PSPRIMER-FPGASP3

SPARTAN3 Development Kit

Hardware & Software User Manual

Contents

1. *Introduction*

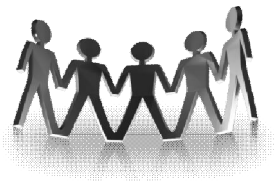
Thank you for purchasing the Xilinx Spartan-3 Development Kit. You will find it useful in developing your Spartan-3 FPGA application.

FPGASP3 Kit (DPK) is an exclusive general-purpose kit for the SPARTAN family. The intention of the design is to endorse the engineers and scholars to exercise and explore the capabilities of FPGA architectures with many interfacing modules on board point LEDs, Slide switches, Traffic light, LCD, 7-Seg, UART, VGA, and PS/2 with ease to create a stand-alone versatile test platform.

1.1 – Packages

- Spartan3 Development Kit (XC3S200)
- Serial Port Cable
- JTAG Programming Cable
- Printed User Manual
- CD contains
  - Software (Programmers, ISE)
  - Example Programs
  - User Manual

1.2 - Technical or Customer Support



E-mail questions to : support@pantechsolutions.net

Send questions by mail to
Pantech Solutions Pvt Ltd.,
151/34, Mambalam High Road,
Sri Ranga Building, T. Nagar,
Chennai - 600 017
Tamilnadu, India
Phone : +91-44-4260 6470
Fax : +91-44-4260 6350
Website : www.pantechsolutions.net

2. *Key Components and Features*

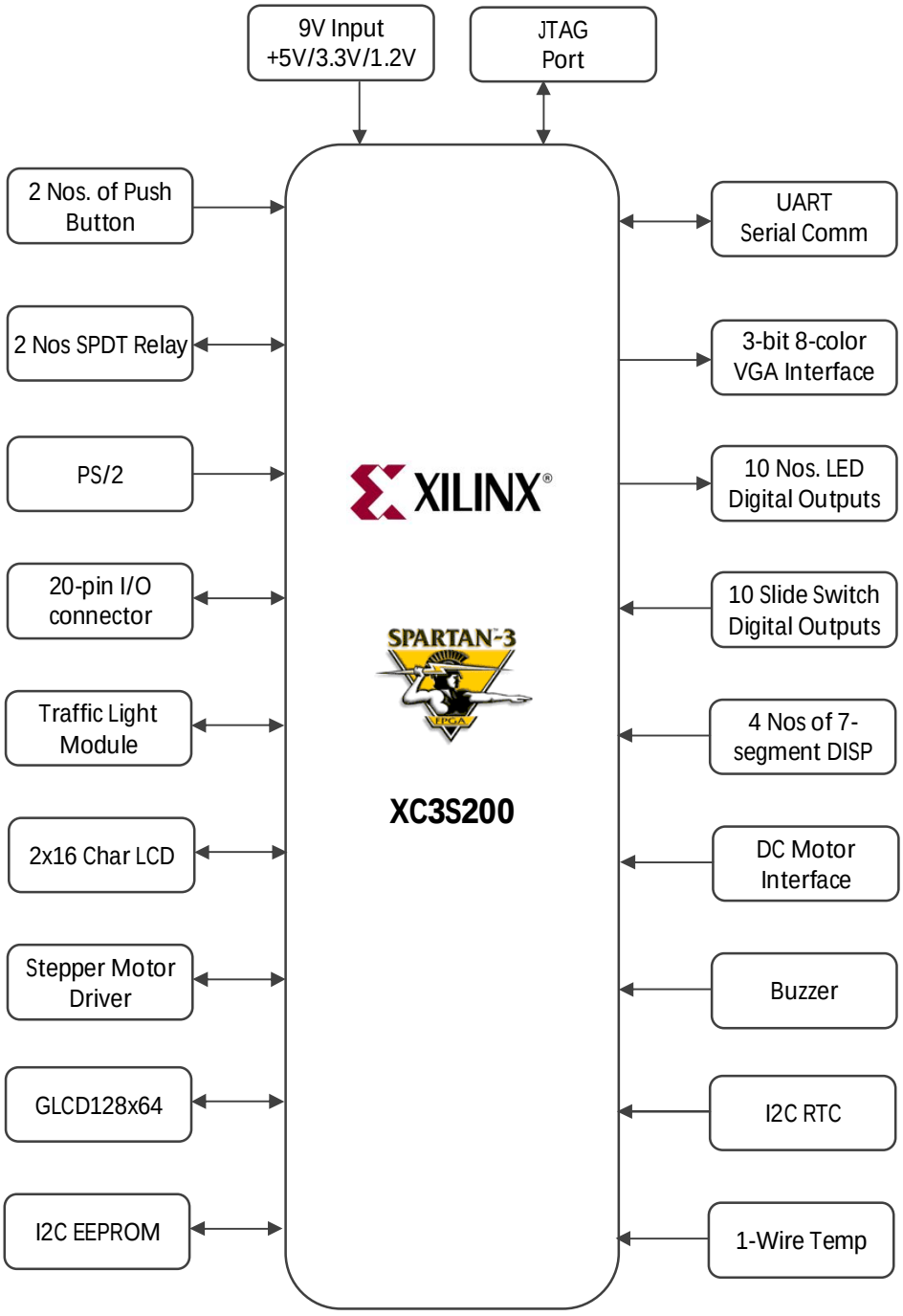
On-Chip Features

- 200,000-gate Xilinx Spartan 3 FPGA in a 144-TQFP (XC3S200-4TQG144C)
    - 4,320 logic cell equivalents
    - Twelve 18K-bit block RAMs (216K bits)
    - Twelve 18x18 hardware multipliers
    - Four Digital Clock Managers (DCMs)
    - Up to 97 user-defined I/O signals

On-Board Features

- 10 Nos. Slide Switches for digital inputs
- 10 nos. of Point LEDs for Digital outputs
- 2 Nos. of Push Button
- 2x16 Character LCD interface
- 4 Nos. 7-segment LED CA display
- 2 Nos. of 5V SPDT Relay with termination.
- 4x4 Matrix Keypad interface
- 4-Way Traffic Light controller Module
- Stepper motor Driver interface
- DC Motor interface controlled by PWM
- 3-bit, 8-color VGA display port.
- RS-232 Serial Port.
- PS/2-connector(for mouse/keyboard interface) port
- 128x64 GLCD Module Interface (Optional)
- Serial EEPROM (Optional)
- I2C Real Time Clock with battery back-up (Optional)
- 1-Wire Digital Temperature Sensor (Optional)
- 50 MHz crystal oscillator clock source
- 20-pin I/o connector for interface external peripherals modules
- JTAG port for download user program through cable
- 9V AC/DC power input through adapter
- On-board 5V, 3.3V, 2.5V, and 1.2V regulators.

2.1- General Block Diagram



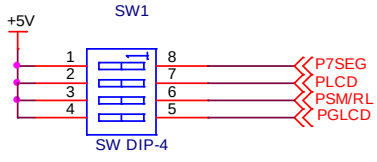
3. *Jumper & Switch Details*

Stepper / Relay **JP5**		Internal Supply (+5V) External Supply(+5V)
DC Motor **JP6**		Internal Supply (+5V) External Supply(+5V)
Buzzer (P5) **JP7**		Enable Buzzer Disable Buzzer

Switch Details

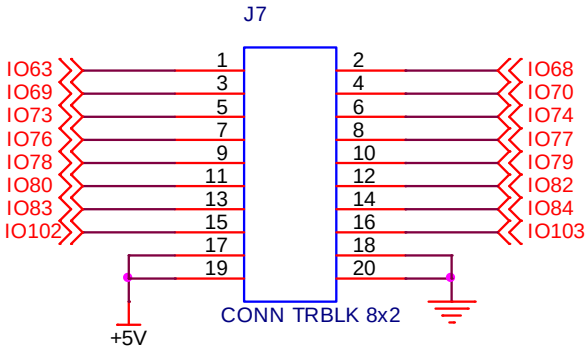
Program Execution Mode Selection (EXE MODE)		
JTAG/PROM **J6**		Execution through ***JTAG*** Execution through ***PROM***

GLCD / Traffic Light Selection		
GLCD/Traffic **SW28**		Select ***GLCD*** Select ***TRAFFIC***

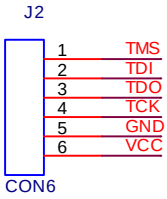
+5V Power Selection


4. *Connector Details*

20pin – Box Connector



JTAG Connector



5. *Power Supply*

The external power can be AC or DC, with a voltage between (9V, 1A output) at 230V AC input. The SPARTAN3 board produces +5V using an LM7805 voltage regulator, which provides supply to the peripherals.

USB socket meant for power supply and USB communication, user can select either USB or Ext power supply through **SW1**. Separate **On/Off** Switch (SW1) for controlling power to the board.

ON/OFF SW2		Power +5V ***ON*** **-**External through ***Adaptor*** Power +5V ***ON*** **-** Internal through ***USB***

There are multiple voltages supplied on the Spartan-3 Evaluation Kit, 3.3V, 2.5V and 1.2V regulators. Similarly, the 3.3V regulator feeds all the VCCO voltage supply inputs to the FPGA's I/O banks and powers most of the components on the board. The 2.5V regulator supplies power to the FPGA's VCCAUX supply inputs. The VCCAUX voltage input supplies power to Digital Clock Managers (DCMs) within the FPGA and supplies some of the I/O structures.

In specific, all of the FPGA's dedicated configuration pins, such as DONE, PROG_B, CCLK, and the FPGA's JTAG pins, are powered by VCCAUX. The FPGA configuration interface on the board is powered by 3.3V. Consequently, the 2.5V supply has a current shunt resistor to prevent reverse current. Finally, a 1.2V regulator supplies power to the FPGA's VCCINT voltage inputs, which power the FPGA's core logic. The board uses four discrete regulators to generate the necessary voltages.

6. *On-board Peripherals*

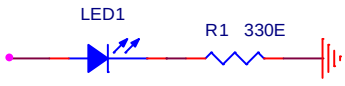
The Development Kit comes with many interfacing options

- 8 Nos. Slide Switches for digital inputs
- 8 nos. of Point LEDs for Digital outputs
- 2x16 Character LCD interface
- 4 Nos. 7-segment LED CA display
- 2 Nos. of 5V SPDT Relay with termination.
- 4x4 Matrix Keypad interface
- 4-Way Traffic Light controller Module
- Stepper motor Driver interface
- DC Motor interface controlled by PWM
- Buzzer Interface
- UART for serial port communication through PC
- GLCD | I2C EEPROM | RTC | 1-Wire Temp Sensor (Optional)
- JTAG Programmer
- Clock Source
- 3-bit, 8 Color VGA Interface
- PS/2 Keyboard interface

6.1 - Light Emitting Diodes

- Light Emitting Diodes (LEDs) are the most commonly used components, usually for displaying pin's digital states.

- The FPGASP3 KIT has 8 nos., of Point LEDs, connected with port pins (details tabulated below); the cathode of each LED connects to ground via a 330Ω resistor. To light an individual LED, drive the associated FPGA control signal to **High.**

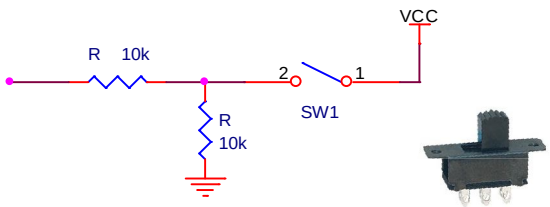
	Point LEDs	XC3S200 - Pins	LED Selection
DIGITAL OUTPUTS	LED-D9	P14	 Make Pin High – LED ON Make Pin Low – LED OFF
	LED-D10	P15	
	LED-D11	P17	
	LED-D12	P18	
	LED-D13	P20	
	LED-D14	P21	
	LED-D15	P23	
	LED-D16	P24	
	LED-D29	P50	
	LED-D30	P53	

6.2 – Digital Inputs

- This is another simple interface, of 8-Nos. of slide switch, mainly used to give an input to the port lines, and for some control applications also.

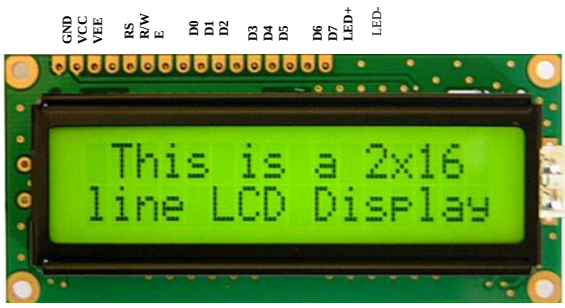
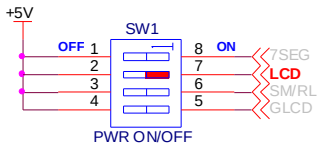
- The FPGASP3 KIT, slide switches (SW20-SW27) directly connected with FPGA I/O lines (details tabulated below), user can give logical inputs **high** through slide switches.

The switches are connected to +3.3V, in order to detect a switch state, by default lines are pull-downed through resistors. The switches typically exhibit about 2 ms of mechanical bounce and there is no active de-bouncing circuitry, although such circuitry could easily be added to the FPGA design programmed on the board. A 10KΩ series resistor provides nominal input protection.

	Slide Switch	XC3S200 - Pins	Slide Switch Logic
DIGITAL INTPUTS	SW20	P25	 Make Switch Close – High Make Switch Open – Low
	SW21	P26	
	SW22	P27	
	SW23	P28	
	SW24	P30	
	SW25	P31	
	SW26	P32	
	SW27	P33	
	SW29	P93	
	SW30	P103	

6.3 – 2x16 Char LCD Display

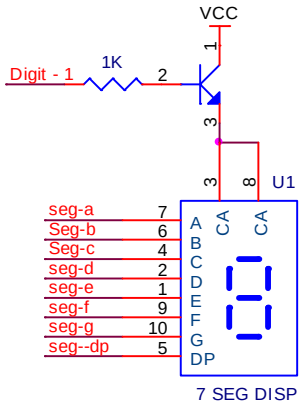
The 2x16 character LCD interface card with supports both modes 4-bit and 8-bit interface, and also facility to adjust contrast through trim pot. In 8-bit interface 11 lines needed to create 8-bit interface; 8 data bits (D0 – D7), three control lines, address bit (RS), read/write bit (R/W) and control signal (E). The LCD controller is a standard KS0070B or equivalent, which is a very well-known interface for smaller character based LCDs.

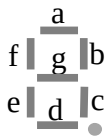
<table>
<tr><th></th><th>LCD MODULE</th><th>XC3S200-Pins</th><th>2x16 LCD Selection</th></tr>
<tr><td rowspan="3">CONTROL</td><td>RS</td><td>P104</td><td rowspan="11"> 
</td></tr>
<tr><td>RW</td><td>P105</td></tr>
<tr><td>E</td><td>P107</td></tr>
<tr><td rowspan="8">DATA LINES</td><td>D0</td><td>P108</td></tr>
<tr><td>D1</td><td>P112</td></tr>
<tr><td>D2</td><td>P113</td></tr>
<tr><td>D3</td><td>P116</td></tr>
<tr><td>D4</td><td>P118</td></tr>
<tr><td>D5</td><td>P119</td></tr>
<tr><td>D6</td><td>P122</td></tr>
<tr><td>D7</td><td>P123</td></tr>
<tr><td colspan="2"> 
Make switch SW1 to ‘LCD’ label marking position</td><td colspan="2"> 
</td></tr>
</table>

6.4 – 7-Segment Display

The Spartan-3 FPGASP3 Kit has a four-character, seven segments LED display controlled by FPGA user-I/O pins. Each digit shares eight common control signals to light individual LED segments. Each individual character has a separate anode control input. The pin number for each FPGA pin connected to the LED display is shown in below table. To light an individual signal, drive the individual segment control signal Low along with the associated anode control signal for the individual character.

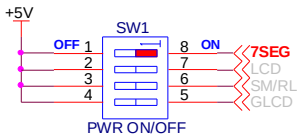
	7-Segment	XC3S200-Pins	7-Segment Display
Digit Select	Digit - 1	P124	
	Digit – 2	P125	
	Digit – 3	P127	
	Digit - 4	P128	
Segment Lines	Seg - a	P129	
	Seg – b	P130	
	Seg – c	P131	
	Seg – d	P132	
	Seg – e	P135	
	Seg – f	P137	
	Seg – g	P140	
	Seg – dp	P141	





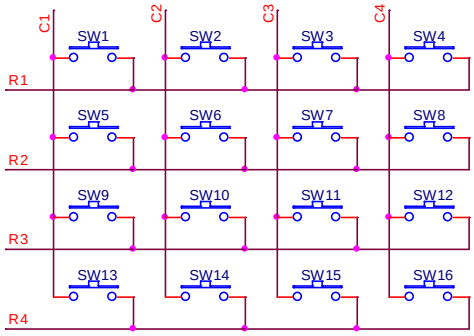
Make **high** to - **digit** selection
Make **low** to - **segment**

Make switch **SW1** to '7SEG' label marking position



6.5 - 4x4 Matrix keypad and Push Button

Keypads arranged by matrix format, each row and column section pulled by high, all row lines and column lines connected directly by the i/o pins.

