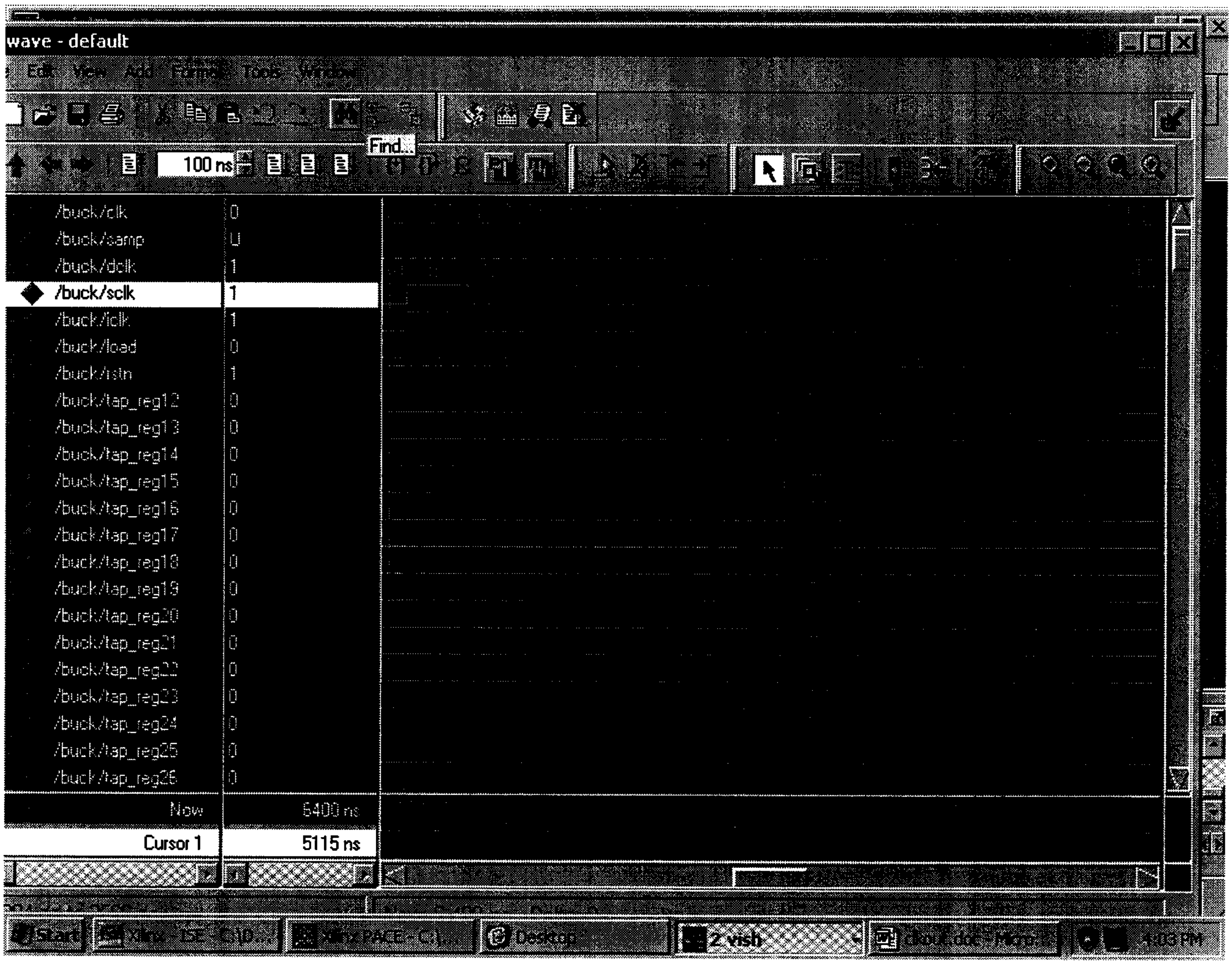
The total time period of DCLK=6.25ns.

The total time period of SCLK=50ns.

The total time period of ICLK=50ns.

The SCLK lags behind ICLK by 3.125ns.

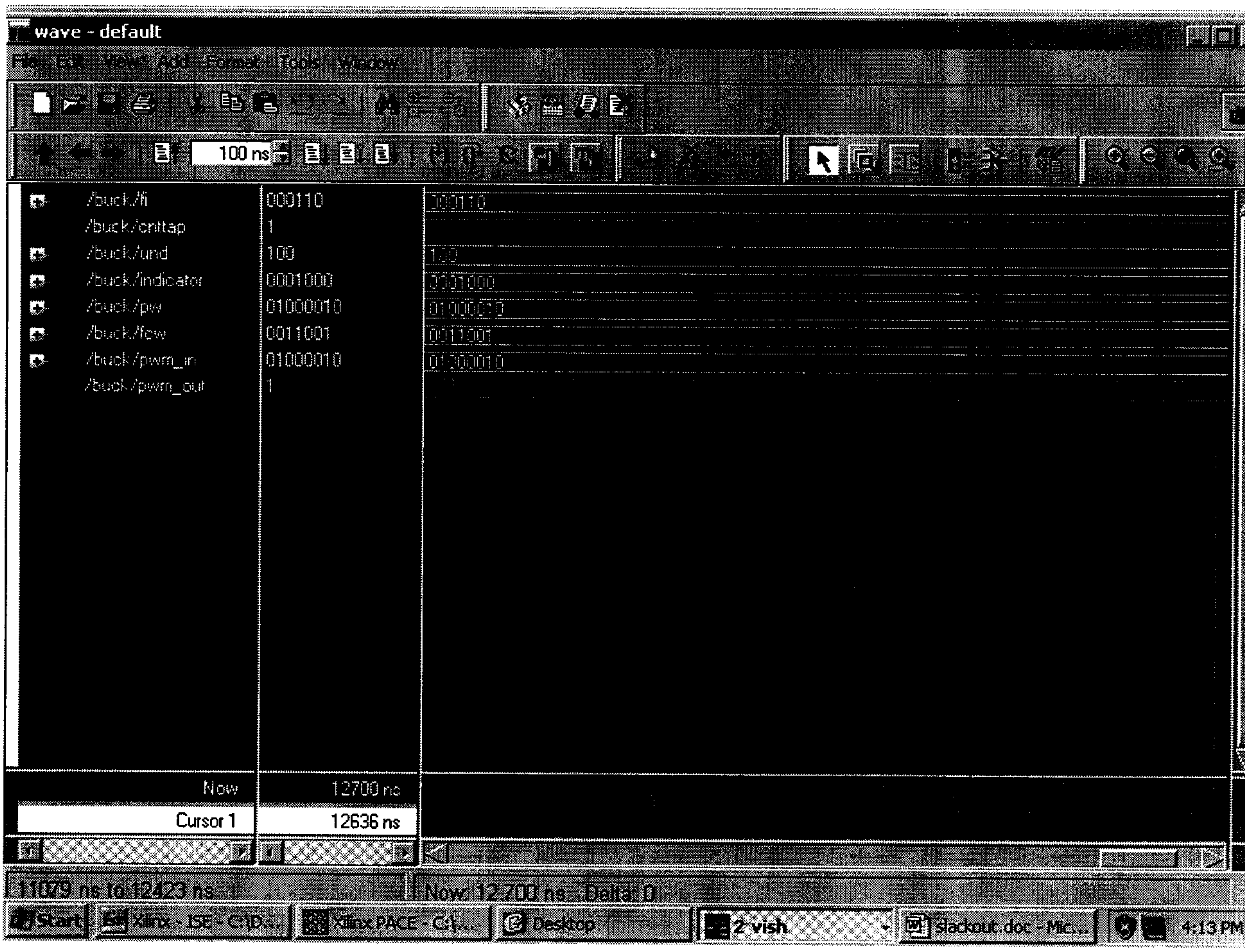
4.2 SIMULATION RESULT OF SLACKTIME DETECTOR:



```
/buck/clk          0
/buck/samp         U
/buck/dclk         1
/buck/sclk         1
/buck/iclk         1
/buck/load         0
/buck/istn         1
/buck/tap_reg12    0
/buck/tap_reg13    0
/buck/tap_reg14    0
/buck/tap_reg15    0
/buck/tap_reg16    0
/buck/tap_reg17    0
/buck/tap_reg18    0
/buck/tap_reg19    0
/buck/tap_reg20    0
/buck/tap_reg21    0
/buck/tap_reg22    0
/buck/tap_reg23    0
/buck/tap_reg24    0
/buck/tap_reg25    0
/buck/tap_reg26    0
            Now    5400 ns
       Cursor 1    5115 ns
```

This figure shows the simulation result of the slack time detector. When the count of sample clock (SCLK) reaches 32, the load goes high (1) and it causes the delay line enabling signal to become low active signal. At this instant the delay lines are tapped into the tap registers (tap_reg12 to 27). The time interval between each delay line is 1ns.

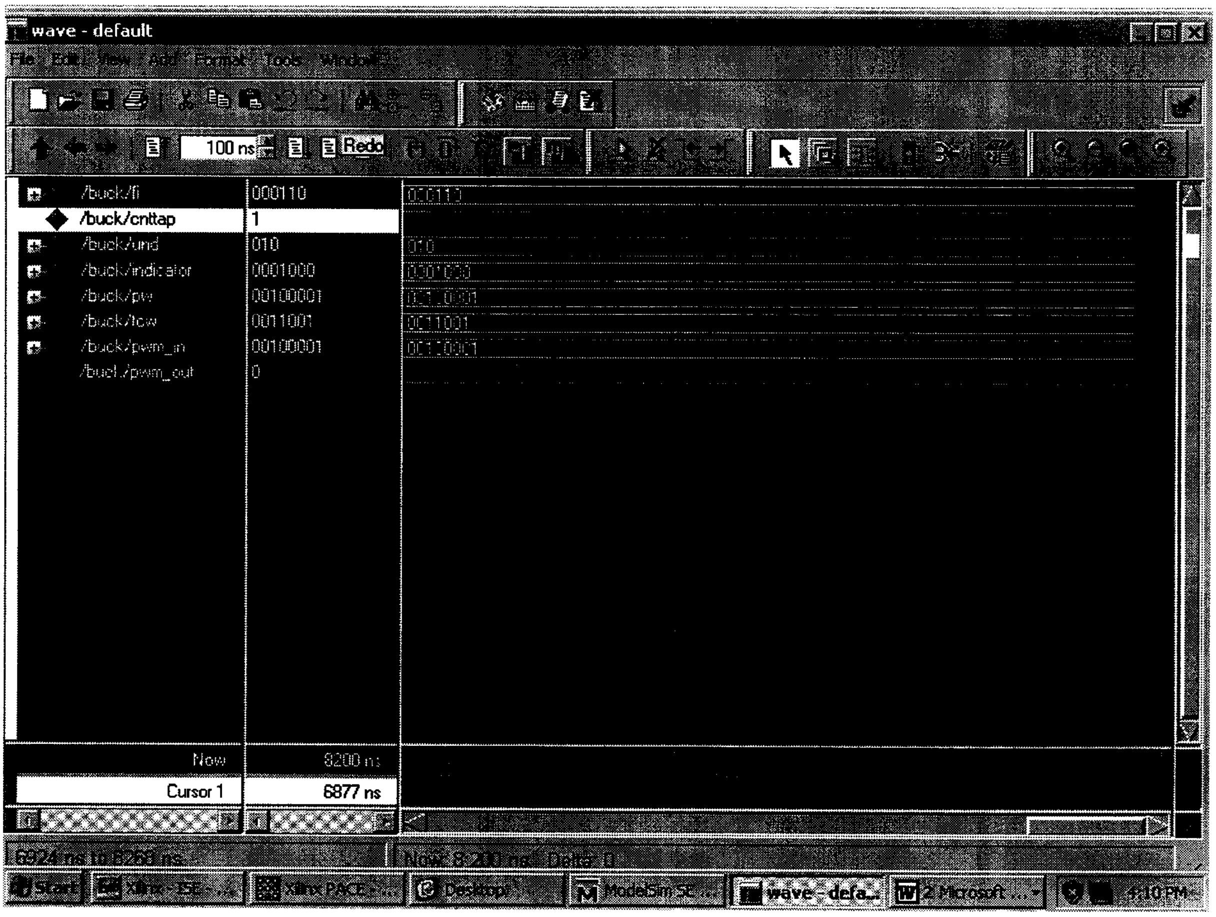
4.3 SIMULATION RESULT OF DPWM MODULATOR FOR (UND=100):



```
/buck/fi          000110
/buck/cnttap      1
/buck/und         100
/buck/indicator   0001000
/buck/pw          01000010
/buck/fcw         0011001
/buck/pwm_in      01000010
/buck/pwm_out     1

Now               12700 ns
Cursor 1          12636 ns

11079 ns to 12423 ns      Now: 12 700 ns  Delta: 0
```

This shows the simulation result of the DPWM modulator. The und value is assigned to 100 and the indicator value is assigned to 0001000. For und=100,pulses of 100ns are generated. These pulses are tapped out by PWM_OUT port.

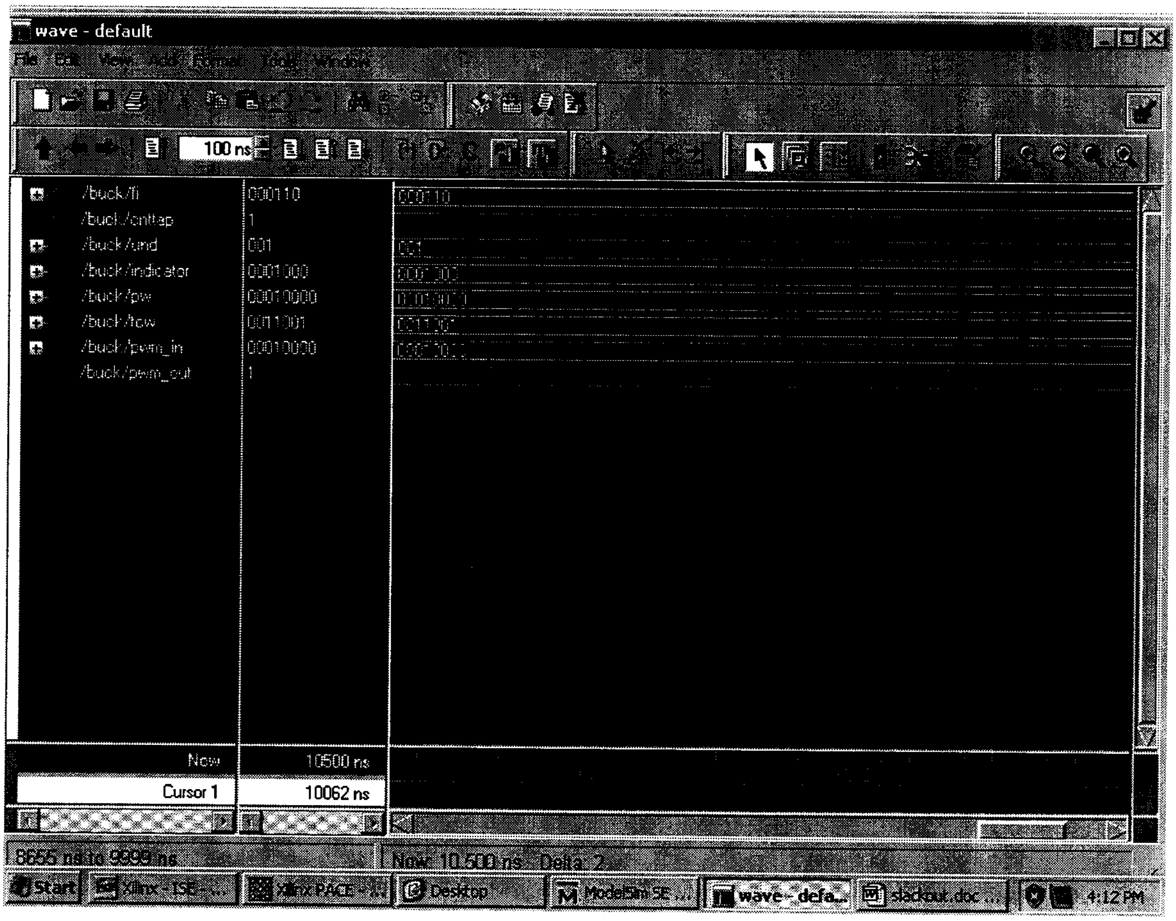
4.4 SIMULATION RESULT OF DPWM MODULATOR FOR (UND=010):



```
/buck/fi          000110
/buck/cnttap      1
/buck/und         010
/buck/indicator   0001000
/buck/pw          00100001
/buck/tow         0011001
/buck/pwm_in      00100001
/buck/pwm_out     0
Now               8200 ns
Cursor 1          6877 ns
```

This shows the simulation result of the DPWM modulator. The und value is assigned to 100 and the indicator value is assigned to 0001000. For und=010,pulses of 100ns are generated. These pulses are tapped out by PWM_OUT port.