	4x4 Matrix Keys	XC3S200-pins	Matrix Format
ROW	ROW-1	P85	
	ROW-2	P86	
	ROW-3	P87	
	ROW-4	P89	
COLUMN	COLUMN-1	P90	
	COLUMN-2	P92	
	COLUMN-3	P95	
	COLUMN-4	P96	

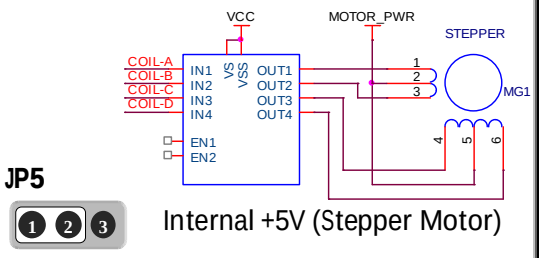
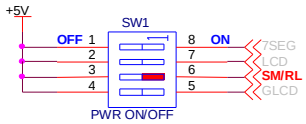
The Spartan3 FPGA Kit has four contact push button switches for interrupt input.

PUSH BUTTON	XC3S200-pins
SW31	P103
SW32	P102

6.6 – Stepper Motor

The ULN2803A is a high-voltage, high-current Darlington transistor array. The device consists of eight NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector-current rating of each Darlington pair, 500 mA.

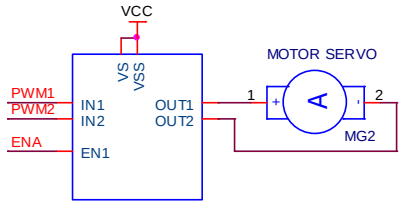
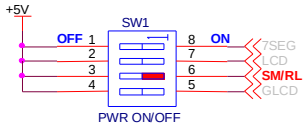
ULN2803 is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.

	Stepper Motor(5V)	XC3S200-Lines	Stepper Motor PWR Select
STEPPER MOTOR	COIL-A	P82	
	COIL-B	P83	
	COIL-C	P99	
	COIL-D	P100	
 Make switch **SW1** to ***SM/RL*** label marking position.			

For Motor/relay designer get power from on-board (internal) or external supply through jumper **JP5**, by default JP5 pin 1&2 shorted

6.7 - DC Motor

5V DC Motor speed has controlled through PWM signal. Motor can run both clockwise/counter clockwise, Motor speed controlled by varying ENA (duty cycle) signal through program.

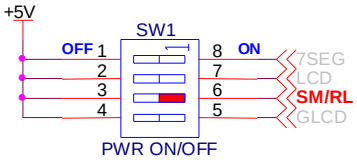
	DC Motor(5V)	XC3S200-Lines	DC Motor PWR Select
DC MOTOR	COIL-A	P82	
	COIL-B	P83	
	ENA	P99	JP6  Internal +5V (Stepper Motor)
 Make switch **SW1** to ***SM/RL*** label marking position.			

For DC Motor designer get power from on-board (internal) or external supply through jumper JP6, by default JP6 pin 1&2 shorted.

6.8 – Relay Interface

ULN2803 is used as a driver for port I/O lines, drivers output connected to relay modules. Connector provided for external power supply if needed.

Relay Module : Spartan3 FPGA pins (Realy1 – P59) and (Relay2-P60) for relay module, make port pins to high, relay will activated

	RELAY SPDT	XC3S200-pins	RELAY Power Select
RELAY Modules	Relay-1	P59	
	Relay-2	P60	**JP5**  Internal +5V (Stepper Motor)
	**Note** : Relay selection make switch **SW1** to ***SM/RL*** label marking position		

For Motor/relay designer get power from on-board (internal) or external supply through jumper JP5, by default JP5 pin 1&2 shorted.

6.9 – Buzzer Interface

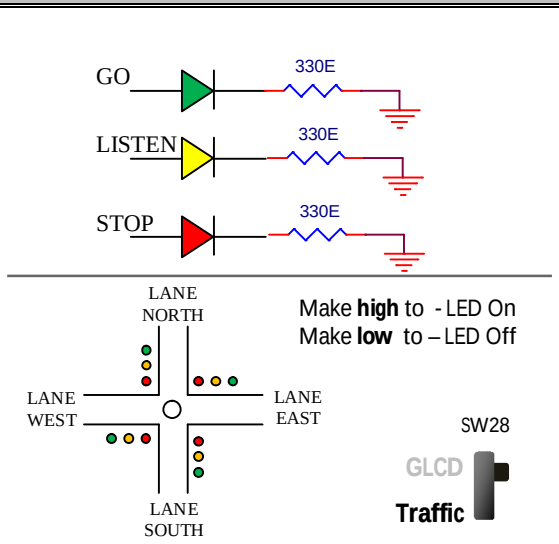
5V continuous buzzer connected through FPGA's I/O pins (P5), to enable buzzer place jumper JP7 at E label mark position.

Buzzer Module : Spartan3 FPGA pins (Buzzer – P5), make port pins to high, buzzer will activated

	5V Buzzer	XC3S200-pins	RELAY Power Select
Buzzer	Buzzer	P5	**JP7**  - To Enable Buzzer

6.10 – Traffic Light Controller

Traffic light controller card consist of 12 Nos. point led arranged by 4Lanes. Each lane has Go (Green), Listen(Yellow) and Stop(Red) LED is being placed. Each LED has provided for current limiting resistor to limit the current flows to the LEDs.

LAN Direction	XC3S200-pins	LED's	Traffic Light Controller
NORTH	P40	D19-Go	
	P36	D18-Listen	
	P35	D17-Stop	
WEST	P46	D22-Go	
	P44	D21-Listen	
	P41	D20-Stop	
SOUTH	P51	D25-Go	
	P50	D24-Listen	
	P47	D23-Stop	
EAST	P56	D28-Go	
	P53	D27-Listen	
	P52	D26-Stop	

Note : Make **SW28** to "***Traffic"*** label marking position

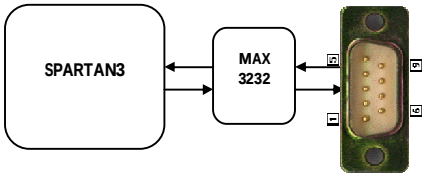
7 - Peripherals Section - II

7.1 - RS-232 Communication (USART)

USART stands for Universal Synchronous Asynchronous Receiver Transmitter. FPGASP3 Kit provides an RS232 port that can be driven by the Spartan-3 FPGA. A subset of the RS232 signals is used on the Spartan 3 kit to implement this interface (RxD and TxD signals).

- RS-232 communication enables point-to-point data transfer. It is commonly used in data acquisition applications, for the transfer of data between the microcontroller/FPGA and a PC.

- The voltage levels of a FPGA and PC are not directly compatible with those of RS-232, a level transition buffer such as MAX3232 be used.

	UART DB-9 Connector	SPARTAN3 FPGA Lines	Serial Port Section
UART	TXD	P15	
	RXD	P14	

7.2 – Clock Source

The FPGASP3 Kit has a dedicated 50 MHz series clock oscillator source and an optional socket for another clock oscillator source.

	U18	Signal	SPARTAN3 FPGA Lines	Crystal Oscillator
Oscillator	50MHz	Clock	P55	

7.3 – JTAG Programmer

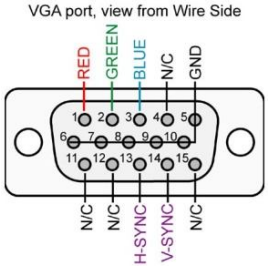
The FPGASP3 Kit includes a JTAG programming and debugging chain. Pantech JTAG3 low-cost parallel to JTAG cable is included as part of the kit and connects to the JTAG header. DB-25 parallel port connector to 6 pin female header connector. The JTAG cable connect directly to the parallel port of a PC and to a standard 6 pin JTAG programming header in the kit, can program a devices that have a JTAG voltage of 1.8V or greater.

	6-Pin Header	JTAG Signals	SPARTAN3 FPGA Lines	JTAG Cable – III
JTAG Programmer	1	TMS	P111	
	2	TDI	P144	
	3	TDO	P109	
	4	TCK	P110	
	5	GND		
	6	VCC		

The Pantech low-cost parallel port to JTAG cable fits directly over the header stake pins, as shown in above figure. When properly fitted, the cable is perpendicular to the board. Make sure that the signals at the end of the JTAG cable align with the labels listed on the board. The other end of the Pantech cable connects to the PC's parallel port. The Pantech cable is directly compatible with the Xilinx iMPACT software.

7.4 - VGA Interface

The FPGASP3 Kit includes a VGA display port through DB15 connector, Connect this port directly to most PC monitors or flat-panel LCD displays using a standard monitor cable As shown in table, the Spartan-3 FPGA controls five VGA signals: RED (R) its 1ST pin in connector, GREEN (G) its 2nd pin, BLUE (B) its 3rd pin, Horizontal Sync (HS) 13th pin, and Vertical Sync (VS) its 14th pin, all available on the VGA connector.

DB-15 Connector	VGA Signals	SPARTAN3 FPGA Lines	VGA port, view from Wire Side
	Vertical Sync(VS)	P6	
	Horizontal Sync(HS)	P7	
	Blue	P8	
	Green	P10	
	Red	P11	

Each color line has a series resistor to provide 3-bit color, with one bit each for Red, Green, and Blue. The series resistor uses the 75 ohm VGA cable termination to ensure that the color signals remain in the VGA-specified 0V to 0.7V range. The HS and VS signals are TTL level. Drive the R, G, and B signals High or Low to generate the eight possible colors shown in below table.

RED	GREEN	BLUE	RESULTING COLOUR
0	0	0	BLACK
0	0	1	BLUE
0	1	0	GREEN
0	1	1	CYAN
1	0	0	RED
1	0	1	MAGNETA
1	1	0	YELLOW
1	1	1	WHITE

VGA signal timing is specified, published, copyrighted, and sold by the Video Electronics Standards Association (VESA). The following VGA system and timing information is provided as

an example of how the FPGA might drive VGA monitor in 640 by 480 modes. For more precise information or for information on higher VGA frequencies, refer to documents available on the VESA website or other electronics Websites: Video Electronics Standards Association

http://www.vesa.org

VGA Timing Information
http://www.epanorama.net/documents/pc/vga_timing.html

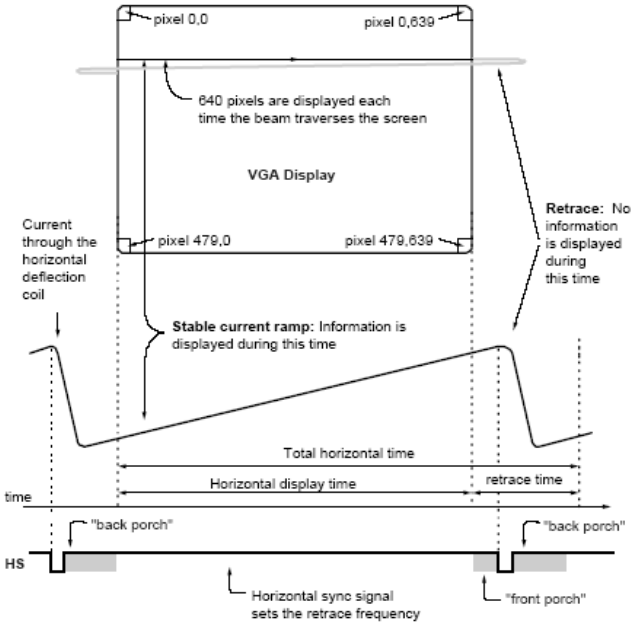
Signal Timing for a 60Hz, 640x480 VGA Display

CRT-based VGA displays use amplitude-modulated, moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel by- pixel basis.

Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays. Consequently, the following discussion pertains to both CRTs and LCD displays. Within a CRT display, current waveforms pass through the coils to produce magnetic fields that deflect electron beams to transverse the display surface in a "raster" pattern, horizontally from left to right and vertically from top to bottom.

As shown in below figure, information is only displayed when the beam is moving in the "forward" direction—left to right and top to bottom—and not during the time the beam returns back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking" periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass.

The size of the beams, the frequency at which the beam traces across the display, and the frequency at which the electron beam is modulated determine the display resolution. Modern VGA displays support multiple display resolutions, and the VGA controller indicates the resolution by producing timing signals to control the raster patterns. The controller produces TTL-level synchronizing pulses that set the frequency at which current flows through the deflection coils, and it ensures that pixel or video data is applied to the electron guns at the



correct time. Video data typically comes from a video refresh memory with one or more bytes assigned to each pixel location.

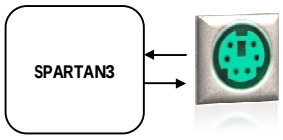
The Spartan-3 Evaluation Kit uses three bits per pixel, producing one of the eight possible colors shown in above table. The controller indexes into the video data buffer as the beams move across the display. The controller then retrieves and applies video data to the display at precisely the time the electron beam is moving across a given pixel.

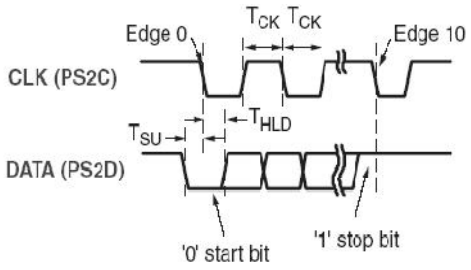
The VGA controller generates the HS (horizontal sync) and VS (vertical sync) timings signals and coordinates the delivery of video data on each pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the "refresh" frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies in the 60 Hz to 120 Hz range.

The number of horizontal lines displayed at a given refresh frequency defines the horizontal "retrace" frequency.

7.5 - PS/2 Interface

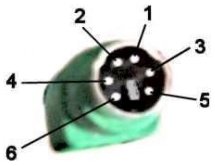
The FPGASP3 Kit includes a PS/2 port and the standard 6-pin mini-DIN connector, labeled U11 on the board. User can connect PS/2 Devices like keyboard, mouse to the FPGASP3 KIT. PS/2's DATA (P8) and CLK (P10) lines connected to SPARTAN3 FPGA I/O Lines.

6PIN MINI Connector	PS/2	SPARTAN3 FPGA Lines	PS/2 PORT SELECT
U11 PS/2	DATA	P1	
	CLK	P2	



PS/2 keyboard connector (MINI-DIN6)

Connector Pin #	Purpose
Pin 1	KBDAT (data)
Pin 2	not used
Pin 3	GND
Pin 4	VCC (+5V)
Pin 5	KBDCLK (clock)
Pin 6	not used



ıouse connector pinout is identical to PS/2 keyboard.

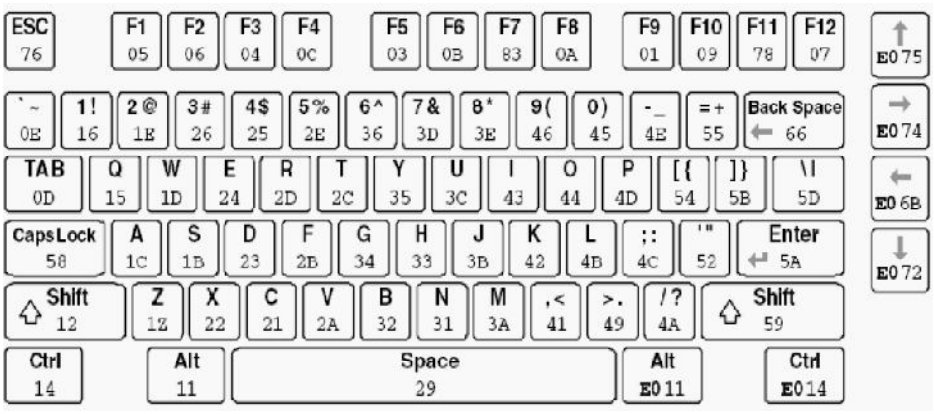
Both a PC mouse and keyboard use the two-wire PS/2 serial bus to communicate with a host device, the Spartan-3 FPGA in this case. The PS/2 bus includes both clock and data. Both a mouse and keyboard drive the bus with identical signal timings and both use 11-bit words that include a start, stop and odd parity bit. However, the data packets are organized differently for a mouse and keyboard. Furthermore, the keyboard interface allows bidirectional data transfers so the host device can illuminate state LEDs on the Keyboard.

The PS/2 bus timing appears as shown in above figure. The clock and data signals are only driven when data transfers occur; otherwise they are held in the idle state at logic High. The timing defines signal requirements for mouse-to-host communications and bidirectional keyboard communications. The attached keyboard or mouse writes a bit on the data line when the clock signal is High, and the host reads the data line when the clock signal is Low.

Keyboard

The keyboard uses open-collector drivers so that either the keyboard or the host can drive the two-wire bus. If the host never sends data to the keyboard, then the host can use simple input pins. A ps/2-style keyboard uses scan codes to communicate key press data nearly all keyboards in use today are ps/2 style. Each key has a single, unique scan code that is sent whenever the corresponding key is pressed.

The scan codes for most keys appear in below figure. If the key is pressed and held, the keyboard repeatedly sends the scan code every 100 ms or so. When a key is released, the keyboard sends an "f0" key-up code, followed by the scan code of the released key. the keyboard sends the same scan code, regardless if a key has different shift and non-shift characters and regardless whether the shift key is pressed or not. The host determines which character is intended. Some keys, called extended keys, send an "e0" ahead of the scan code and furthermore, they might send more than one scan code. When an extended key is released, an "e0 f0" key-up code is sent, followed by the scan code.



The host can also send commands and data to the keyboard. Below figure provides a short list of some often-used

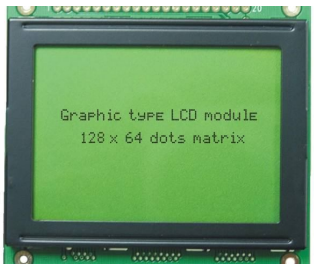
Command	Description
ED	Turn on/off Num Lock, Caps Lock, and Scroll Lock LEDs
EE	Echo. Upon receiving an echo command, the keyboard replies with the same scan code "EE".
F3	Set scan code repeat rate. The keyboard acknowledges receipt of an "F3" by returning an "FA", after which the host sends a second byte to set the repeat rate.
FE	Resend. Upon receiving a resend command, the keyboard resends the last scan code sent
FF	Reset. Resets the keyboard

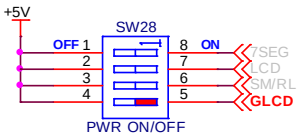
The keyboard sends commands or data to the host only when both the data and clock lines are High, the Idle state, Because the host is the bus master, the keyboard checks whether the host is sending data before driving the bus. The clock line can be used as a clear to send signal. If the host pulls the clock line Low, the keyboard must not send any data until the clock is released. The keyboard sends data to the host in 11-bit words that contain a '0' start bit, followed by eight bits of scan code (LSB first), followed by an odd parity bit and terminated with a '1' stop bit.

7.6 - 128x64 GLCD Graphical LCD

The **FPGASP3 KIT** is the GLCD. 14 pins are needed to create 8-bit interface; 8 data bits (DB0-DB7), two chip select line (CS1) and (CS2), address bit (R/S), read/write bit (R/W) and control signal (E) and Reset (RST). The GLCD controller is a standard S6B0108 or equivalent, which is a very well-known interface for Graphical based LCDs.

Figure below illustrate the GLCD part of the design and which pins are used for the interface. The GLCD is powered from the 5V power supply enabled by switch **SW1**.

<table>
<tr><th></th><th>GLCD</th><th>XC3S200 LINES</th><th>128x64 GLCD Selection</th></tr>
<tr><td rowspan="5">CONTROL L LINES</td><td>CS1</td><td>P35</td><td rowspan="14"> 
</td></tr>
<tr><td>CS2</td><td>P36</td></tr>
<tr><td>RS</td><td>P40</td></tr>
<tr><td>R/W</td><td>P41</td></tr>
<tr><td>E</td><td>P44</td></tr>
<tr><td rowspan="9">LCD – DATA LINES</td><td>DB0</td><td>P46</td></tr>
<tr><td>DB1</td><td>P47</td></tr>
<tr><td>DB2</td><td>P50</td></tr>
<tr><td>DB3</td><td>P51</td></tr>
<tr><td>DB4</td><td>P52</td></tr>
<tr><td>DB5</td><td>P53</td></tr>
<tr><td>DB6</td><td>P56</td></tr>
<tr><td>DB7</td><td>P57</td></tr>
<tr><td>RST</td><td>High</td></tr>
<tr><td colspan="3">  Make switch SW1 and SW28 to ‘GLCD’ label marking position</td><td> 


</td></tr>
</table>

Pin Details of GLCD

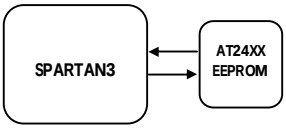
Pin No.	Symbol	I/O Type	Description
1 2	/CSA /CSB	I	Chip selection When CS1=L,CS2=H, select IC1 When CS1=H,CS2=L, select IC2
3	VSS	Supply	Ground
4	VDD	Supply	Power supply
5	V0	Supply	LCD driver supply voltage
6	D/I		Data input/output pin of internal shift register **MS SHL DIO1 DIO2** H H Output Output H L Output Output L H Input Output L L Output Input
7	R/W		Read or Write **RW Description** H Data appears at DB[7:0] and can be read by the CPU while E= H CS1B=L,CS2B=L and CS3=H L Display data DB[7:0] can be written at falling edge of E when CS1B=L, CS2B=L and CS3=H.
8	E		Enable signal **E Description** H Read data in DB[7:0] appears while E= "High". L Display data DB[7.0] is latched at falling edge of E.
9~16	DB0~ DB7	I/O	Data bus [0~7] Bi-directional data bus
17	/RST	I	Reset signal. When RSTB=L 【1】 ON/OFF register becomes set by 0.(display off) 【2】 display start line register becomes set by 0 (Z-address 0 set, display from line 0) 【3】 After releasing reset , this condition can be changed only by instruction.
18	VEE	Power	VEE is connected by the same voltage.
19	A		Back light anode
20	K		Back-light cathode

7.7 – Serial EEPROM

The AT24C01A/02/04/08/16 provides 1024/2048/4096/8192/16384 bits of serial electrically erasable and programmable read-only memory (EEPROM) organized as 128/256/512/1024/2048 words of 8 bits each. The device is optimized for use in many industrial and commercial applications where low-power and low-voltage operation are essential.

Features of AT24Cxx:

- Internally Organized 128 x 8 (1K), 256 x 8 (2K), 512 x 8 (4K)
- 2-wire Serial Interface
- Bi-directional Data Transfer Protocol
- 100 kHz (1.8V, 2.5V, 2.7V) and 400 kHz (5V) Compatibility
- Write Protect Pin for Hardware Data Protection
- 8-byte Page (1K, 2K), 16-byte Page (4K, 8K, 16K) Write Modes
  - Data Retention: 100 Years.

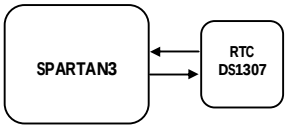
	I2C EEPROM	XC3S200-pins	Serial EEPROM
AT 24XX	SDA	P97	
	SCL	P98	

7.8 - Real Time Clock (DS1307)

The Real Time Clock (RTC) is a set of counters for measuring time when system power is on, and optionally when it is off. It uses little power in Power-down mode. On the LPC2148, the RTC can be clocked by a separate 32.768 KHz oscillator, or by a programmable prescale divider based on the VPB clock. Also, the RTC is powered by its own power supply pin, VBAT, which can be connected to a battery or to the same 3.3 V supply used by the rest of the device.

Features

- Measures the passage of time to maintain a calendar and clock.
- Ultra Low Power design to support battery powered systems.
- Provides Seconds, Minutes, Hours, Day of Month, Month, Year, Day of Week, Day of Year.
- Dedicated 32 kHz oscillator or programmable pre-scalar from VPB clock.
- Dedicated power supply pin can be connected to a battery or to the main 3.3 V.

	I2C RTC	XC3S200	Real Time Clock
DS1307	SDA	P97	
	SCL	P98	

VLSI LAB Examples

(L-Scheme)

Contents

8. VLSI Lab Experiments

SIMULATION OF VHDL CODE FOR COMBINATIONAL CIRCUIT (Sum of Product)

1A

Description

Optimize a 4 variable combinational function (SOP or POS), describe it in VHDL code and Simulate it.
Example: F= (0, 5, 8, 9, 12) in sop

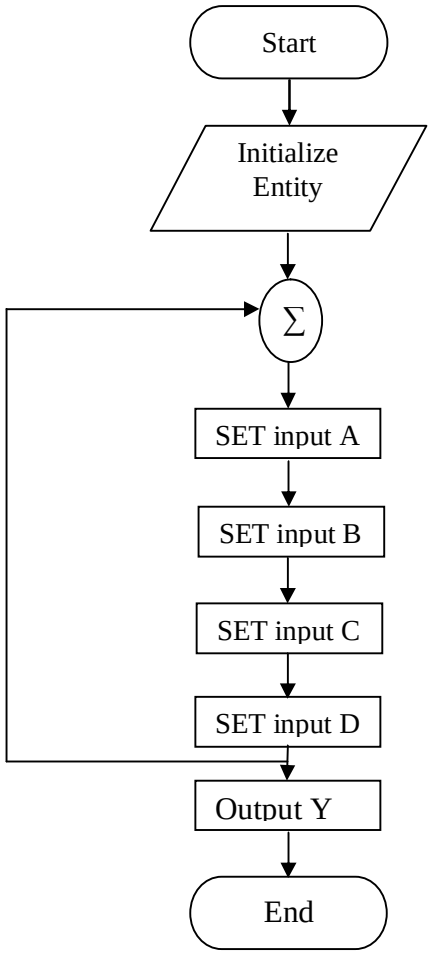
Truth Table for Sum of Product Simplification

A	B	C	D	Y	Sum of Product
0	0	0	0	1	A'B'C'D'
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	A'BC'D
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	1	AB'C'D'
1	0	0	1	1	AB'C'D
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	1	ABC'D'
1	1	0	1	0	
1	1	1	0	0	
1	1	1	1	0	

Boolean Expression:

Y = A'B'C'D'+A'BC'D+AB'C'D'+AB'C'D+ABC'D'
Y = B'C'D'(A'+A) +C'(A'BD+AB'D+ABD')
Y = B'C'D'+A'BC'D+AB'C'D+ABC'D'

Flow Chart

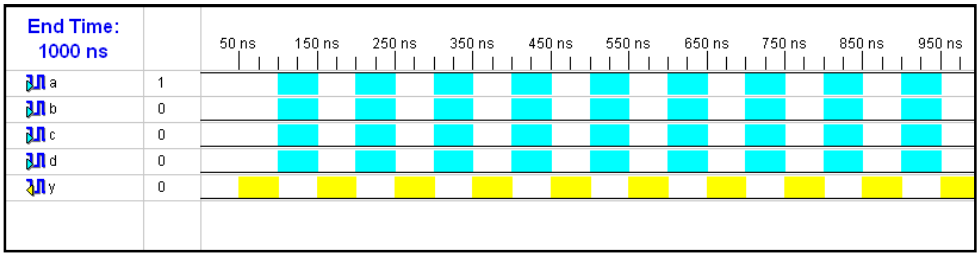

Start
Initialize Entity
Σ
SET input A
SET input B
SET input C
SET input D
Output Y
End

Code Listing

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sop is
port(a,b,c,d : in std_logic;
     y : out std_logic
);
 end sop;

architecture data of sop is

begin

 y <= (( not b and not c and not d) or (not a and b and not c and
d) or (a and not b and not c and  d) or (a and  b and not c and not
d) );

end data;
```

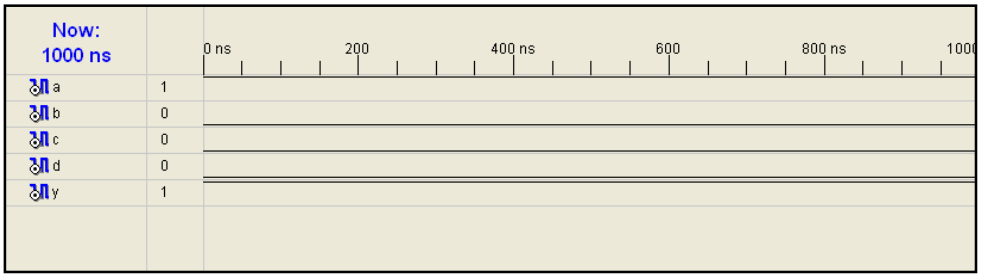
Input Waveform

We give input a=high in waveform window



Output waveform

Finally we get output in simulation window(y=high according to that SOP Equation)



SIMULATION OF VHDL CODE FOR COMBINATIONAL CIRCUIT (Product of Sum)

1.B

Description

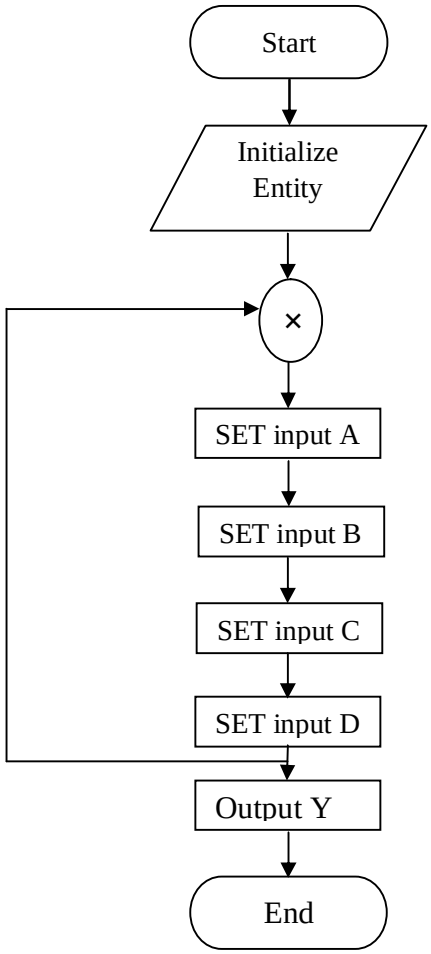
Optimize a 4 variable combinational function (POS), describe it in VHDL code and Simulate it.
Example: F= (0, 5, 8, 9, 12) in POS.

Truth Table for Product of Sum Simplification

A	B	C	D	Y	Product of Sum
0	0	0	0	1	
0	0	0	1	0	A'+B'+C'+D
0	0	1	0	0	A'+B'+C+D'
0	0	1	1	0	A'+B'+C+D
0	1	0	0	0	A'+B+C'+D'
0	1	0	1	1	
0	1	1	0	0	A'+B+C+D'
0	1	1	1	0	A'+B+C+D
1	0	0	0	1	
1	0	0	1	1	
1	0	1	0	0	A+B'+C+D'
1	0	1	1	0	A+B'+C+D
1	1	0	0	1	
1	1	0	1	0	A+B+C'+D
1	1	1	0	0	A+B+C+D'
1	1	1	1	0	A+B+C+D

Y = (A'+B'+C'+D) (A'+B'+C+D') (A'+B'+C+D) (A'+B+C'+D') (A'+B+C+D') (A'+B+C+D) (A+B'+C+D') (A+B'+C+D) (A+B+C'+D) (A+B+C+D') (A+B+C+D)

Flow Chart

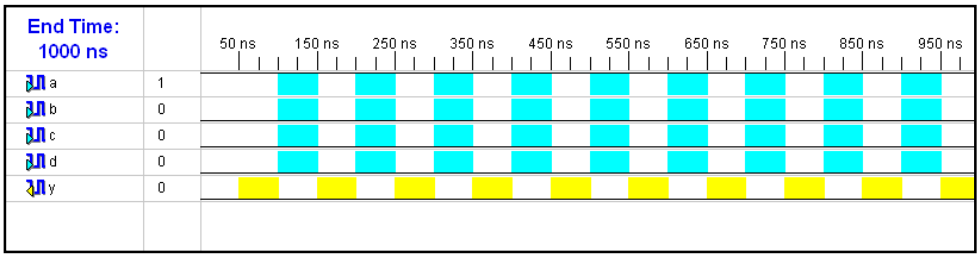

Start
Initialize Entity
×
SET input A
SET input B
SET input C
SET input D
Output Y
End

Code Listing

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pos is
port(a,b,c,d : in std_logic;
     y : out std_logic
);
 end pos;

architecture data of pos is

begin

 y <= ((not a or not b or not c or  d ) and(not a or not b or  c or
not  d) and (not a or not b or  c or  d) and (not a or   b or not c
or not d)and (not a or b or  c or not d)and(not a or b or c or d)
and (a or not b or c not d) and (a or not b or c or d)and(a or b or
not c or d  ) and (a or b or c or not d) and (a or b  or c or d)) ;

end data;
```