4.5 SIMULATION RESULT OF DPWM MODULATOR FOR (UND=001):



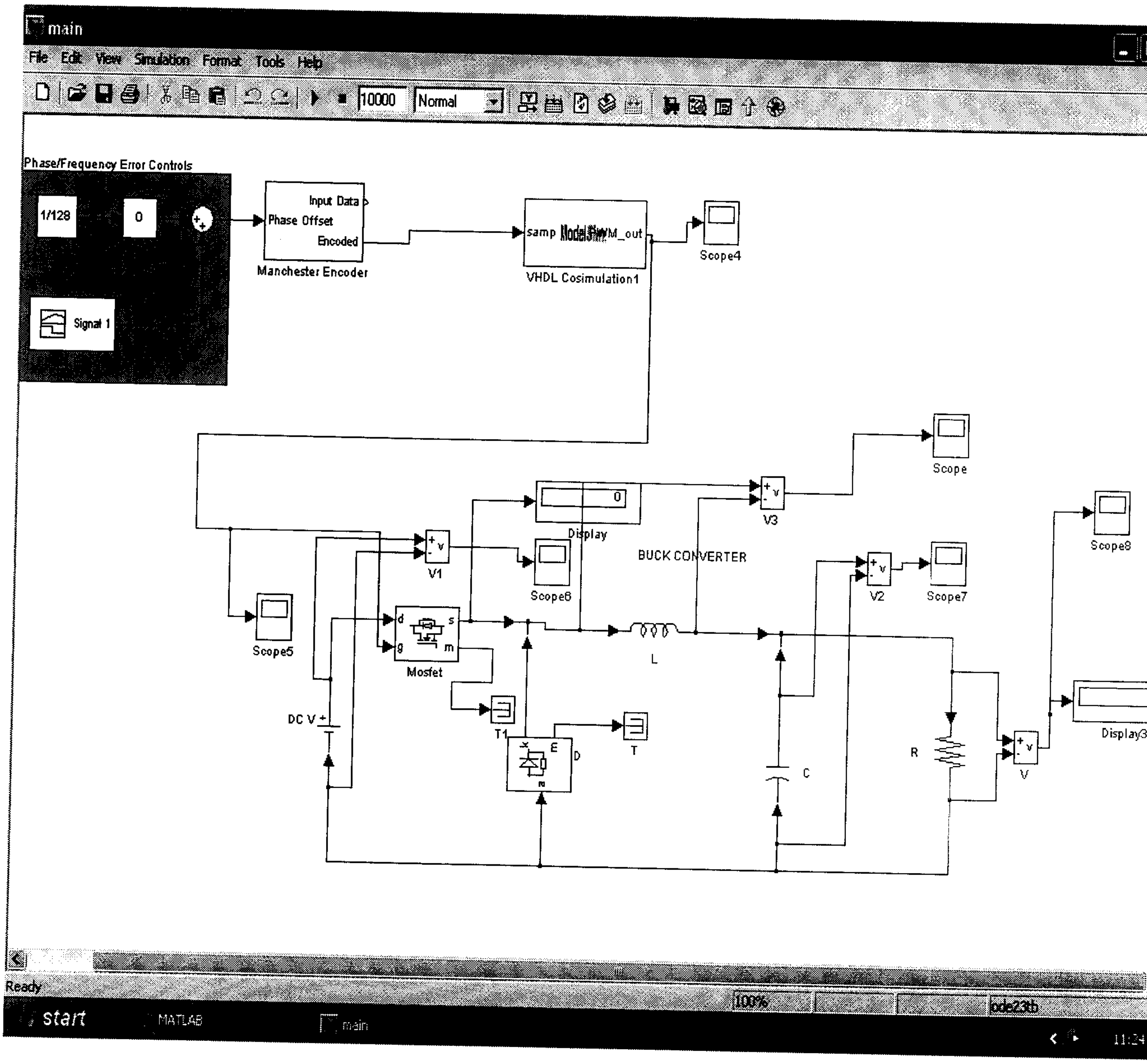
```
wave - default
/buck/fi            000110
/buck/cnttap        1
/buck/und           001
/buck/indicator     0001000
/buck/pw            00010000
/buck/tow           0011001
/buck/pwm_in        00010000
/buck/pwm_out       1

Now                 10500 ns
Cursor 1            10062 ns

8665 ns to 9999 ns    Now: 10,500 ns  Delta: 2
```

This shows the simulation result of the DPWM modulator. The und value is assigned to 100 and the indicator value is assigned to 0001000. For und=010, pulses of 100ns are generated. These pulses are tapped out by PWM_OUT port.

CHAPTER-5

5.MATLAB

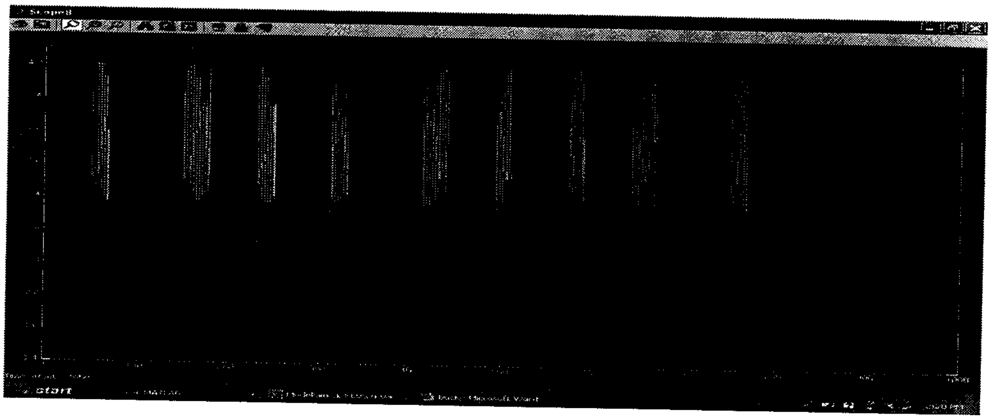
5.1 MATLAB IMPLEMENTATION OF BUCK COVERTER:



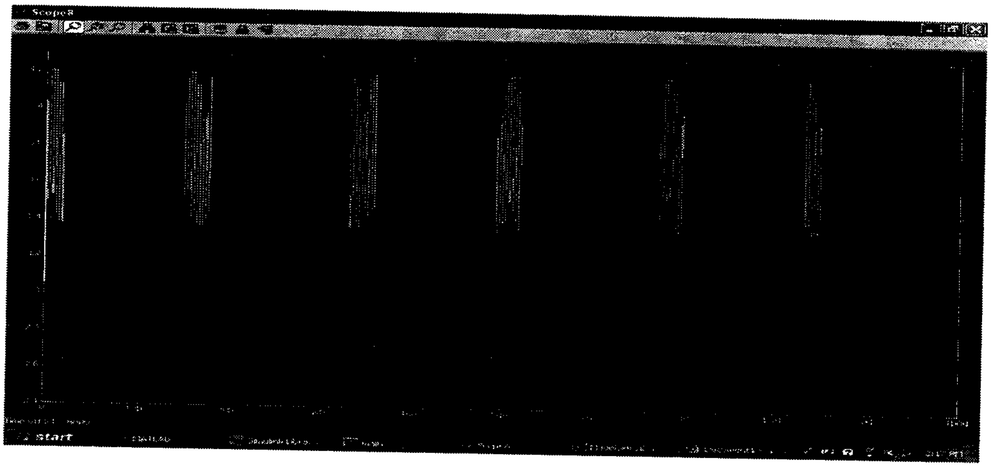
The sample clock is generated by Manchester encoder. When the sample clock is given to the VHDL co simulation block, it outputs the PWM pulses generated by MODELSIM. These pulses are input to MOSFET as gating signal. The voltage bucked is obtained across load resistors.

5.2 SIMULATION RESULTS:

Output voltage for und=100:



Output voltage for und=010:



Output voltage for und=001:

CHAPTER - 6

6. FPGA

6.1 INTRODUCTION:

A Field Programmable Gate Array (FPGA) is a semiconductor device containing electrically programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories. An FPGA is similar to a PLD, but whereas PLDs are generally limited to hundreds of gates, FPGAs support thousands of gates.

A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer (hence the term "field programmable", i.e. programmable in the field) so that the FPGA can perform any logical function needed.

6.2 ARCHITECTURE:

The typical basic architecture consists of an array of configurable logic blocks (CLBs) and routing channels. Multiple I/O pads may fit into the height of one row or the width of one column in the array. Generally, all the routing channels have the same width (number of wires). The basic components of FPGA are

- Configurable Logic Block (CLBs)
- Interconnect
- Input Output Block(IOBs)
- Memory and Complete Clock Management

An application circuit must be mapped into an FPGA with adequate resources. The typical FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop, as shown

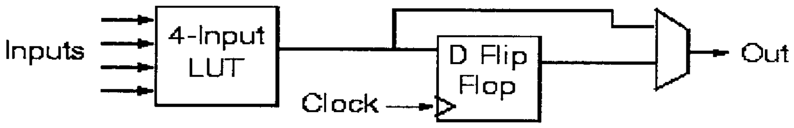


FIG 16: LOGIC BLOCK

There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since clock signals (and often other high-fanout signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed. For this example architecture, the locations of the FPGA logic block pins are shown

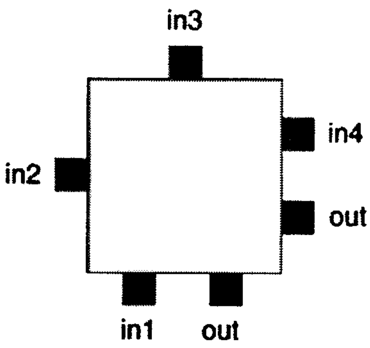


FIG 17: LOGIC BLOCK PIN LOCATIONS

Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block. Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

6.3 FPGA MANUFACTURERS:

Field programmable devices are manufactured by several companies and a few of them are listed

- Altera Corporation
- Xilinx Inc
- Atmel Corporation
- QuickLogic Corporation
- Actel Corporation
- Aeroflex Inc
- Lattice semiconductor corporation
- Leopard Logic Inc

Xilinx first formed in 1984, are the major manufacturers of CPLDs and SRAM based FPGAs. Xilinx leads FPGA industry by its products

- Virtex series
- Spartan series.

The PWM control IC designed is programmed into Spartan II XC3S400 PQ208

6.4 FPGA DESIGN FLOW

The standard design flow for Spartan generation FPGAs include following three major steps

- Design Entry and Synthesis
- Design Implementation
- Design Verification

6.4.1 DESIGN DESCRIPTION

Designer describes design functionality either by using schema editors or by using one of the various Hardware Description Languages (HDLs) like Verilog or VHDL. The control IC is designed in VHDL code. A standard simulator which supports VHDL is used to verify the correctness of the design. Data can be analyzed in a number of ways. Waveform display and tabular display are generally used to trace down the errors in VHDL code. Functional simulation is done by creating testbench. Testbench is an environment, where a design is checked by applying stimuli and monitoring responses.

6.4.2 SYNTHESIS

Next step is to synthesize the design. The goal of VHDL synthesis is to create a design that implements the required functionality and matches the designer's constraints an area, speed or power. The synthesis tool reads the VHDL design and reports syntax errors and synthesis errors. If there are no syntax errors, the designer can synthesize the design and map to target technology. If any changes are to be made in the VHDL description, then the description needs to be simulated again and the output is validated for correctness. The synthesizer produces an output netlist in the target technology and a number of report files. From the netlist, it can be determined that the design is reasonable or not. The reports such as timing report and device utilization summary helps to determine the quality of the design.

6.4.3 DESIGN IMPLEMENTATION:

Implementation includes Partition, Place and route. Place and Route translates the logic design into physical design, maps the components used in the design into specific elements, places them and routes the interconnection between them. Place and Route also helps to do timing analysis. After all cells are placed and routed, the output of

the place and route tool consists of data files that can be used to implement the chip. The output of design implementation phase is bit-stream file. These files describe all the connections needed to make the FPGA macro cells implement the functionality required.

6.4.4 DESIGN VERIFICATION:

Bit stream file is fed to a simulator which simulates the design functionality and reports errors in desired behavior of the design. Timing tools are used to determine maximum clock frequency of the design. Now the design is loading onto the target FPGA device and testing is done in real environment.

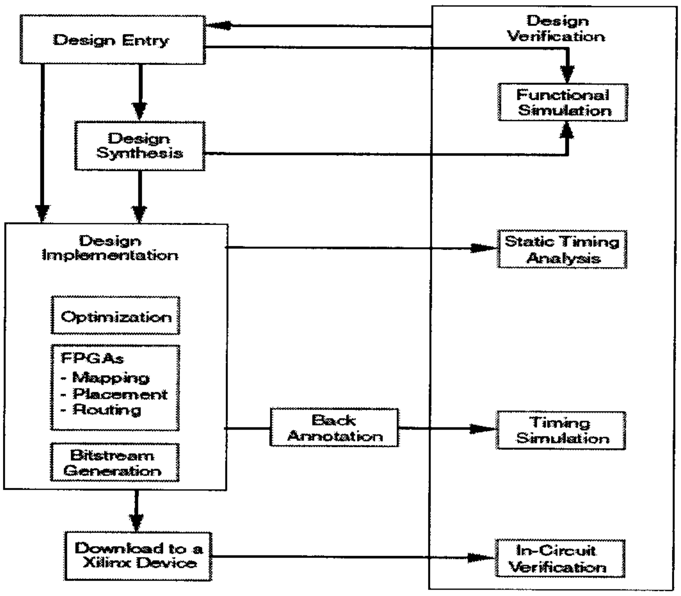


FIG 18:DESIGN FLOW

6.5 ARCHITECTURE OF XC3S400:

6.5.1 BLOCK DIAGRAM:

The Spartan-III family of FPGAs is implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs), interconnected by a powerful hierarchy of versatile routing resources. The architecture also provides advanced functions such as Block RAM and clock control blocks as shown in figure 19. Features of XC2S200 model are

- System gates -200,000

- Logic cells – 5292
- CLBs – 1176
- Block RAM (bits) – 56K
- I/O – 208

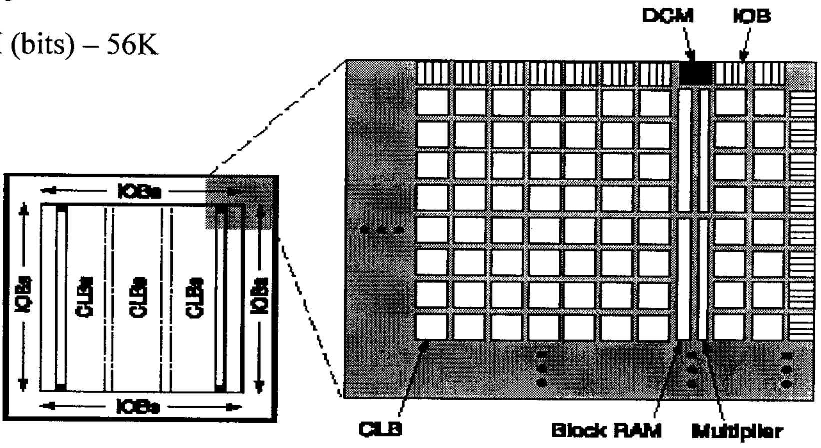


FIG.19: BASIC ARCHITECTURE OF XC3S400

6.5.2 Input/Output Block

The Spartan-II IOB features inputs and outputs that support 16 I/O signaling standards, including LVCMOS, HSTL, SSTL, and GTL. These high-speed inputs and outputs are capable of supporting various state-of-the-art memory and bus interfaces. The three IOB registers function either as edge-triggered D-type flip-flops or as level sensitive latches.

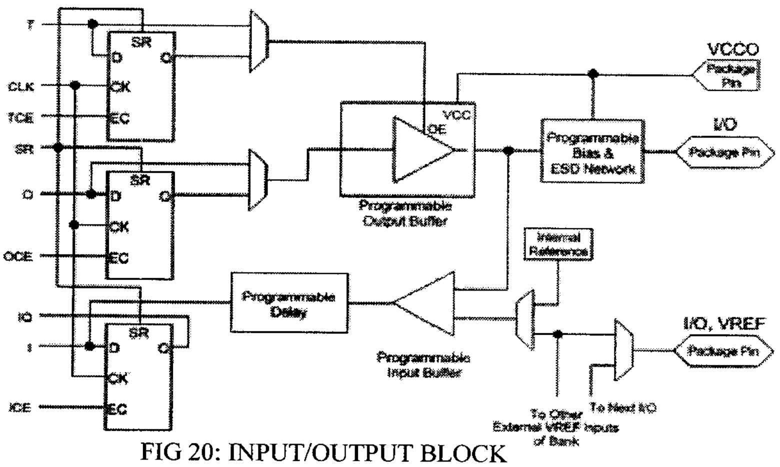


FIG 20: INPUT/OUTPUT BLOCK

6.5.3 Logic Cells

The basic building block of the Spartan-II CLB is the logic cell (LC). An LC includes a four-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each Spartan-III CLB contains four LCs, organized in two similar slices. In addition to the four basic LCs, the Spartan-III CLB contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 LCs.

Spartan-III function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. The Spartan-III LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as digital signal processing. The storage elements in the Spartan-III slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches.

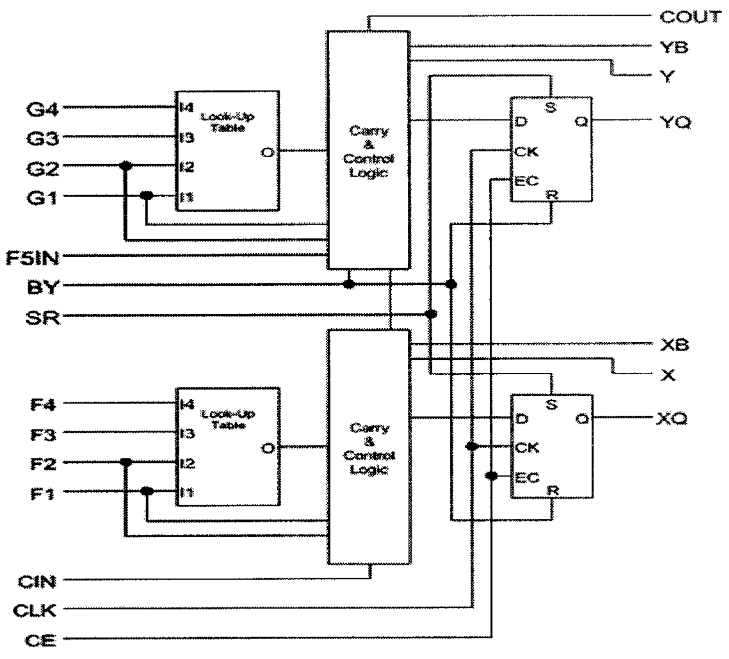


FIG 21: LOGIC CELL OF XC3S400

6.5.4 Delay-Locked Loop

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synchronism with clock edges arriving at the input.