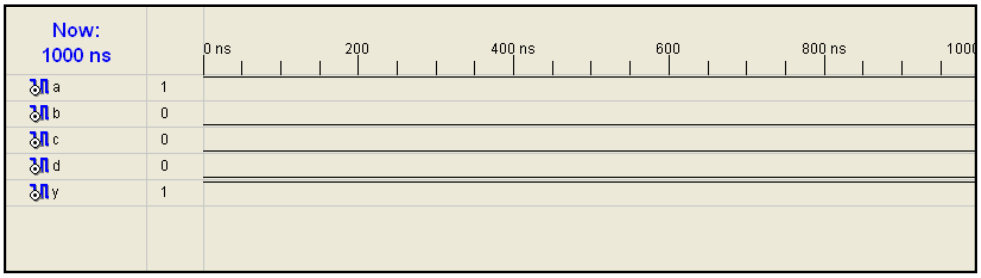
Input Waveform

We give input a=high in waveform window



Output waveform

Finally we get output in simulation window(y=high according to that POS Equation)



SIMULATION OF VHDL CODE FOR ARITHMETIC CIRCUITS

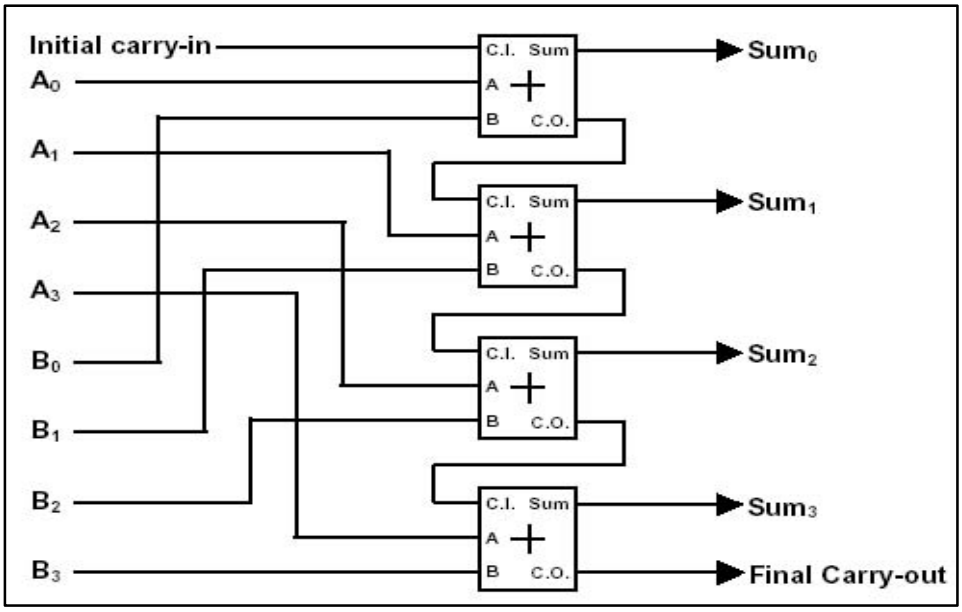
2

Description

Design and Develop the circuit for the following arithmetic function in VHDL Codes and Simulate it. Addition, Subtraction Multiplication (4 x 4 bits)

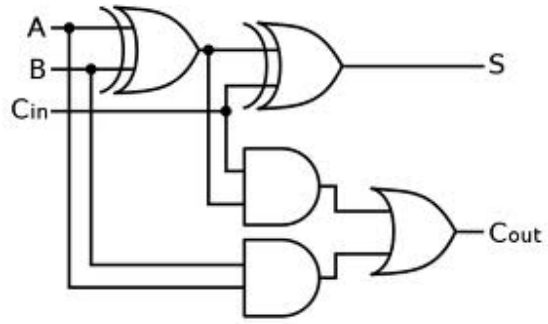
2.1 – Addition (Program for 4-bit addition using 4 bit Ripple adder)

It is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a C_{in}, which is the C_{out} of the previous adder. This kind of adder is called a ripple-carry adder, since each carry bit "ripples" to the next full adder.

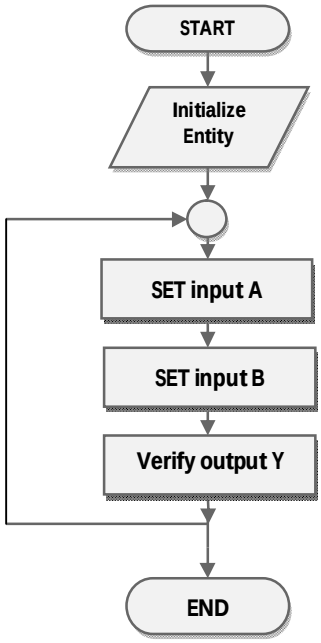


Full Adder

The full-adder circuit adds three one-bit binary numbers (Cin, A, B) and outputs two one-bit binary numbers, a sum (S) and a carry (Cout).



Flow Chart

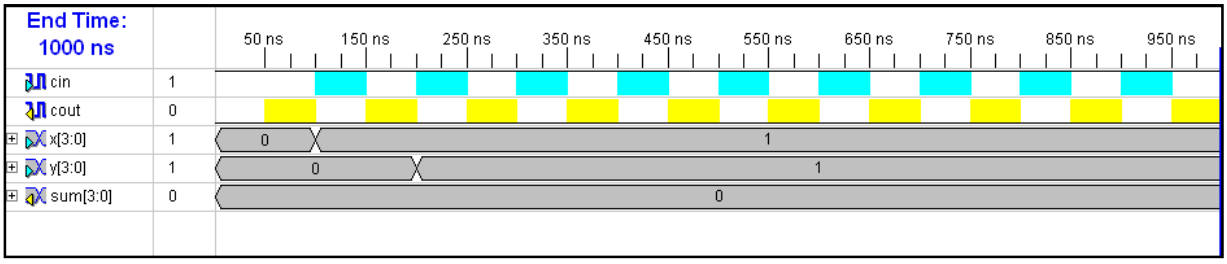


Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder is
   Port ( x: in  STD_LOGIC_VECTOR (3 downto 0);
          y : in  STD_LOGIC_VECTOR (3 downto 0);
          cin : in  STD_LOGIC;
          sum : out  STD_LOGIC_VECTOR (3 downto 0);
          cout : out  STD_LOGIC);
end adder;

architecture Behavioral of adder is

component fulladder
port(
      x,y,cin:in std_logic;
      sum,cout: out std_logic);
end component;
signal c: std_logic_vector(2 downto 0);

begin
a1:fulladder port map(x(0), y(0), cin, sum(0), c(0));
a2:fulladder port map(x(1), y(1), c(0), sum(1), c(1));
a3:fulladder port map(x(2), y(2), c(1), sum(2), c(2));
a4:fulladder port map(x(3), y(3), c(2), sum(3), cout);

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
port(
            x,y,cin:in std_logic;
            sum,cout: out std_logic);
end fulladder;

architecture comb of fulladder is
begin
sum <= x xor y xor cin;
cout <= (x and y) or (cin and (x xor y));
end comb;
```

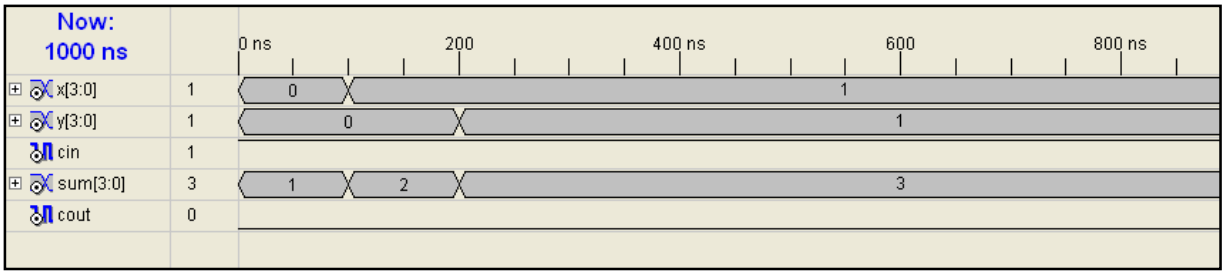
Input Wave form

We give input x=’1’, y=’1’ and cin=’1’in waveform window



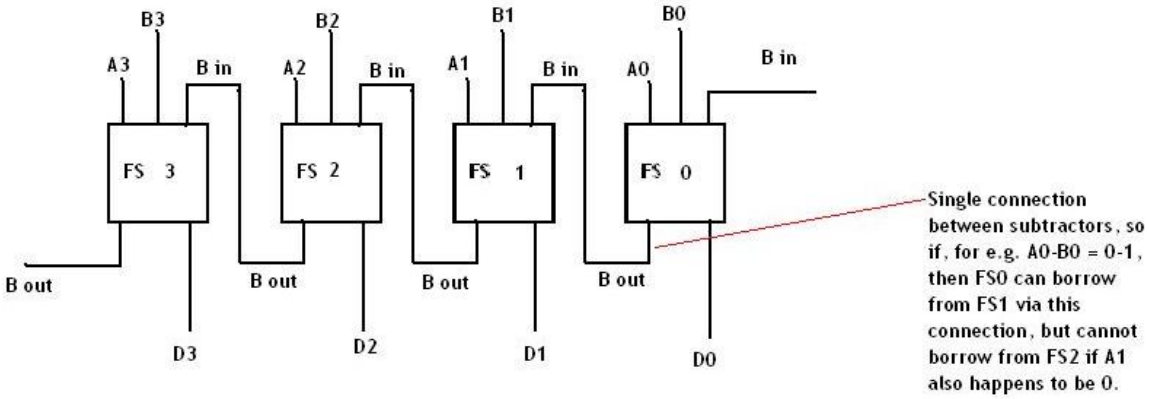
Output Wave form

We give input x=’1’, y=’1’ and cin=’1’ so we get sum=’3’waveform window



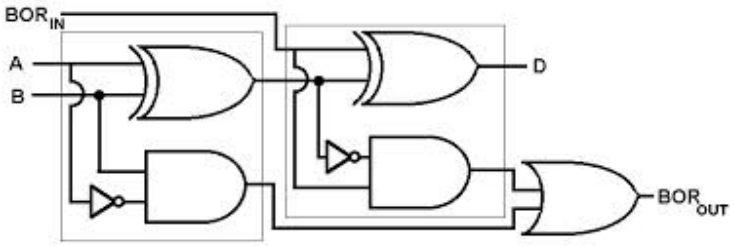
2.2- Subtraction (Program for 4-bit subtraction using arithmetic operator)

It is possible to create a logical circuit using multiple full subtractor to subtract N-bit numbers. Each full subtractor inputs a B_{in}, which is the B_{out} of the previous subtractor . This kind of subtractor is called a ripple-carry subtractor, since each Borrow bit "ripples" to the next full subtractor.

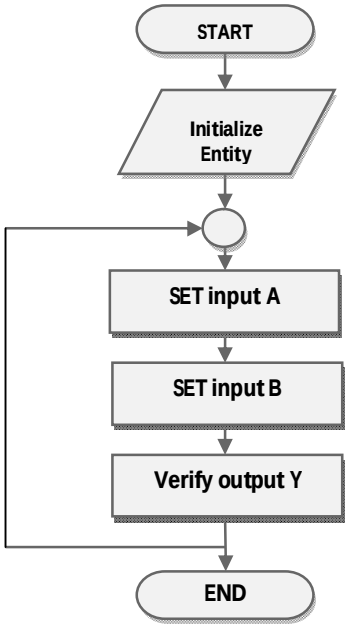


Full Subtractor

A combinational circuit which performs the subtraction of three input bits is called full subtractor. The three input bits include two significant bits and a previous borrow bit.



➢ **Flow Chart**

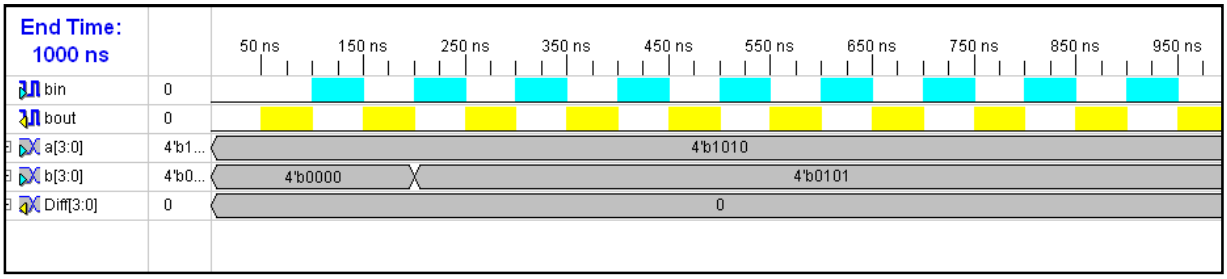

START
Initialize Entity
SET input A
SET input B
Verify output Y
END

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sub is
   Port ( a: in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          bin : in  STD_LOGIC;
          Diff : out  STD_LOGIC_VECTOR (3 downto 0);
          bout : out  STD_LOGIC);
end  sub;

architecture Behavioral of sub is
component fullsub
port(
       a,b,bin:in std_logic;
       Diff,bout: out std_logic);
end component;
signal D: std_logic_vector(2 downto 0);
begin
x1:fullsub port map(a(0), b(0), bin, Diff(0), D(0));
x2:fullsub port map(a(1), b(1), D(0), Diff(1), D(1));
x3:fullsub port map(a(2), b(2), D(1), Diff(2), D(2));
x4:fullsub port map(a(3), b(3), D(2), Diff(3), Bout);

end Behavioral;

library ieee;
use ieee.std_logic_1164.all;

entity fullsub is
port(
       a,b,bin:in std_logic;
       Diff,bout: out std_logic);
end fullsub;

architecture comb of fullsub is
begin
Diff <= a xor b xor bin;
bout <= ((not a )and b) or ((not bin )and (a xor b)) ;

end comb;
```

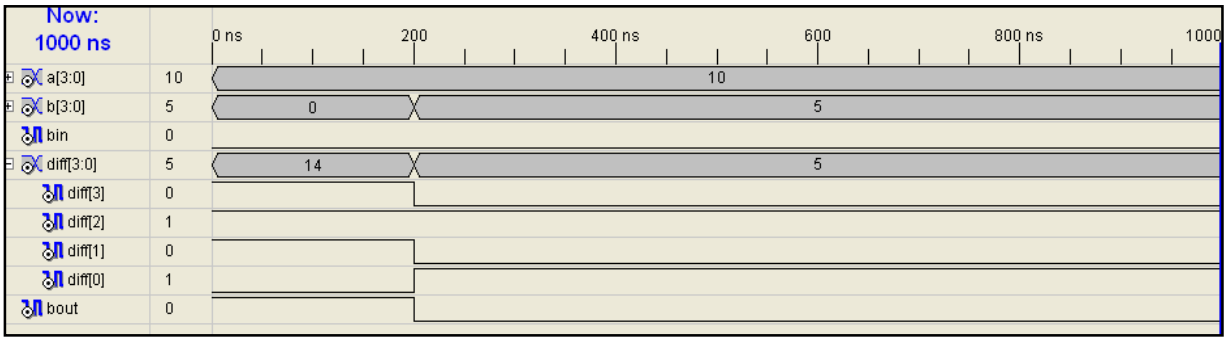
Input Waveform

We give input a=’1010’, b=’0101’ in waveform window



Output Waveform

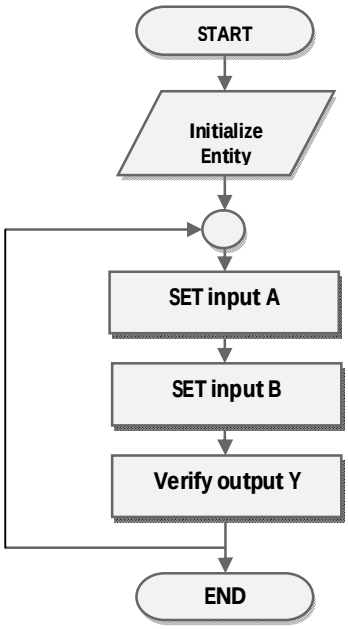
We get output in simulation window(according to that subtraction Operation)



2.3 – Multiplication (Program for simple 4x4 multiplications using arithmetic operator)

Consider the multiplication of two numbers as 4 x4 a * b, where a and b are 4 bit numbers and the output of multiplication is taken in y as 8 bit number.