6.5.5 Block RAM

Spartan-III FPGAs incorporate several large Block SelectRAM+ memories of 4K bits. These complement the distributed SelectRAM+ resources that provide shallow RAM structures implemented in CLBs. Each block can be configured at ratios between 4Kx1 and 256x16.

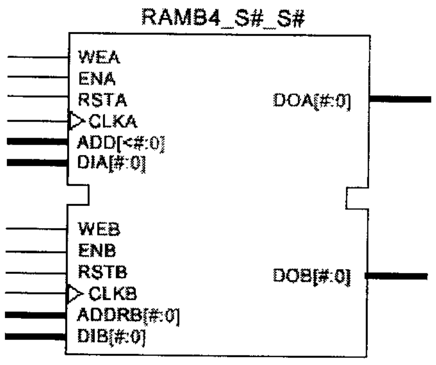


FIG 22: BLOCK RAM

6.5.6 Features of XC3S400

The XC3S400 features up to 200,000 system gates, 14 total RAM blocks, and up to 284 I/Os in three low-cost, high-volume packages: PQ208, FG256, and FG456.

The XC3S400 includes

- Distributed and Block Memory
- Four Digital Delay Locked Loops per Device
- Versatile I/O Interface Technology
- Full PCI Compliance

6.6 APPLICATIONS OF FPGA:

Applications of FPGAs include DSP, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation and a growing range of other areas. FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SOC). Due to their programmable nature, FPGAs are an ideal fit for many different markets such as

- Aerospace & Defense
- Automotive
- Broadcast
- Industrial/Scientific/Medical
- Storage & Server
- Wireless and Wired Communications

CHAPTER-7

7. HARDWARE IMPLEMENTATION

7.1 FPGA XC2S200 PQ208:

The photo shows the programmed FPGA XC2S200 PQ208 device

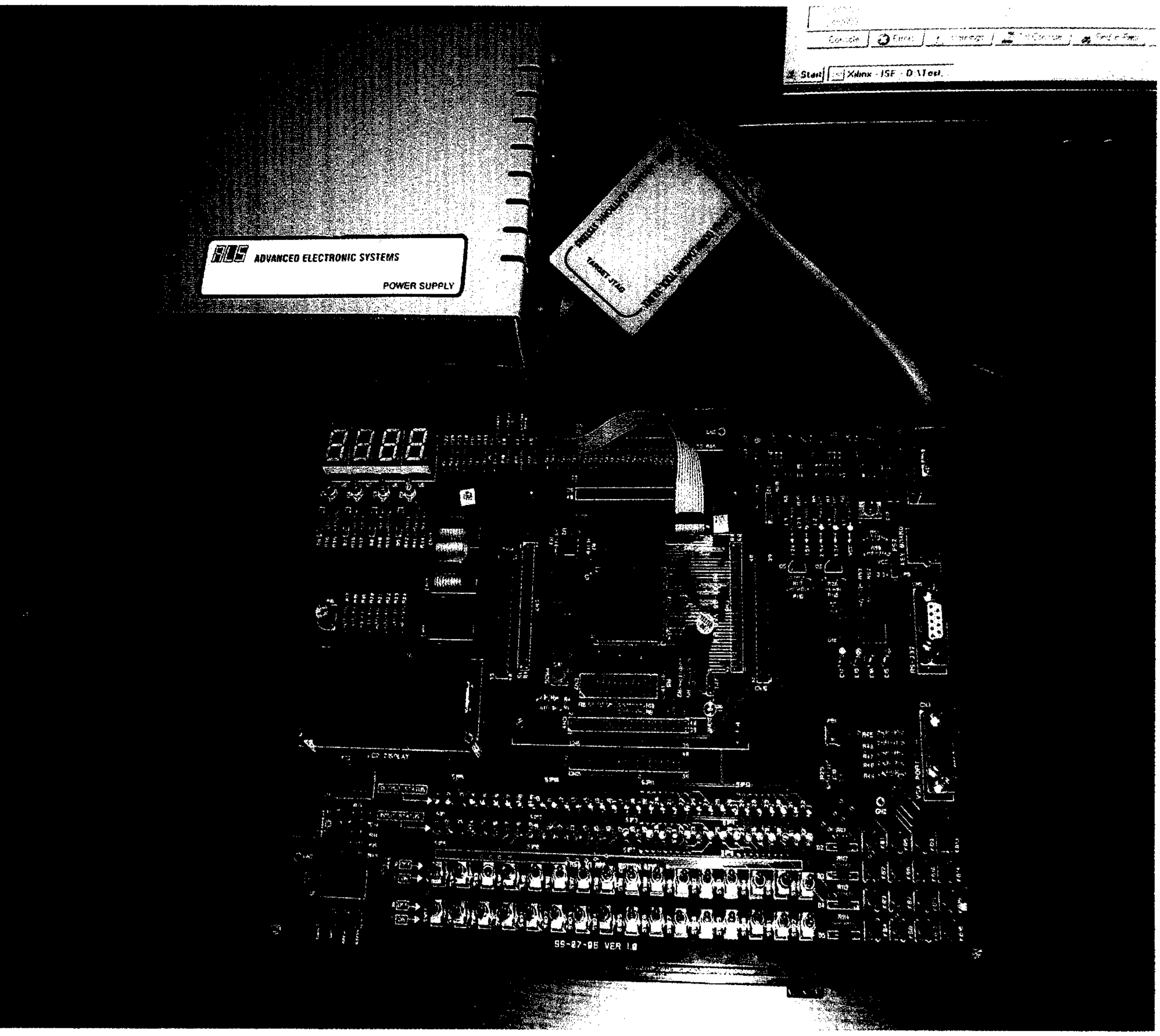
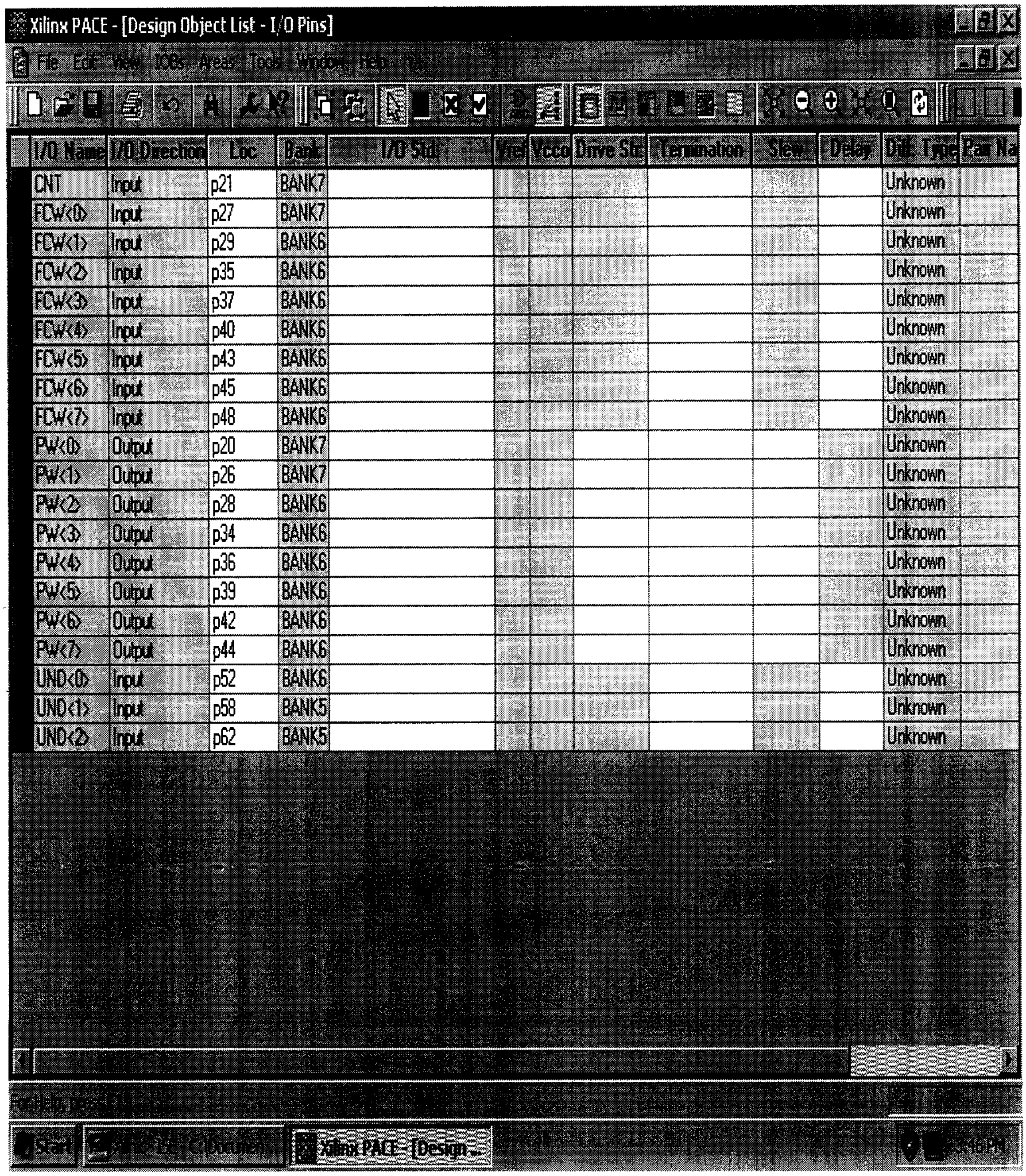


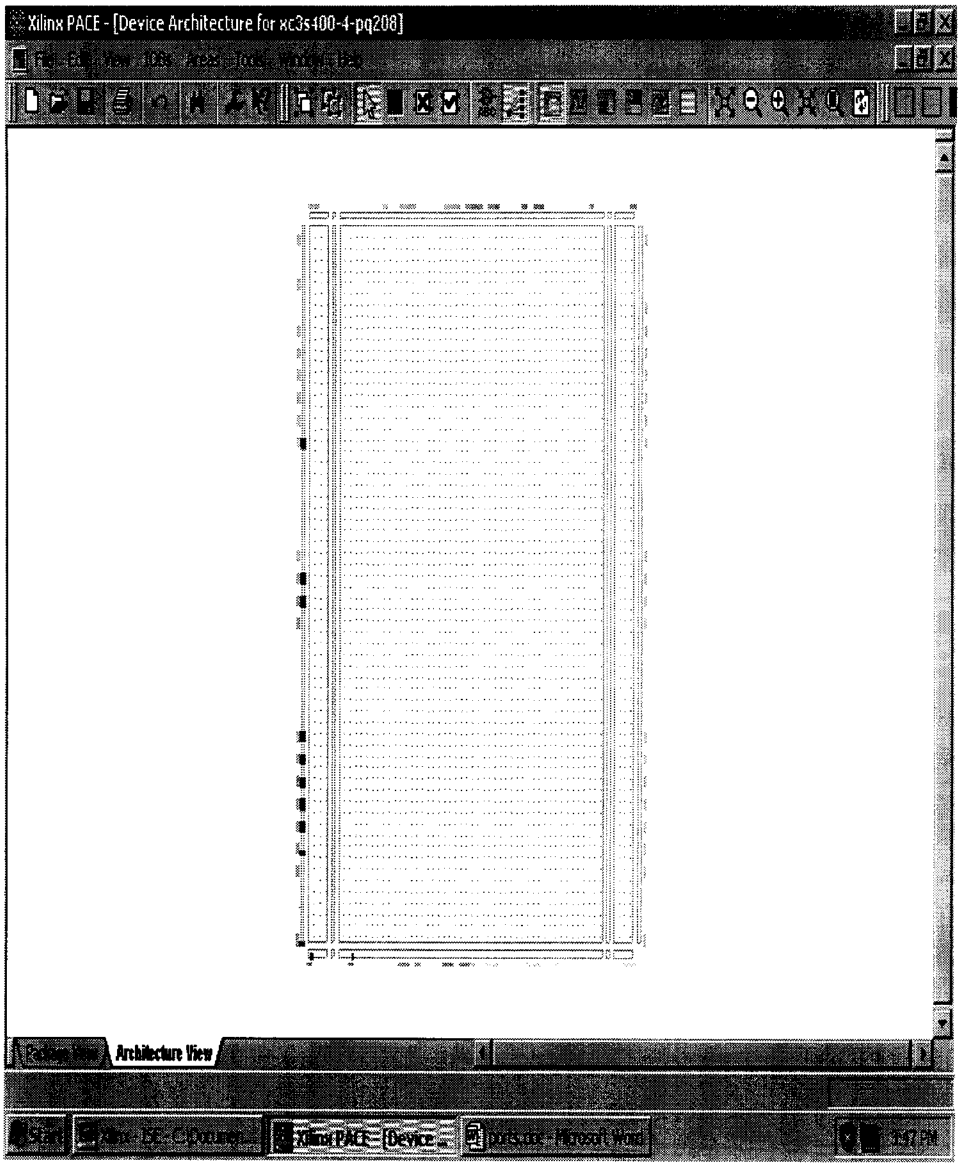
FIG 23: FPGA XC2S300 PQ208 DEVICE

7.2 DESIGN OBJECT LIST-I/O PINS:


Xilinx PACE - [Design Object List - I/O Pins]
File Edit View IOBs Areas Tools Window Help
For Help, press F1
Start
Xilinx PACE - [Design...

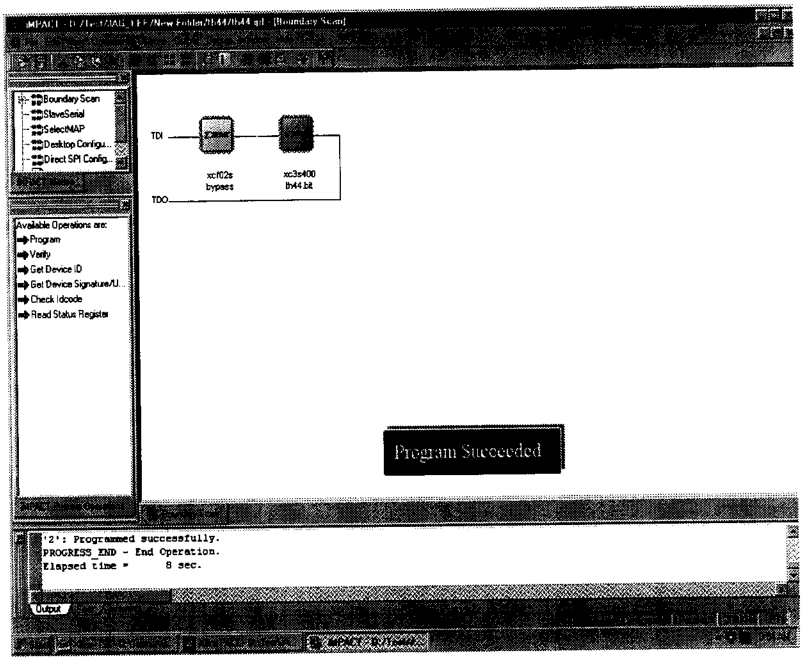
I/O Name	I/O Direction	Loc	Bank	I/O Std	Vref	Vcco	Drive Str	Termination	Slew	Delay	Diff. Type	Pair Na
CNT	Input	p21	BANK7								Unknown	
FCW<0>	Input	p27	BANK7								Unknown	
FCW<1>	Input	p29	BANK6								Unknown	
FCW<2>	Input	p35	BANK6								Unknown	
FCW<3>	Input	p37	BANK6								Unknown	
FCW<4>	Input	p40	BANK6								Unknown	
FCW<5>	Input	p43	BANK6								Unknown	
FCW<6>	Input	p45	BANK6								Unknown	
FCW<7>	Input	p48	BANK6								Unknown	
PW<0>	Output	p20	BANK7								Unknown	
PW<1>	Output	p26	BANK7								Unknown	
PW<2>	Output	p28	BANK6								Unknown	
PW<3>	Output	p34	BANK6								Unknown	
PW<4>	Output	p36	BANK6								Unknown	
PW<5>	Output	p39	BANK6								Unknown	
PW<6>	Output	p42	BANK6								Unknown	
PW<7>	Output	p44	BANK6								Unknown	
UND<0>	Input	p52	BANK6								Unknown	
UND<1>	Input	p58	BANK5								Unknown	
UND<2>	Input	p62	BANK5								Unknown	

7.3 ARCHITECTURE VIEWS OF XC3S 400 PQ208:


Xilinx PACE - [Device Architecture for xc3s400-4-pq208]
File Edit View IOBs Areas Tools Window Help
Package View
Architecture View
Start
Xilinx PACE - [Device ...
ports.doc - Microsoft Word

7.4 FINAL RESULTS:

RTL Top Level Output File Name	: ADSH.ngr
Top Level Output File Name	: ADSH
Output Format	: NGC
Optimization Goal	: Speed



```
'2': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =      8 sec.
```

Design Statistics

# IOs	: 20

Cell Usage:

#	BELS	: 29
#	LUT3	: 9
#	LUT4	: 18
#	MUXF5	: 2
#	IO Buffers	: 19
#	IBUF	: 11
#	OBUF	: 8

Device utilization summary:

Selected Device : 3s400pq208-4

```
Number of Slices            :      15 out of   3584    0%
Number of 4 input LUTs      :      27 out of   7168    0%
Number of IOs               :      20
Number of bonded IOBs       :      19 out of    141   13%
```

Clock Information:

No clock signals found in this design

Asynchronous Control Signals Information:

No asynchronous control signals found in this design

Timing Summary:

Speed Grade: -4

```
   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 16.130ns
```