➢ Flow Chart

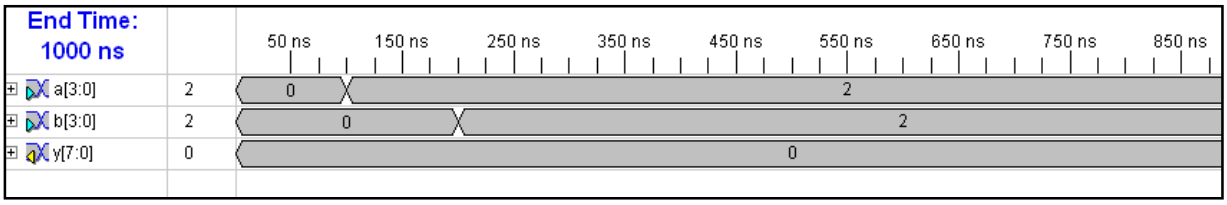


Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity codef is
   Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
          b : in  STD_LOGIC_VECTOR (3 downto 0);
          y : out  STD_LOGIC_VECTOR (7 downto 0));
end codef;

architecture Behavioral of codef is

begin

y <= a * b;
end Behavioral;
```

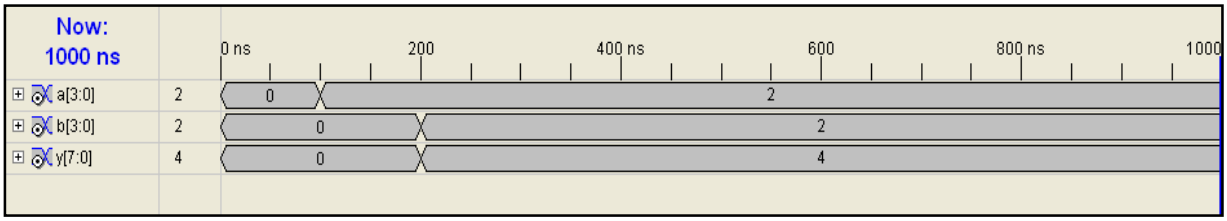
Input Waveform

We give input a =’2’& b =’2’ .



Output Waveform

We give input a =’2’& b =’2’ .so, we get output y=’4’ according to that multiplication operation.



SIMULATION OF VHDL CODE FOR MULTIPLEXER

3

Description

Design and develop a 2 bit multiplexer and port map the same for developing up to 8 bit multiplexer.

Multiplexer

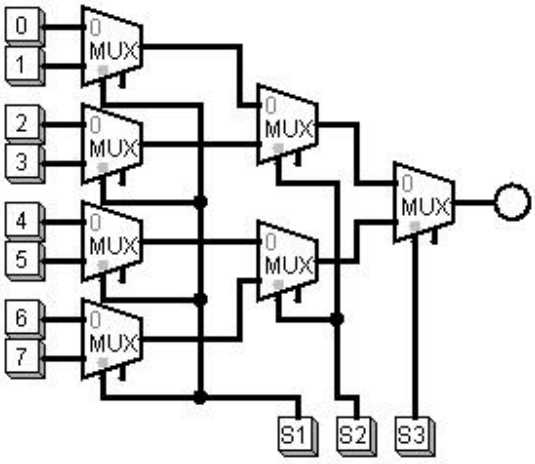
A multiplexer is a combinatorial circuit that is given a certain number (usually a power of two) data inputs, let us say 2^n, and n address inputs used as a binary number to select one of the data inputs. The multiplexer has a single output, which has the same value as the selected data input.

In other words, the multiplexer works like the input selector of a home music system. Only one input is selected at a time, and the selected input is transmitted to the single output. While on the music system, the selection of the input is made manually, the multiplexer chooses its input based on a binary number, the address input.

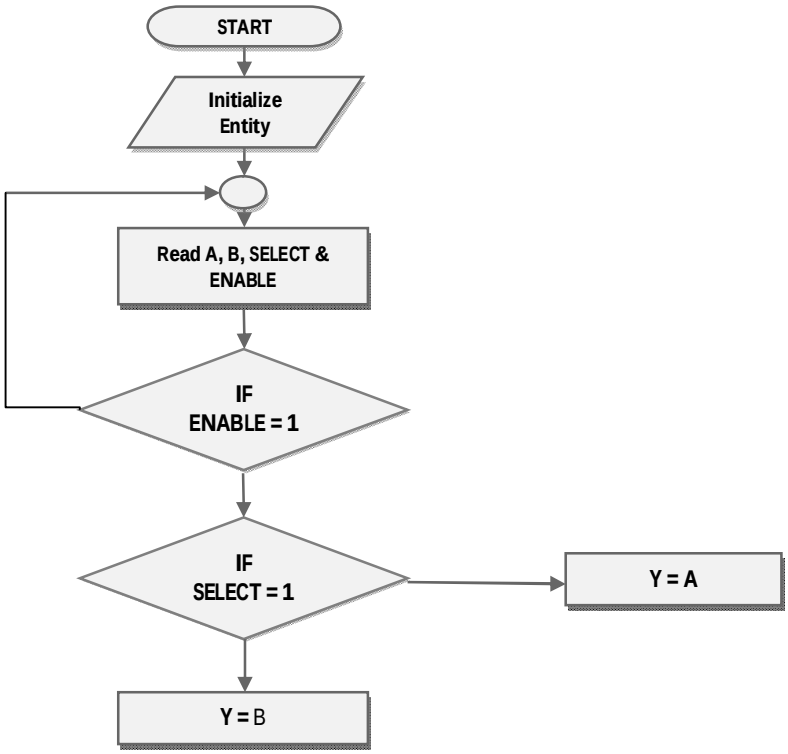
The truth table for a multiplexer is huge for all but the smallest values of n. We therefore use an abbreviated version of the truth table in which some inputs are replaced by `-' to indicate that the input value does not matter.

Here is such an abbreviated truth table for $n = 3$. The full truth table would have $2^{(3 + 23)} = 2048$ rows.

2:1 mux to construct 8:1 mux



Flow chart



Code Listing

2:1 multiplexer program

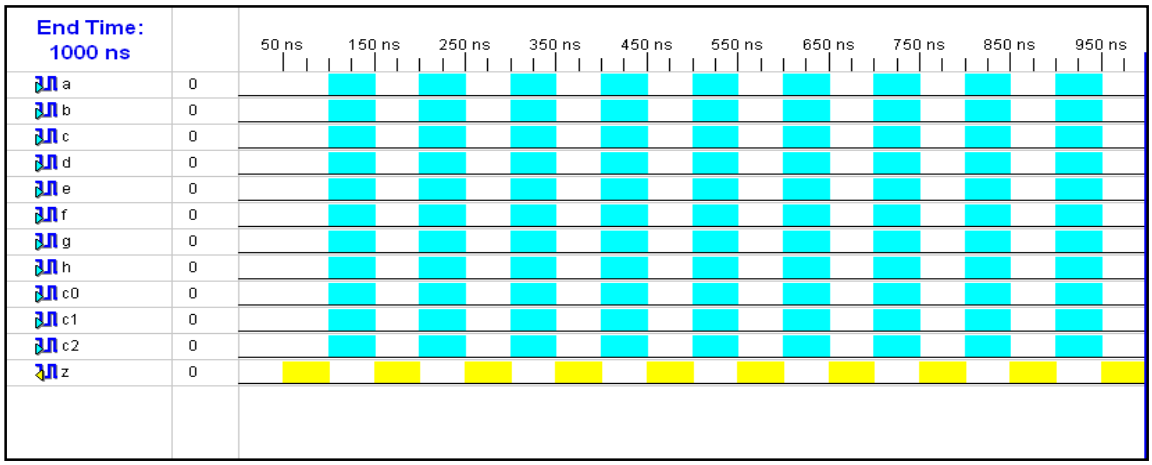
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux is
    Port ( x : in  STD_LOGIC;
           y : in  STD_LOGIC;
           sel : in  STD_LOGIC;
           z : out  STD_LOGIC);
end mux;

architecture Behavioral of mux is
begin
process (x,y,sel)
begin
if(sel='0') then
z<=x;
elsif (sel='1') then
z<=y;
end if;
end process;
```

8:1 mux port map program

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity multiplexer is
    Port (
             a, b,c,d,e,f,g,h : in  STD_LOGIC ;
     c0,c1,c2: in std_logic;
                  z : out  STD_LOGIC );
end multiplexer;
architecture Behavioral of multiplexer is
component mux
port(
 x:in std_logic;
 y: in std_logic;
 sel: in std_logic;
 z: out std_logic
 );
 end component ;
 signal z1,z2,z3,z4,z5,z6 : std_logic :='0';
 begin
      signal z1,z2,z3,z4,z5,z6 : std_logic :='0';
 begin
mux1: mux port map (a, b, c0, z1);
mux2: mux port map (c, d, c0, z2);
mux3: mux port map (e, f, c0, z3);
mux4  mux port map (g, h, c0, z4);
mux5: mux port map (z1, z2, c1, z5);
mux6: mux port map (z3, z4, c1, z6);
mux7: mux port map (z5, z6, c2, z);

end Behavioral;
```

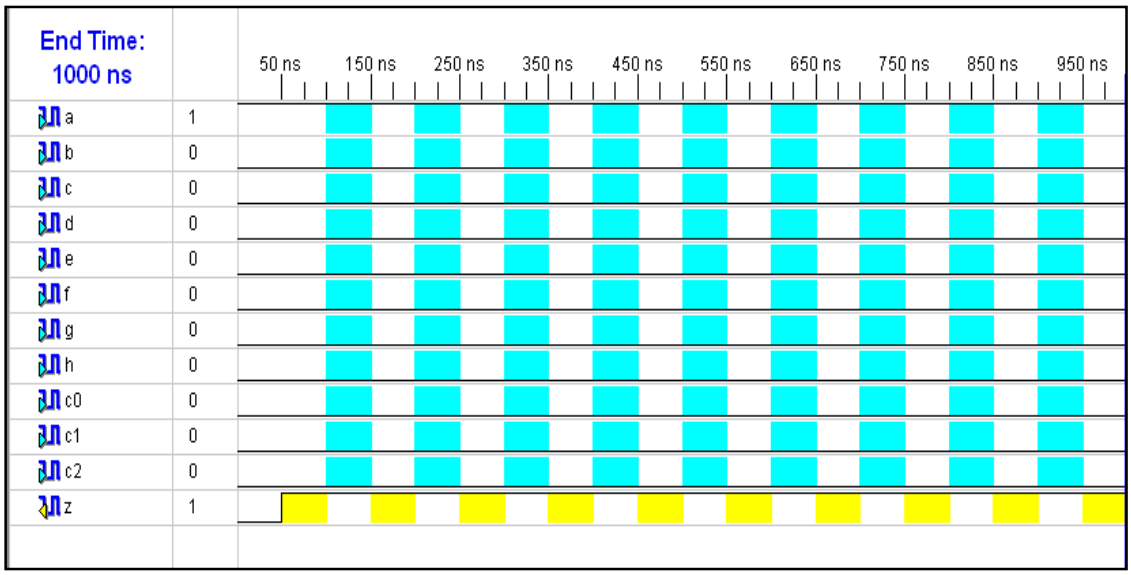
Input Waveform

We give input a=’high’ in waveform window



Output Waveform

We get output in simulation window (y=’high’ according to that multiplexer operation)



SIMULATION OF VHDL CODE FOR DEMULTIPLEXER

3

Description

Design and develop an 8 output demultiplexer using truth table.

Demultiplexer

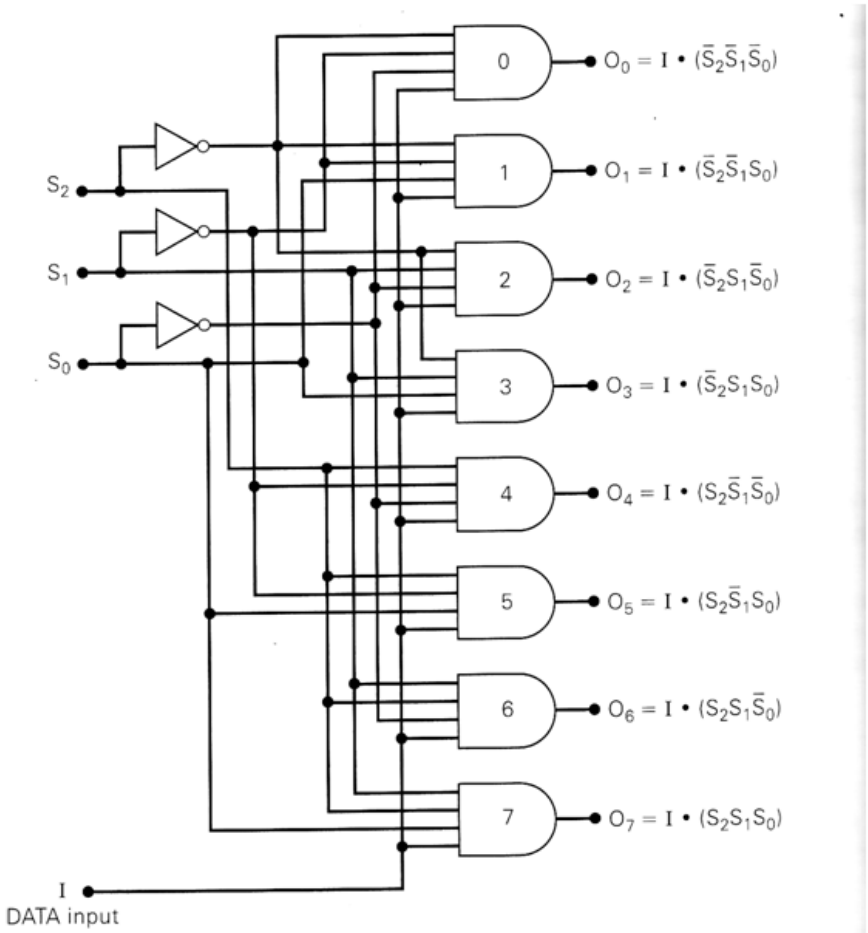
The demultiplexer is the inverse of the multiplexer, in that it takes a single data input and n address inputs. It has 2^n outputs. The address input determine which data output is going to have the same value as the data input. The other data outputs will have the value 0.

Here is an abbreviated truth table for the demultiplexer. We could have given the full table since it has only 16 rows, but we will use the same convention as for the multiplexer where we abbreviated the values of the data inputs.

S2	S1	S0	E	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Here is one possible circuit diagram for the demultiplexer:

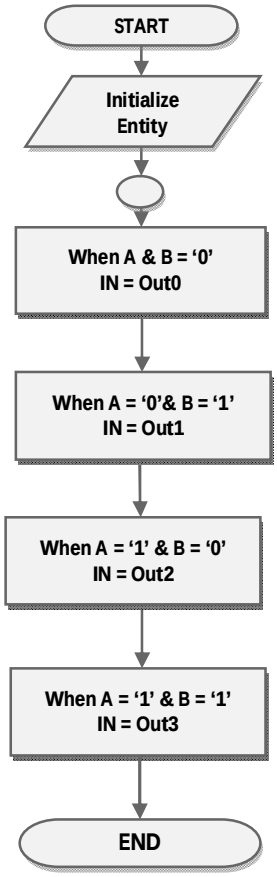
In this program a 1 x 8 de-multiplexer have two 1- bit inputs, a 3-bit select line and a 8- bit output. Additional control signals may be added such as enable. The output of the multiplexers depends on the level of the select line.



SELECT code			OUTPUTS							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

Note: I is the data input

Flow Chart

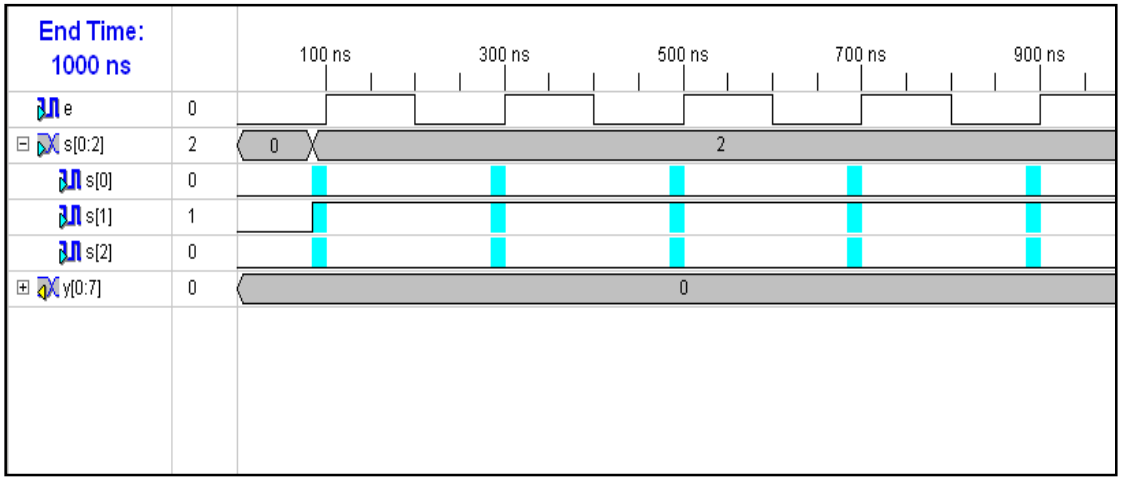

START
Initialize
Entity
When A & B = '0'
IN = Out0
When A = '0'& B = '1'
IN = Out1
When A = '1' & B = '0'
IN = Out2
When A = '1' & B = '1'
IN = Out3
END

Code Listing