CHAPTER-8

8. CODING OF THE PROGRAM

```
-------------------------------------------------------------------
----------------------POWER DELIVERY SYSTEM------------------------
-------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity buck is
port(clk: inout bit; dclk,sclk,iclk,load,rstn: inout bit;
    tap_reg12,tap_reg13,tap_reg14,tap_reg15:inout bit;
    tap_reg16,tap_reg17,tap_reg18,tap_reg19:inout bit;
    tap_reg20,tap_reg21,tap_reg22,tap_reg23:inout bit;
    tap_reg24,tap_reg25,tap_reg26,tap_reg27:inout bit;
    dataout1,dataout2,dataout3,dataout4:inout bit;
    dataout5,dataout6,dataout7,dataout8:inout bit;
    dataout9,dataout10,dataout11,dataout12:inout bit;
    dataout13,dataout14,dataout15,dataout16:inout bit;
    dataout17,dataout18,dataout19,dataout20:inout bit;
    dataout21,dataout22,dataout23,dataout24:inout bit;
    dataout25,dataout26,dataout27:inout bit;
    FI:in std_logic_vector(0 to 5);  -- force the input
    cnttap:in std_logic;             -- force the input
    und:in std_logic_vector(2 downto 0 ) ;
    indicator:in std_logic_vector(6 downto 0);
     PW: inout std_logic_vector(7 downto 0);
    FCW:inout std_logic_vector(0 to 6);
    PWM_in : inout std_logic_vector (7 downto 0) ;
    PWM_out : out std_logic);
end entity;
```

```
architecture only of buck is
function delayline (input1,input2:bit)
                return bit is
                variable res:bit;
                variable a:bit;
                variable b:bit;
                variable c:bit;
                variable d:bit;
                variable e:bit;
 begin
 a:=((not (input1)) nor input2);
 b:=((not (a)) nor input2);
 c:=((not (b)) nor input2);
 d:=((not (c)) nor input2);
 e:=((not (d)) nor input2);
 res:=e;
 return res;
 end function delayline;
  function left_shift (q:std_logic_vector(0 to 5))
                  return std_logic_vector is
    variable r,res:std_logic_vector(0 to 5);
  begin
      r(0):=q(1);
      r(1):=q(2);
      r(2):=q(3);
      r(3):=q(4);
      r(4):=q(5);
      r(5):='0';
  res:=r;
  return res;
  end left_shift;
```

```
    function right_shift (q:std_logic_vector(0 to 5))
                    return std_logic_vector is
    variable r,res:std_logic_vector(0 to 5);
  begin
      r(0):='0';
      r(1):=q(1);
      r(2):=q(2);
      r(3):=q(3);
      r(4):=q(4);
      r(5):=q(5);

   res:=r;
 return res;
end right_shift;

function left_shift1 (q:std_logic_vector(0 to 7))
                    return std_logic_vector is
    variable r,res:std_logic_vector(0 to 7);
  begin


      r(0):=q(1);
      r(1):=q(2);
      r(2):=q(3);
      r(3):=q(4);
      r(4):=q(5);
      r(5):=q(6);
      r(6):=q(7);
      r(7):='0';
   res:=r;
  return res;
  end left_shift1;
  function right_shift1 (q:std_logic_vector(0 to 7))
                 return std_logic_vector is
     variable r,res:std_logic_vector(0 to 7);
  begin
       r(0):='0';
       r(1):=q(0);
       r(2):=q(1);
       r(3):=q(2);
       r(4):=q(3);
       r(5):=q(4);
       r(6):=q(5);
       r(7):=q(6);
   res:=r;
  return res;
  end right_shift1;

constant t1:time:=12 ns;
    --80MHZ  (CLK)
signal count:natural:=0;
constant t5:time:=1 ns;
--signal detect_volt:real;
type indata is array(12 to 27) of bit;
signal Tap_reg_data,IX_reg_data:indata;
--signal voltage_error:real;
signal ECW:std_logic_vector(0 to 5);
signal binary_detect_volt,binary_voltage_error:std_logic_vector(1 to 6);
signal ff:std_logic:='0';
signal sub1,sub3,sub2:std_logic_vector(0 to 5);
signal counter_out:std_logic_vector(0 to 5):="000111";
signal R_FI:std_logic_vector(0 to 5);
signal a:std_logic_vector(0 to 5);
signal mux1:std_logic_vector(7 downto 0);
signal add1,reg1,add2:std_logic_vector(0 to 7);
signal  PWM_Accumulator : std_logic_vector(8 downto 0):="000000000";
signal IREF:std_logic_vector(7 downto 0):="00001000";     --8 volt
begin
 --CLOCK GENERATOR MODULE
process(dclk)
    begin
                dclk<='1' after 6 ns;
               -- wait for (t1/2) ;
                dclk<='0' after 6 ns;
              --  wait for (t1/2) ;
    end process;

      process(sclk)
    begin
               sclk<='1' after  3 ns ;
              -- wait for (t1*4);
               sclk<='0' after  3 ns ;
              -- wait for (t1*4) ;
               end process ;



 process(iclk)
               begin
               iclk<='1'  after 48 ns;
              -- wait for (t1*4);
               iclk<='0' after 48 ns;
              -- wait for (t1*4) ;
               end process ;
               process (clk)
```

```
                begin
                clk<='1' after  12 ns;
               -- wait for t1;
                clk<='0'after   12 ns;
              --  wait for t1;
                end process ;
 process(SCLK)
begin
if SCLK='1' then
 count<=count+1;
 LOAD<='0';
 RSTN<='1';
if count=31 then
 LOAD<='1';
 RSTN<='0';
 count<=0;
end if;
end if;
end process;

--SLACKTIME DETECTOR MODULE
p6:process(iclk,rstn,dataout1,dataout2,dataout3,dataout4,dataout5,dataout6,dataout7,data
out8 , dataout9,dataout10,dataout11,dataout12,dataout13,dataout14,dataout15,dataout16,
      dataout17,dataout18,dataout19,dataout20,dataout21,dataout22,dataout23,dataout24,
        dataout25,dataout26,dataout27)
begin
dataout1<= delayline(iclk,rstn) ;
dataout2<= delayline (dataout1,rstn)after t5 ;
dataout3<=delayline  (dataout2,rstn) after t5 ;
dataout4<=delayline  (dataout3,rstn) after t5;
dataout5<=delayline  (dataout4,rstn) after t5;
```

```
dataout6<=delayline ( dataout5,rstn) after t5;
dataout7<=delayline  (dataout6,rstn) after t5;
dataout8<=delayline  (dataout7,rstn) after t5;
dataout9<=delayline  (dataout8,rstn) after t5;
dataout10<=delayline (dataout9,rstn)after t5;
dataout11<=delayline (dataout10,rstn) after t5;
dataout12<=delayline (dataout11,rstn) after t5;
dataout13<=delayline (dataout12,rstn) after t5;
dataout14<=delayline (dataout13,rstn)after t5;
dataout15<=delayline (dataout14,rstn) after t5;
dataout16<=delayline (dataout15,rstn) after t5;
dataout17<=delayline (dataout16,rstn) after t5;
dataout18<=delayline (dataout17,rstn)after t5;
dataout19<=delayline (dataout18,rstn) after t5;
dataout20<=delayline (dataout19,rstn) after t5;
dataout21<=delayline (dataout20,rstn) after t5;
dataout22<=delayline( dataout21,rstn) after t5;
dataout23<=delayline (dataout22,rstn) after t5;
dataout24<=delayline (dataout23,rstn) after t5;
dataout25<=delayline (dataout24,rstn) after t5;
dataout26<=delayline (dataout25,rstn) after t5;
dataout27<=delayline( dataout26,rstn) after t5;
  end process p6;
p7:process(sclk,dataout12,dataout13,dataout14,dataout15,dataout16,
dataout17,dataout18,dataout19,dataout20,dataout21,dataout22,dataout23,dataout24,
        dataout25,dataout26,dataout27)
begin
if SCLK='1' then
tap_reg12<=dataout12 ;
tap_reg13<=dataout13 ;
tap_reg14<=dataout14 ;
```

```
tap_reg15<=dataout15 ;
tap_reg16<=dataout16 ;
tap_reg17<=dataout17 ;
tap_reg18<=dataout18 ;
tap_reg19<=dataout19 ;
tap_reg20<=dataout20 ;
tap_reg21<=dataout21 ;
tap_reg22<=dataout22 ;
tap_reg23<=dataout23 ;
tap_reg24<=dataout24 ;
tap_reg25<=dataout25 ;
tap_reg26<=dataout26 ;
tap_reg27<=dataout27 ;
Tap_reg_data<=(tap_reg12,tap_reg13,tap_reg14,tap_reg15,tap_reg16,tap_reg17,tap_reg1
8,tap_reg19,
tap_reg20,tap_reg21,tap_reg22,tap_reg23,tap_reg24,tap_reg25,tap_reg26,tap_reg27);

end if;
end process;
--ERROR COMPENSATOR
p8:process(tap_reg12,tap_reg13,tap_reg14,tap_reg15,
        tap_reg16,tap_reg17,tap_reg18,tap_reg19,
        tap_reg20,tap_reg21,tap_reg22,tap_reg23,
        tap_reg24,tap_reg25,tap_reg26,tap_reg27,Tap_reg_data,IX_reg_data)
variable IX12,IX13,IX14,IX15,IX16,IX17,IX18,IX19,IX20:bit;
variable IX21,IX22,IX23,IX24,IX25,IX26,IX27:bit;
variable binary_reference_voltage:std_logic_vector(1 to 6):="011000";
--variable reference_voltage:real:=1.20;
begin
IX12:=(tap_reg12 nand (not(tap_reg13)));
IX13:=(tap_reg13 nand (not(tap_reg14)));
```

```
IX14:=(tap_reg14 nand (not(tap_reg15)));
IX15:=(tap_reg15 nand (not(tap_reg16)));
IX16:=(tap_reg16 nand (not(tap_reg17)));
IX17:=(tap_reg17 nand (not(tap_reg18)));
IX18:=(tap_reg18 and (not(tap_reg19)));
IX19:=(tap_reg19 nand (not(tap_reg20)));
IX20:=(tap_reg20 nand (not(tap_reg21)));
IX21:=(tap_reg21 nand (not(tap_reg22)));
IX22:=(tap_reg22 nand (not(tap_reg23)));
IX23:=(tap_reg23 nand (not(tap_reg24)));
IX24:=(tap_reg24 nand (not(tap_reg25)));
IX25:=(tap_reg25 nand (not(tap_reg26)));
IX26:=(tap_reg26 nand (not(tap_reg27)));
IX27:=(tap_reg27);
IX_reg_data<=(IX12,IX13,IX14,IX15,IX16,IX17,IX18,IX19,IX20,IX21,IX22,IX23,IX2
4,IX25,IX26,IX27);
if IX_reg_data= "0111111111111110" then
binary_detect_volt<="011011";
--detect_volt<=1.35;
elsif IX_reg_data="1011111111111110" then
 binary_detect_volt<="011010";
--detect_volt<=1.30;
elsif IX_reg_data="1101111111111110" then
binary_detect_volt<="011001";
--detect_volt<=1.25;
elsif IX_reg_data="1110111111111110" then
binary_detect_volt<="011000";
--detect_volt<=1.20;
elsif IX_reg_data="1111011111111110" then
binary_detect_volt<="010111";
--detect_volt<=1.15;
```

```
elsif IX_reg_data="1111101111111110" then
binary_detect_volt<="010110";
--detect_volt<=1.10;
elsif IX_reg_data="1111110111111110" then
binary_detect_volt<="010101";
--detect_volt<=1.05;
elsif IX_reg_data="1111111011111110" then
binary_detect_volt<="010100";
 --detect_volt<=1.00;
elsif IX_reg_data="1111111101111110" then
binary_detect_volt<="010011";
--detect_volt<=0.95;
elsif IX_reg_data="1111111110111110" then
binary_detect_volt<="010010";
--detect_volt<=0.90;
elsif IX_reg_data="1111111111011110" then
binary_detect_volt<="010001";
--detect_volt<=0.85;
elsif IX_reg_data="1111111111101110" then
binary_detect_volt<="010000";
--detect_volt<=0.80;
elsif IX_reg_data="1111111111110110" then
binary_detect_volt<="001111";
--detect_volt<=0.75;
elsif IX_reg_data="1111111111111010" then
binary_detect_volt<="001110";
--detect_volt<=0.70;
elsif IX_reg_data="1111111111111100" then
binary_detect_volt<="001101";
--detect_volt<=0.65;
elsif IX_reg_data="1111111111111111" then
```

```
binary_detect_volt<="000000";
--detect_volt<=0.60;
end if;
--voltage_error<=((detect_volt-reference_voltage));
binary_voltage_error<=(binary_detect_volt-binary_reference_voltage);
ECW<=(binary_reference_voltage-binary_voltage_error);
  end process p8;
 --FREQUENCY COMPENSATOR
 process(ECW,FI,load,cnttap,counter_out)
variable RFI:std_logic_vector(0 to 5):="000100";
 variable cmp :std_logic_vector(0 to 5);
begin
sub1<=(FI-RFI) ;
cmp:=(counter_out-sub1);
if load='1' then
if sub1=counter_out then
R_FI<=counter_out;
elsif sub1>counter_out then
counter_out<=(counter_out+"000001");
R_FI<=counter_out;
elsif sub1<counter_out then
counter_out<=(counter_out-"000001");
R_FI<=counter_out;
end if;
end if;
end process;
process(R_FI,load,cnttap,sub2,sub3)
variable RFI:std_logic_vector(0 to 5):="000100";
begin
sub3<=(RFI-R_FI) ;
a<= left_shift(R_FI) ;
```

```
sub2<=(ECW-a) ;
FCW(1 to 6)<=(sub2+sub3);
 case cnttap is
when '0' =>
ff<=not ff;
when '1' =>
ff<='0';
when others=>
end case;
   if cnttap ='1' then
      if load='1' then
FCW(0)<=ff;
else
end if;
end if;
end process;
--PVT COMPENSATOR
process(clk,FCW,indicator,load,und,IREF,mux1,add2,reg1,PWM_in,PW)
begin
 --- if rising_edge(clk) then
case indicator is
when "1000000" =>
mux1<="00000011";
when  "0100000"=>
mux1<="00000010";
when  "0010000"=>
mux1<="00000001";
when  "0001000"=>
mux1<="00000000";
when  "0000100"=>
mux1<="11111111";
```

```
 when  "0000010"=>
mux1<="11111110";
when  "0000001"=>
mux1<="11111101";
when others=>
end case;
add1<=(IREF+mux1);
if load='1' then
 reg1<=add1;
 elsif load='0' then
 reg1<=IREF;
end if;
add2<=(FCW+reg1);
case und is
when "100" =>
PW<=left_shift1(add2);
when "010" =>
PW<=(add2);
when "001" =>
PW<=right_shift1(add2);
when others=>
PW<="00000000";
end case;
--PWM Modulator module
if clk'event and clk='1' then
  PWM_in<=PW;
 if PWM_in/="XXXXXXXX" then
 PWM_Accumulator  <=  ("0" & PWM_Accumulator(7 downto 0)) + ("0" & PWM_in);
 else
 PWM_Accumulator  <="000000000";
   end if;
 end if;
end process;
PWM_out <= PWM_Accumulator(8);
end architecture;
```

CHAPTER-9

9. CONCLUSION:

In this project, a power delivery system has been developed to provide a constant minimum supply voltage with the maximum peak ripple of 5 mV, and guarantees less propagation delay than critical path delay over changes in PVT, load, and frequency. Therefore, the fully digital technique holds promise as a controller for AVS regulation in digital applications that present a hostile environment for noise-sensitive analog circuits. Moreover, it contributes to the yield improvement since the propagation delay variations due to the variations of intrinsic parameter and operating condition are compensated by dynamically adjusting the supply voltage.

APPENDIX I

XILINX®

XILINX Spartan III – XC3S400

Features

Xilinx XC3S400-FG456 or XC3S1500-FG456 Spartan-3 FPGA

Xilinx platform FLASH configuration PROM

2 Oscillators (66 MHz installed & socket for use frequency selection)

Parallel cable III equivalent JTAG configuration port

2 AvBus expansion connectors

1, 50-pin header for easy I/O access (includes 4 LVDS pairs)

Universal 32-bit PCI edge connector*

10/100 Ethernet port*

DB15 & video DAC

RS-232 console

PS2 keyboard and mouse ports

Analog I/O**

1M SRAM

256kb serial EEPROM

4-position DIP switch

2 push-buttons

8 discrete LEDs

Dual-digit 7-segment LED display

Spartan II FPGA family members and description about RAM, I/O, Gates.

Table 1: **Spartan-II FPGA Family Members**

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O(1)	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	92	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	176	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

REFERENCES:

[1] T. D. Burd and R. W. Brodersen, "Design issues for dynamic voltage scaling," in *Proc. ISLPED Conf.*, 2000, pp. 9–14.

[2] K. Suzuki *et al.*, "Variable supply-voltage scheme for low-power high-speed CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 33, no. 3, pp. 454–462, Mar. 1998.

[3] G.-Yawed and M. Horowitz, "A fully digital, energy-efficient, adaptive Power-supply regulator," *IEEE J. Solid-State Circuits*, vol. 34, no. 4, pp. 520–528, Apr. 1999.

[4] J. Kim and M. Horowitz, "An efficient digital sliding controller for Adaptive power supply regulation," in *Proc. Very Large Scale Integetor. (VLSI) Circuits Dig. Tech. Papers Conf.*, 2001, pp. 133–136.

[5] D. W. Kang, "Low-power digital adaptive voltage controller design based on hybrid control and reverse phase mode," Ph.D. dissertation, Dept. Elect. Comp. Eng., Northeastern Univ., Boston, MA, 2003.

[6] J.Bhaskar "VHDL primer"'3rd Edition 2003,Pearson Education, Inc.

[7]www.digchip.com

[8] www.powerdesigner.com