```
library ieee;
use ieee.std_logic_1164.all;

entity demultiplexer is
port(I : in std_logic;
     s : in std_logic_vector(0 to 2);
     O : out std_logic_vector(0 to 7));
end demultiplexer;

architecture data of demultiplexer is

begin
    O(0) <= (I and not s(0) and not s(1) and not s(2));
    O(1) <= (I and s(0) and not s(1) and not s(2));
    O(2) <= (I and not s(0) and s(1) and not s(2));
    O(3) <= (I and s(0) and s(1) and not s(2));
    O(4) <= (I and not s(0) and not s(1) and s(2));
    O(5) <= (I and s(0) and not s(1) and s(2));
    O(6) <= (I and not s(0) and s(1) and s(2));
    O(7) <= (I and s(0) and s(1) and s(2));
end data;
```

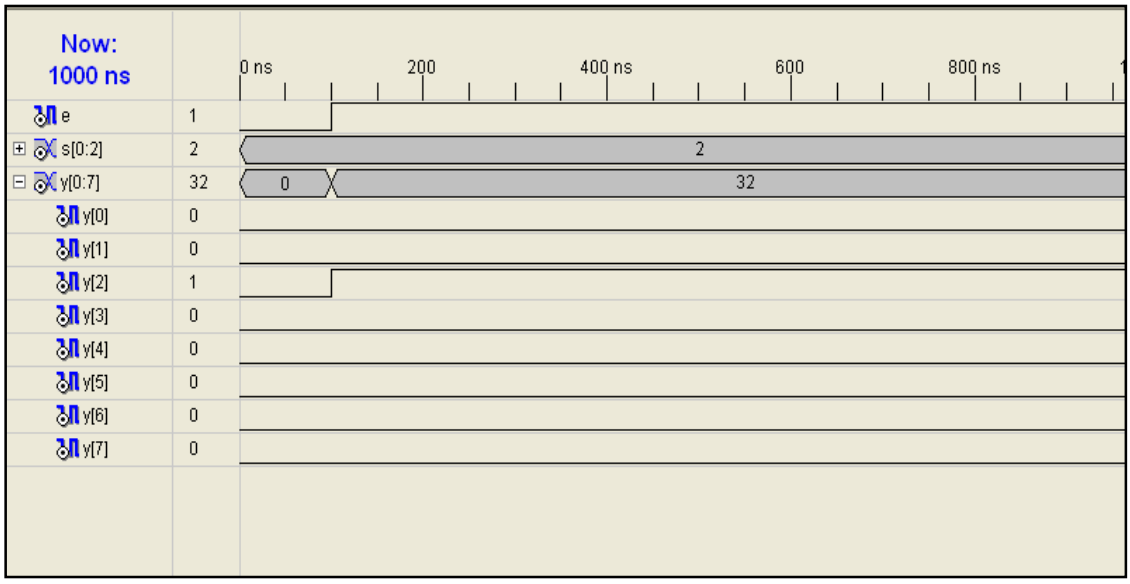
Input Waveform

We put enable as high in all condition and here we give input=’010’ in selection line(according to that selection line we get output)



Output Waveform

We get output in simulation window(according to that selection line)



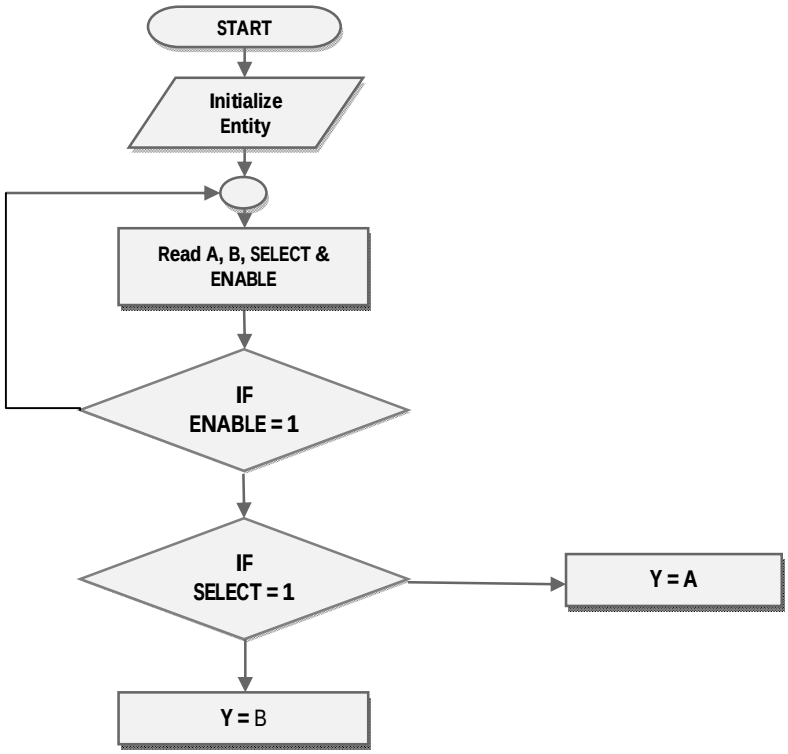
VHDL IMPLEMENTATION OF MULTIPLEXER

5

Description

Describe the code for a multiplexer and implement it in FPGA kit in which switches are connected for select input and for data inputs a LED is connected to the output.

Flow Chart



PIN Description:

I/O Pins	A	B	C	D	SEL0	SEL1	OUTPUT
FPGA LOC	P23	P26	P27	P28	P93	P105	P53

Code Listing

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mux_gate is

port (A,B,C,D: in STD_LOGIC;
      SEL     : in STD_LOGIC_vector (1 downto 0);
      Output    : out STD_LOGIC);
end mux_gate;

architecture behav of mux_gate is
begin
process (SEL,A,B,C,D)
begin
case SEL is
when "00"  => Output <= A;
when "01"  => Output <= B;
when "10"  => Output <= C;
when "11"  => Output <= D;
when others => null;
end case;
end process;
end behave;
```

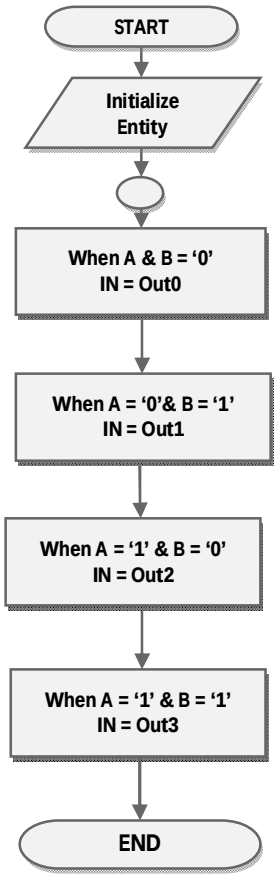
VHDL IMPLEMENTATION OF DEMULTIPLEXER

6

Description

Switches are connected for select inputs and a data input, Eight LEDs are connected to the output of the circuit.

Flow Chart



PIN Description:

I/O PINS	E	S(0)	S(1)	S(2)	Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)	Y(7)
FPGA LOC	P23	P33	P93	P105	P53	P50	P24	P23	P21	P20	P18	P17

Code Listing

```
library ieee;
use ieee.std_logic_1164.all;

entity demultiplexer is
port(e : in std_logic;
     s : in std_logic_vector(0 to 2);
     y : out std_logic_vector(0 to 7));
end demultiplexer;

architecture data of demultiplexer is

begin
    y(0) <= (e and not s(0) and not s(1) and not s(2));
    y(1) <= (e and s(0) and not s(1) and not s(2));
    y(2) <= (e and not s(0) and s(1) and not s(2));
    y(3) <= (e and s(0) and s(1) and not s(2));
    y(4) <= (e and not s(0) and not s(1) and s(2));
    y(5) <= (e and s(0) and not s(1) and s(2));
    y(6) <= (e and not s(0) and s(1) and s(2));
    y(7) <= (e and s(0) and s(1) and s(2));
end data;
```

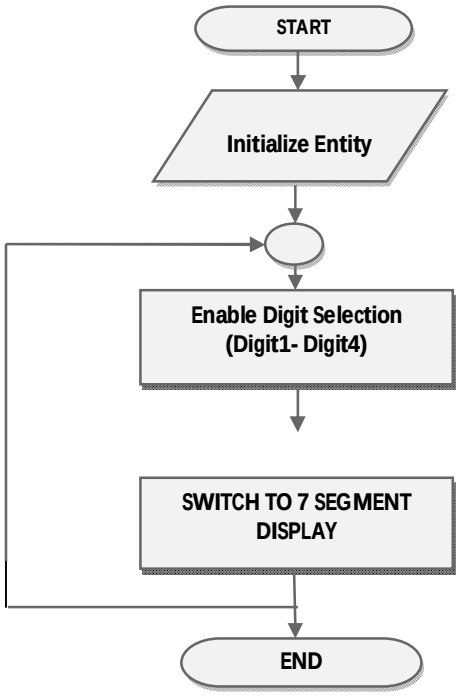
VHDL IMPLEMENTATION OF 7 SEGMENT DECODER

7

Description

Develop Boolean expression for 4 input variables and 7 output variables. Design and develop seven segment decoder in VHDL for 7 equations. A seven segment display is connected to the output of the circuit. Four switches are connected to the input. The 4 bit input is decoded to 7 segment equivalent.

Flow Chart



PIN Description:

I/O PINS	CLK	I0	I1	I2	I3	A	B	C	D	E	F	G	SEL0	SEL1	SEL2	SEL3
FPGA LOC	P55	P32	P33	P93	P105	P129	P130	P131	P132	P135	P137	P140	P124	P125	P127	P128

Look Up Table:

I3	I2	I1	I0	G	F	E	D	C	B	A
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	0	0	0	1	1	0	0	1
0	1	0	1	0	0	1	0	0	1	0
0	1	1	0	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1	0	0	0

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if
instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity seven_seg is
    Port ( I3,I2,I1,I0 : in  STD_LOGIC;
                  sel : out std_logic_vector (3 downto 0);
           A,B,C,D,E,F,G : out  STD_LOGIC);
end seven_seg;

architecture Behavioral of seven_seg is
SIGNAL X0,X1,X2,X3,X4,X5,X6,X7,X8,X9:STD_LOGIC;
begin
sel <= "11111111";
X0<= (NOT I3 AND NOT I2 AND NOT I1 AND NOT I0);
X1<= (NOT I3 AND NOT I2 AND NOT I1 AND I0);
X2<= (NOT I3 AND NOT I2 AND I1 AND NOT I0);
X3<= (NOT I3 AND NOT I2 AND I1 AND I0);
X4<= (NOT I3 AND I2 AND NOT I1 AND NOT I0);
X5<= (NOT I3 AND I2 AND NOT I1 AND I0);
X6<= (NOT I3 AND I2 AND I1 AND NOT I0);
X7<= (NOT I3 AND I2 AND I1 AND I0);
X8<= (I3 AND NOT I2 AND NOT I1 AND NOT I0);
X9<= (I3 AND NOT I2 AND NOT I1 AND I0);

A <= X1 OR X4;
B <= X5 OR X6;
C <= X2;
D <= X1 OR X4 OR X7 OR X9;
E <= X1 OR X3 OR X4 OR X5 OR X7 OR X9;
F <= X1 OR X2 OR X3 OR X7;
G <= X0 OR X1 OR X7;
end Behavioral;
```

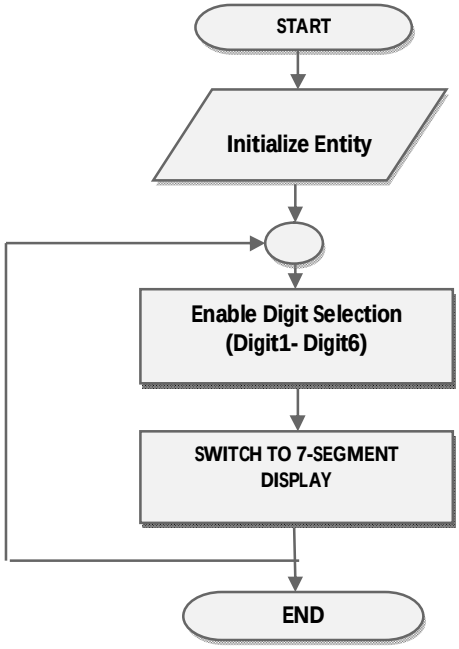
VHDL IMPLEMENTATION OF 7 SEGMENT DECODER BY LUT

8

Description

Develop a 7 segment decoder using Look up table. Describe the seven segment decoder in VHDL using developed Look up table. A seven segment display is connected to the output of the circuit. Four switches are connected to the input. The 4 bit input is decoded into 7 segment equivalent.

Flow Chart



Look Up Table:

I3	I2	I1	I0	G	F	E	D	C	B	A
0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	0	0	0	1	1	0	0	1
0	1	0	1	0	0	1	0	0	1	0
0	1	1	0	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1	0	0	0

PIN Description:

I/O PINS	CLK	I0	I1	I2	I3	A	B	C	D	E	F	G	SEL0	SEL1	SEL2	SEL3
FPGA LOC	P55	P32	P33	P93	P105	P129	P130	P131	P132	P135	P137	P140	P124	P125	P127	P128

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SEVEN_SEG_LUT is
    Port ( I : in  STD_LOGIC_VECTOR (3 downto 0);
           SEL : out  STD_LOGIC_VECTOR (7 downto 0);
           Y : out  STD_LOGIC_VECTOR (6 downto 0));
end SEVEN_SEG_LUT;

architecture Behavioral of SEVEN_SEG_LUT is

begin

SEL <= "1111";
process (I)
BEGIN
case  I is
when "0000"=> Y <="1000000";  -- '0'
when "0001"=> Y <="1111001";  -- '1'
when "0010"=> Y <="0100100";  -- '2'
when "0011"=> Y <="0110000";  -- '3'
when "0100"=> Y <="0011001";  -- '4'
when "0101"=> Y <="0010010";  -- '5'
when "0110"=> Y <="0000010";  -- '6'
when "0111"=> Y <="1111000";  -- '7'
when "1000"=> Y <="0000000";  -- '8'
when "1001"=> Y <="0011000";  -- '9'
when others=> Y <="1111111";
end case;

end process;

end Behavioral;
```

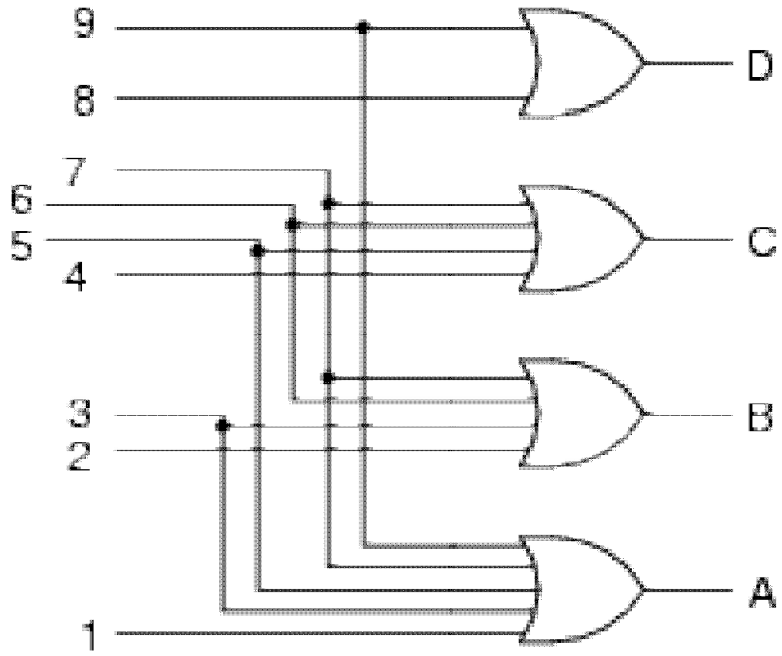
VHDL IMPLEMENTATION OF ENCODER

9

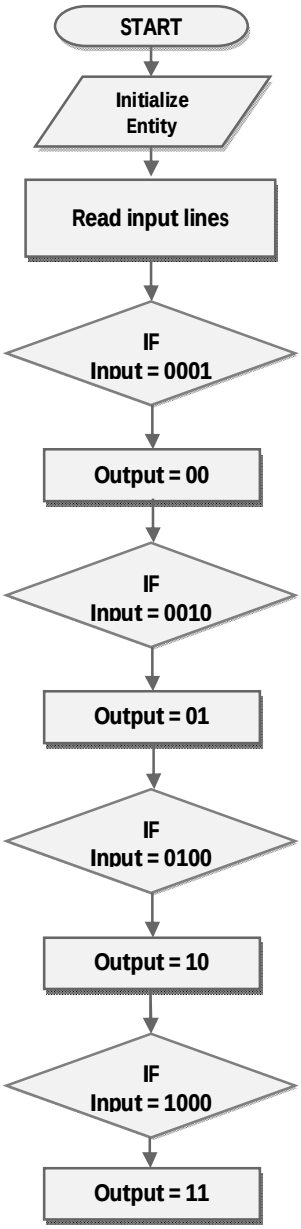
Encoder:

Design and develop HDL code for decimal (Octal) to BCD encoder. There will be10 input switches (or 8 switches) and 4 LEDs in the FPGA kit. The input given from switches and it is noted that any one of the switch is active. The binary equivalent for the corresponding input switch will be glowing in the LED as output

Logical Diagram



Flow Chart


START
Initialize Entity
Read input lines
IF Input = 0001
Output = 00
IF Input = 0010
Output = 01
IF Input = 0100
Output = 10
IF Input = 1000
Output = 11

PIN Description:

I/O PINS	X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)	X(9)	A	B	C	D
FPGA LOC	P105	P93	P33	P32	P31	P30	P28	P27	P26	P53	P50	P24	P23

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( x : in std_logic_vector(9 downto 1);
       a,b,c,d : out std_logic;);
end first;
architecture Behavioral of first is

begin
a<= x1 or x3 or x or x7 or x9;
b<= x2 or x3 or x6 or x7 ;
c <= x4 or x5 or x6 or x7 ;
d<= x9 or x8 ;
end Behavioral;
```

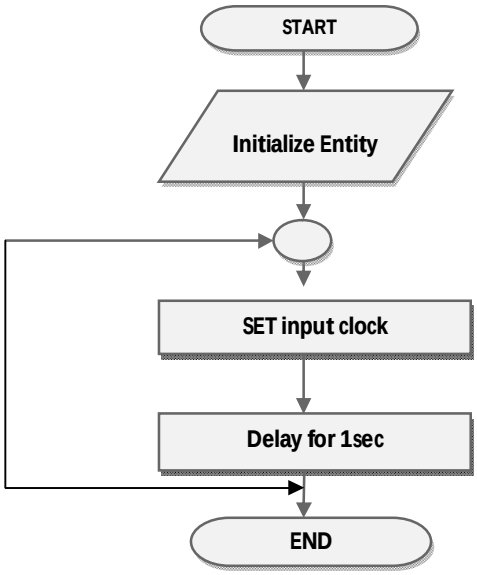
SIMULATION OF VHDL CODE FOR DELAY

10

Description

Develop a VHDL code for making a delayed output for 1second or 2 seconds by assuming clock frequency provided in the FPGA Kit. Simulate same code to get a delayed waveform.

Flow Chart

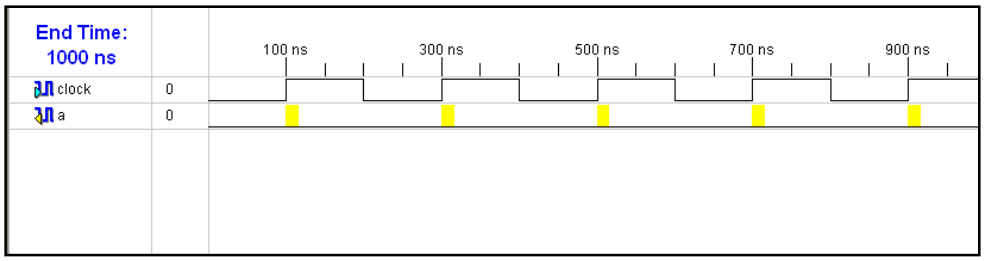


Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clock : in std_logic;
       a     : out std_logic
             );
end first;

architecture Behavioral of first is

begin
process(clock)
variable i : integer := 0;
begin
if clock'event and clock = '1' then
if i <= 50000000 then
i := i + 1;
a <= '1';
elsif i > 50000000 and i < 100000000 then
i := i + 1;
a <= '0';
elsif i = 100000000 then
i := 0;
end if;
end if;
end process;
end Behavioral;
```

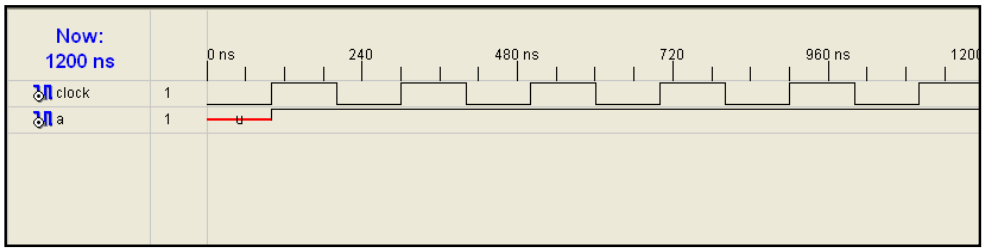
Input Waveform

We give input in waveform window (clk=’1’)



Output Waveform

We get output in simulation waveform (a=’delayed signal’)



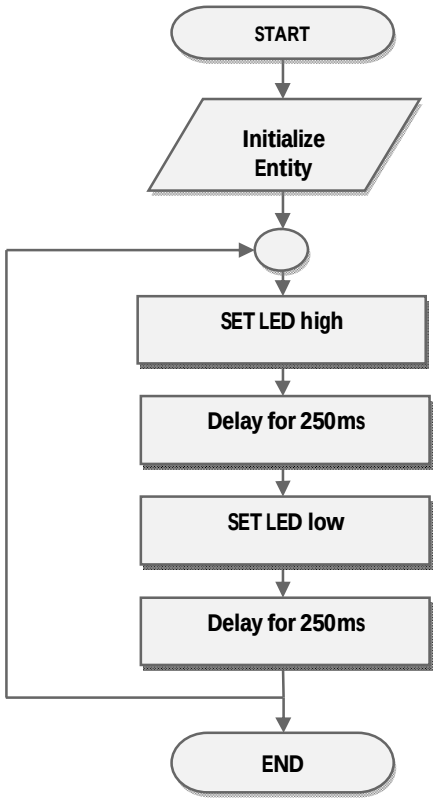
VHDL IMPLEMENTATION FOR BLINKING A LED

11

Description

Develop a VHDL Code for delay and verify by simulating it. This delay output is connected to LED. Delay is adjusted such away LED blinks for every 1 or 2 seconds.

Flow Chart



PIN Description:

I/O PINS	CLK	LED
FPGA LOC	P55	P53

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity first is
port ( clock : in std_logic;

            LED     : out std_logic);
end first;

architecture Behavioral of first is

begin
process(clock)
variable i : integer := 0;
begin
if clock'event and clock = '1' then
if i <= 50000000 then
i := i + 1;
LED <= '0';
elsif i > 50000000 and i < 100000000 then
i := i + 1;
LED <= '1';
elsif i = 100000000 then
i := 0;
end if;
end if;
end process;
end Behavioral;
```

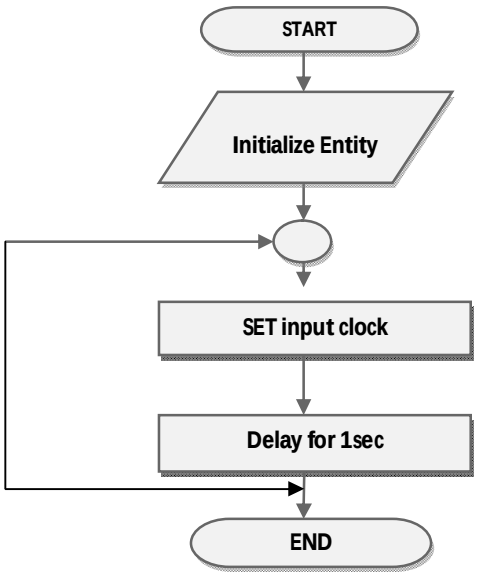
SIMULATE A VHDL TEST BENCH CODE FOR TESTING A GATE

12

Description

Develop a VHDL test bench code for testing any one of the simple gate. Simulate the test bench code in the HDL software.

Flow Chart



Code Listing (AND gate program)

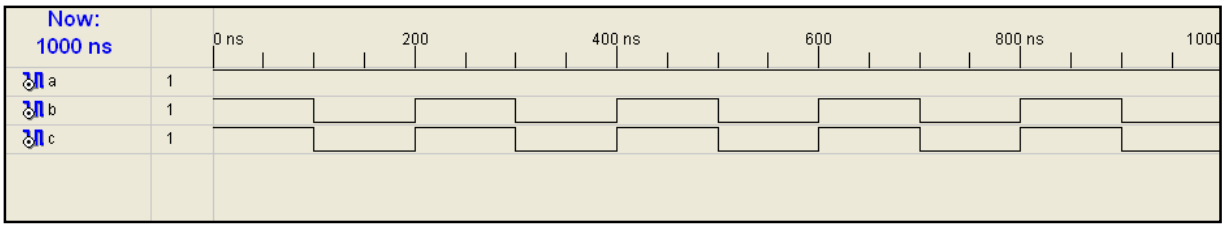
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity gate is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : out  STD_LOGIC);
end gate;

architecture Behavioral of gate is

begin
c <= a and  b;

end Behavioral;
```

Testbench code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY code_vhd IS
END code_vhd;

ARCHITECTURE behavior OF code_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT cd
	PORT(
		a : IN std_logic;
		b : IN std_logic;
		c : OUT std_logic
		);
	END COMPONENT;

	--Inputs
	SIGNAL a :  std_logic := '0';
	SIGNAL b :  std_logic := '0';

	--Outputs
	SIGNAL c :  std_logic;

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: cd PORT MAP(
		a => a,
		b => b,
		c => c
	);

	tb : PROCESS
	BEGIN
		a<='1';
		b<='1';
		-- Wait 100 ns for global reset to finish
		wait for 100 ns;
		a<='1';
		b<='0';
		-- Place stimulus here

		wait for 100 ns; -- will wait forever
	END PROCESS;

END;
```

Output Waveform

We get output in simulation window (c=’1’ according to that gate operation)



\

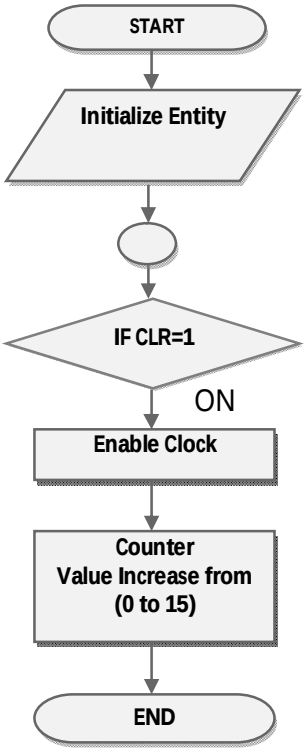
VHDL IMPLEMENTATION FOR BLINKING A ARRAY OF LEDS

13

Description

Design and develop a VHDL Code for 4 bit binary up counter. Four LEDs are connected at the output of the counter. The counter should up for every one seconds.

Flow Chart



PIN Description:

I/O PINS	CLOCK	RST	Q(3)	Q(2)	Q(1)	Q(0)
FPGA LOC	P55	P23	P23	P24	P30	P33

Code Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clock : in std_logic;
  rst:in std_logic;
      q     : out std_logic_vector(3 downto 0)
       );
end first;

architecture Behavioral of first is
signal tmp: std_logic_vector(3 downto 0):="0000";
begin
process(clock ,rst)
variable i : integer := 0;
begin
if (rst='1') then
tmp <= "0000";
elsif clock'event and clock = '1' then
      if i <= 50000000 then
      i := i + 1;
      elsif i = 50000001 then
      i := 0;
      tmp <= tmp+1;
end if;
end if;
end process;
q<= tmp;
end Behavioral;
```

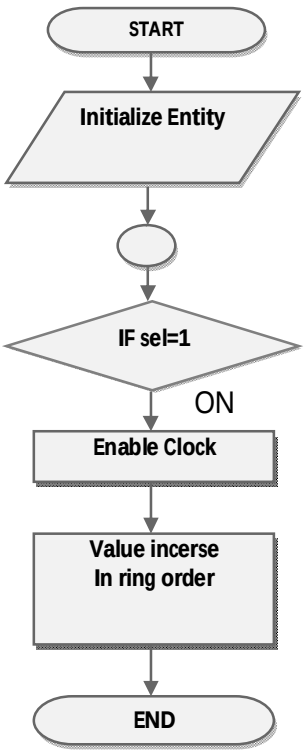
VHDL IMPLEMENTATION OF A SPELLER WITH AN ARRAY OF LEDS

14

Description

Design and develop VHDL Code for a 5 bit Johnson ring counter 4 bit The LEDs are connected at the output of the counter. The speller should work for every one seconds.

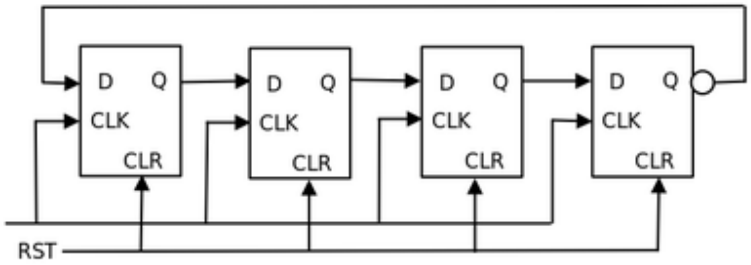
Flow Chart



PIN Description:

I/O PINS	CLOCK	RST	Q(4)	Q(3)	Q(2)	Q(1)	Q(0)
FPGA LOC	P55	P23	P21	P23	P24	P30	P33

Johnson Ring Counter



CODE Listing

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity first is
port ( clk : in std_logic;
       Reset: in std_logic;
       output : out std_logic_vector(4 downto 0));
end first;

architecture Behavioral of first is
signal temp: std_logic_vector(4 downto 0):=”00000”;
begin
process(clk, Reset)
variable i,k : integer := 0;
begin
if Reset = ‘1’ then
temp <= “00000”;
elsif clk'event and clk = '1' then
if i < 50000000 then
i := i + 1;
elsif i = 50000000 then
temp(1) <= temp(0);
temp(2) <= temp(1);
temp(3) <= temp(2);
temp(4) <= temp(3);
temp(0) <= not temp(4);
i:=0;
end if;
end if;
end process;
Reset <= temp;
End Behavioral;
```

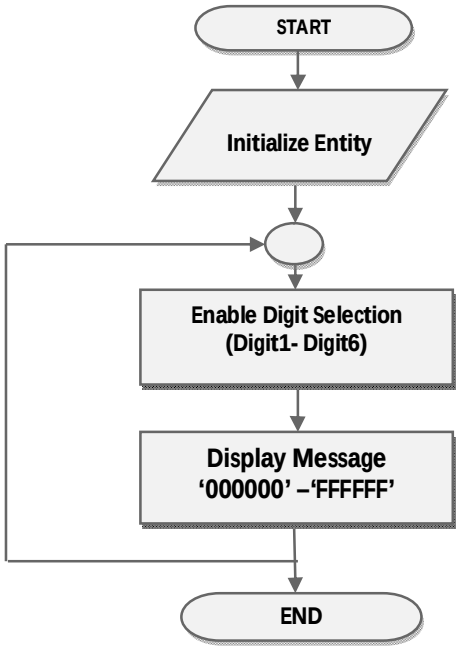
VHDL IMPLEMENTATION OF 7 SEGMENT DISPLAY

15

Description

Design and develop a seven segment decoder in VHDL. Design and develop a 4 bit BCD counter, the output of the counter is given to seven segment decoder. A seven segment display is connected to the output of the decoder. The display shows 0,1, 2.. 9 for every one second

Flow Chart



PIN Description:

I/O PINS	CLK	Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)	Y(7)	SEL0	SEL1	SEL2	SEL3
FPGA LOC	P55	P129	P130	P131	P132	P135	P137	P140	P141	P124	P125	P127	P128

Code Lisitng

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity object_co  is

port(clk    : in  std_logic;
     y      : out std_logic_vector(7 downto 0);
      sel     : out std_logic_vector(7 downto 0)
     );
end object_co;

architecture beav of object_co  is
type state is
(state0,state1,state2,state3,state4,state5,state6,state7,state8,
state9);
signal next_state,ps: state := state0;
begin
sel <= "11111111";
process(clk,next_state)
variable i : integer := 0 ;
begin
if clk'event and clk = '1' then
if i <= 100000000 then
i := i + 1;
elsif i > 100000000 then
i := 0 ;
next_state <= ps ;
end if;
if next_state = state0 then
y <= x"c0" ;
ps <= state1;
elsif next_state = state1 then
y <= x"f9";
ps <= state2;
elsif next_state = state2 then
y <= X"a4";
ps <= state3;
elsif next_state = state3 then
y <= X"b0";
```

```
ps <= state4;
elsif next_state = state4 then
y <= X"99";
ps <= state5;
elsif next_state = state5 then
y <= X"92";
ps <= state6;
elsif next_state = state6 then
y <= X"82";
ps <= state7;
elsif next_state = state7 then
y <= X"f8";
ps <= state8;
elsif next_state = state8 then
y <= X"80";
ps <= state9;
elsif next_state = state9 then
y <= X"98";
ps <= state0;
end if;
end if;
end process;
end beav;
```

Getting Started with Xilinx ISE

(Tutorial)

9 - Getting Started with Xilinx ISE

After installing **Xilinx ISE 8.1 i** software, go to Start menu

Start ➔Programs ➔ Xilinx ISE 8.1 i ➔ Project Navigator (or) double click 

desktop icon (refer **Figure-1**). A Window shown in **Figure-2** will appear.

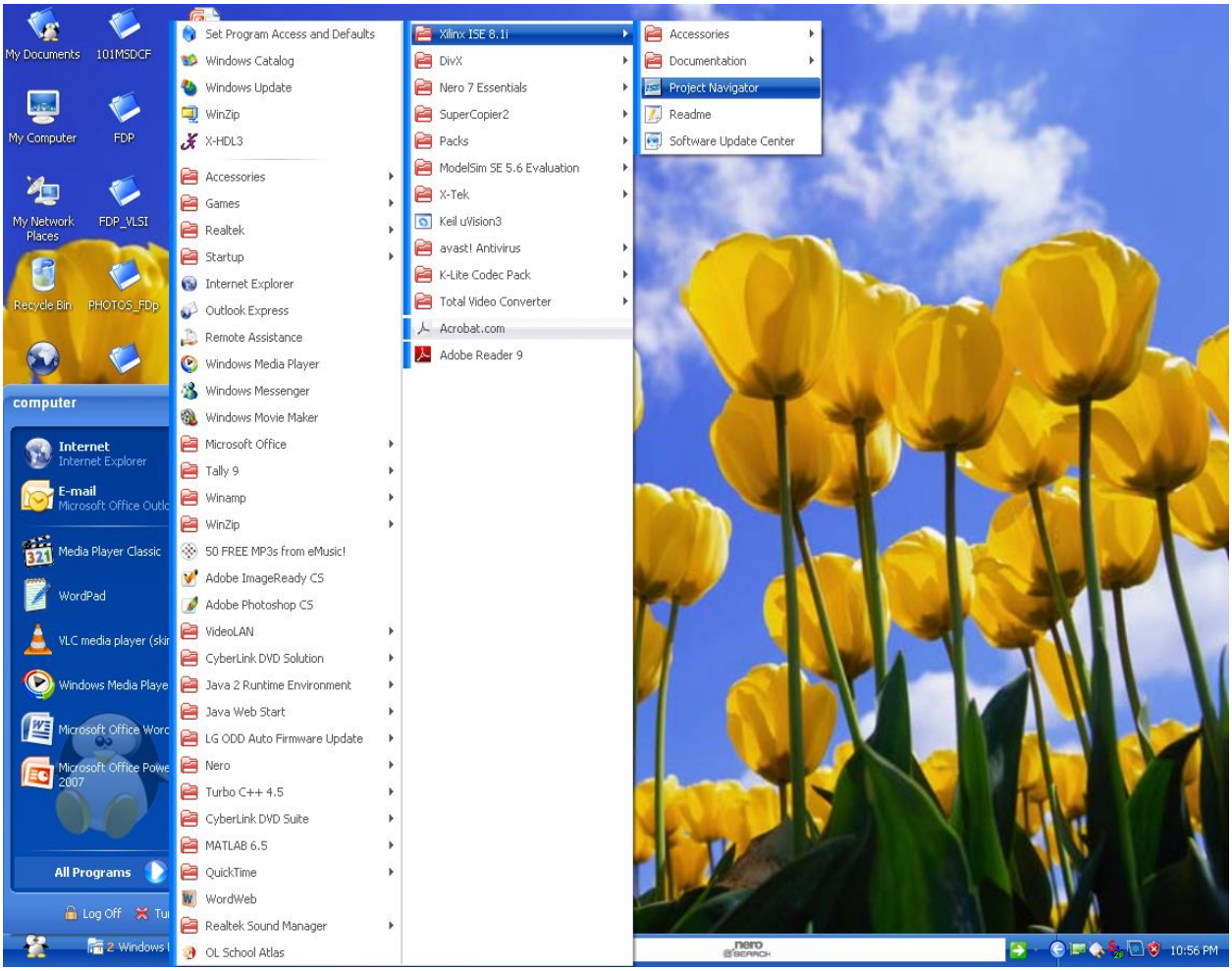


FIGURE - 1

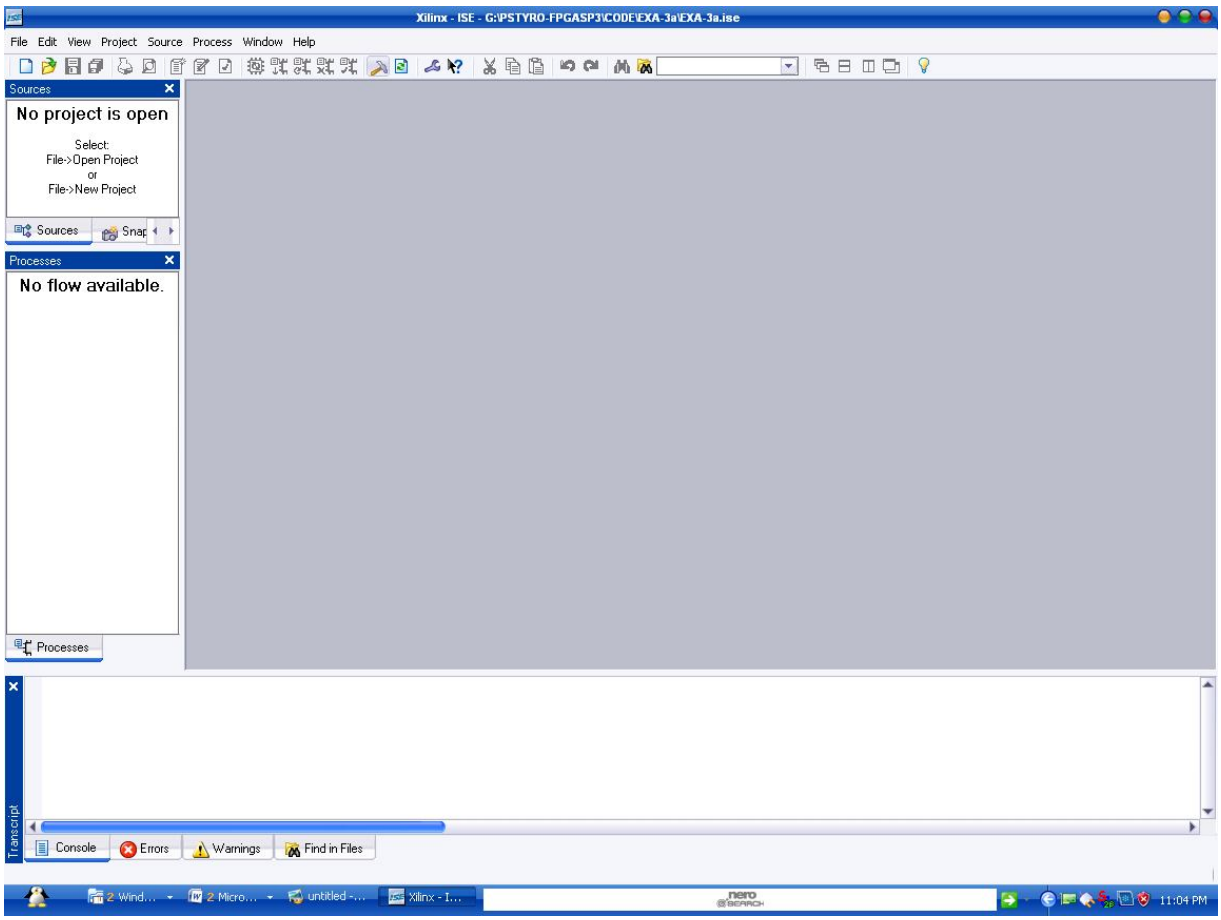


FIGURE - 2

Create a New Project

Create a new ISE project which will target the FPGA device on the Spartan-3 TYRO/PRIMER board.

To create a new project:

1. Select **File ➔ New Project...** The New Project Wizard appears.(refer **figure 3**)

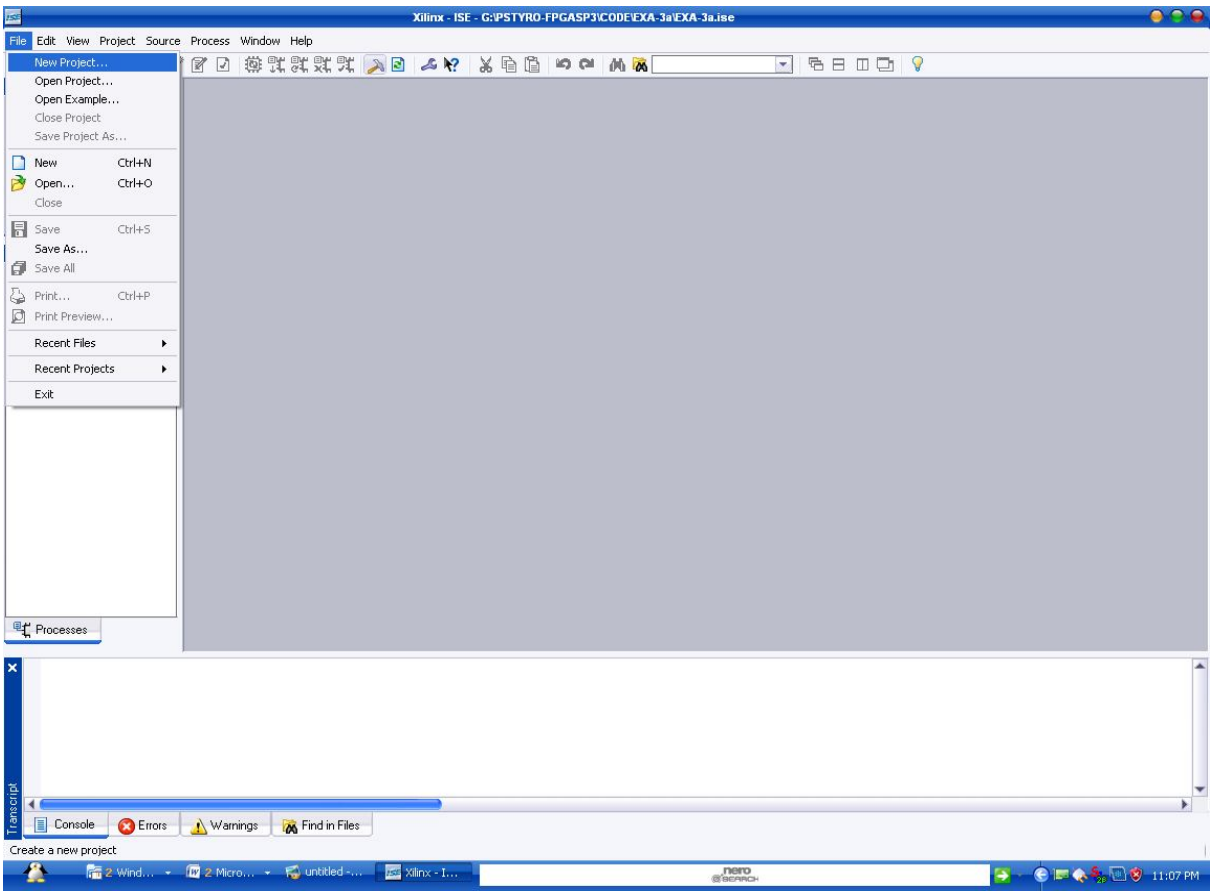


FIGURE 3

2. **New Project wizard** window shown in **Figure -4** will appear. In the **Name** field, enter your project name and enter the location where you want to create the project in the **Location** field (NOTE: don't use c drive or desktop). In the **Top-Level Module** select **HDL** and click **Next (**refer **Figure-5)**.

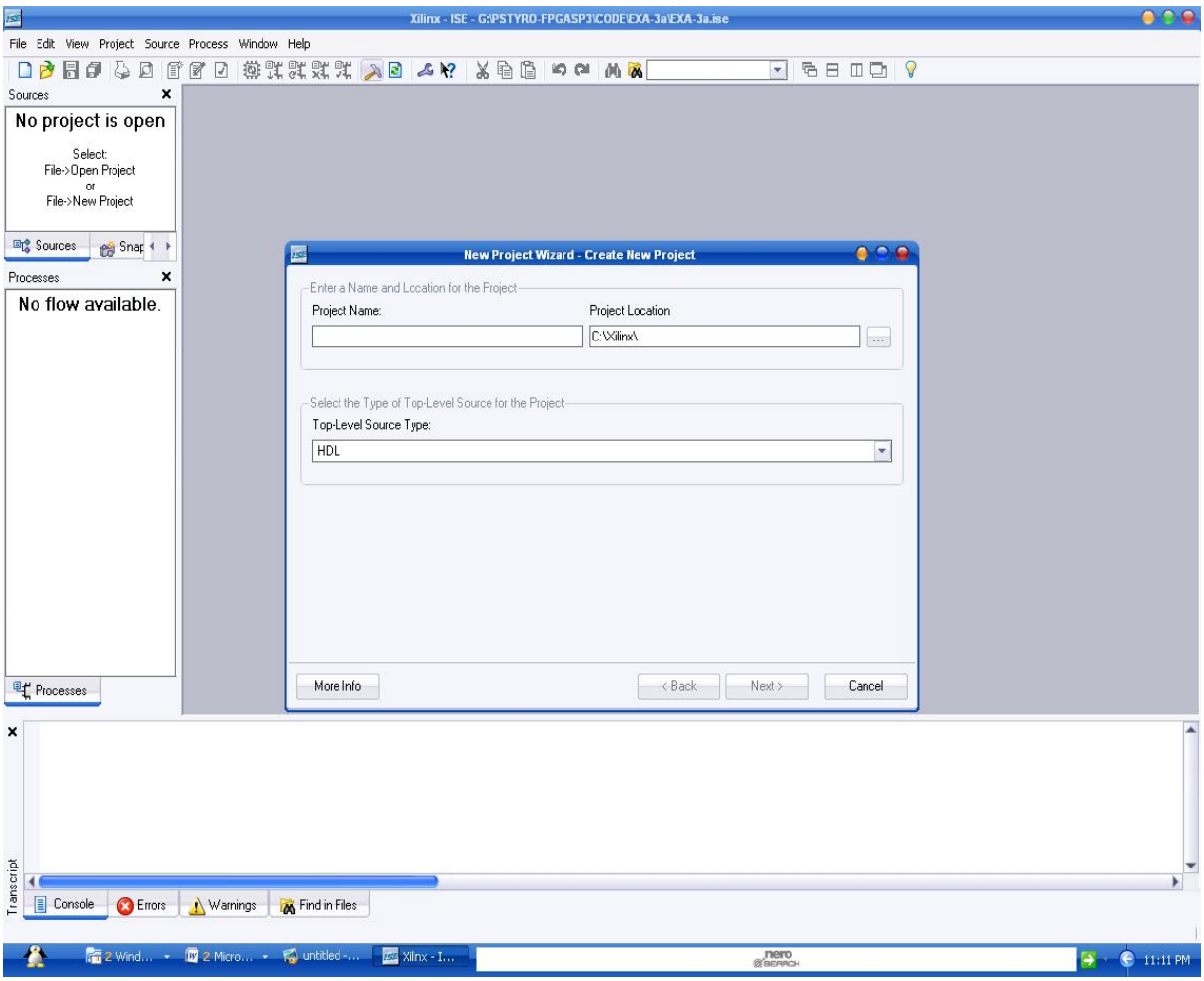

Xilinx - ISE - G:\PSTYRO-FPGASP3\CODE\EXA-3a\EXA-3a.ise
File Edit View Project Source Process Window Help
Sources
No project is open
Select:
File->Open Project
or
File->New Project
Sources
Processes
No flow available.
Processes
New Project Wizard - Create New Project
Enter a Name and Location for the Project
Project Name:
Project Location
C:\Xilinx\
Select the Type of Top-Level Source for the Project
Top-Level Source Type:
HDL
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
11:11 PM

FIGURE 4

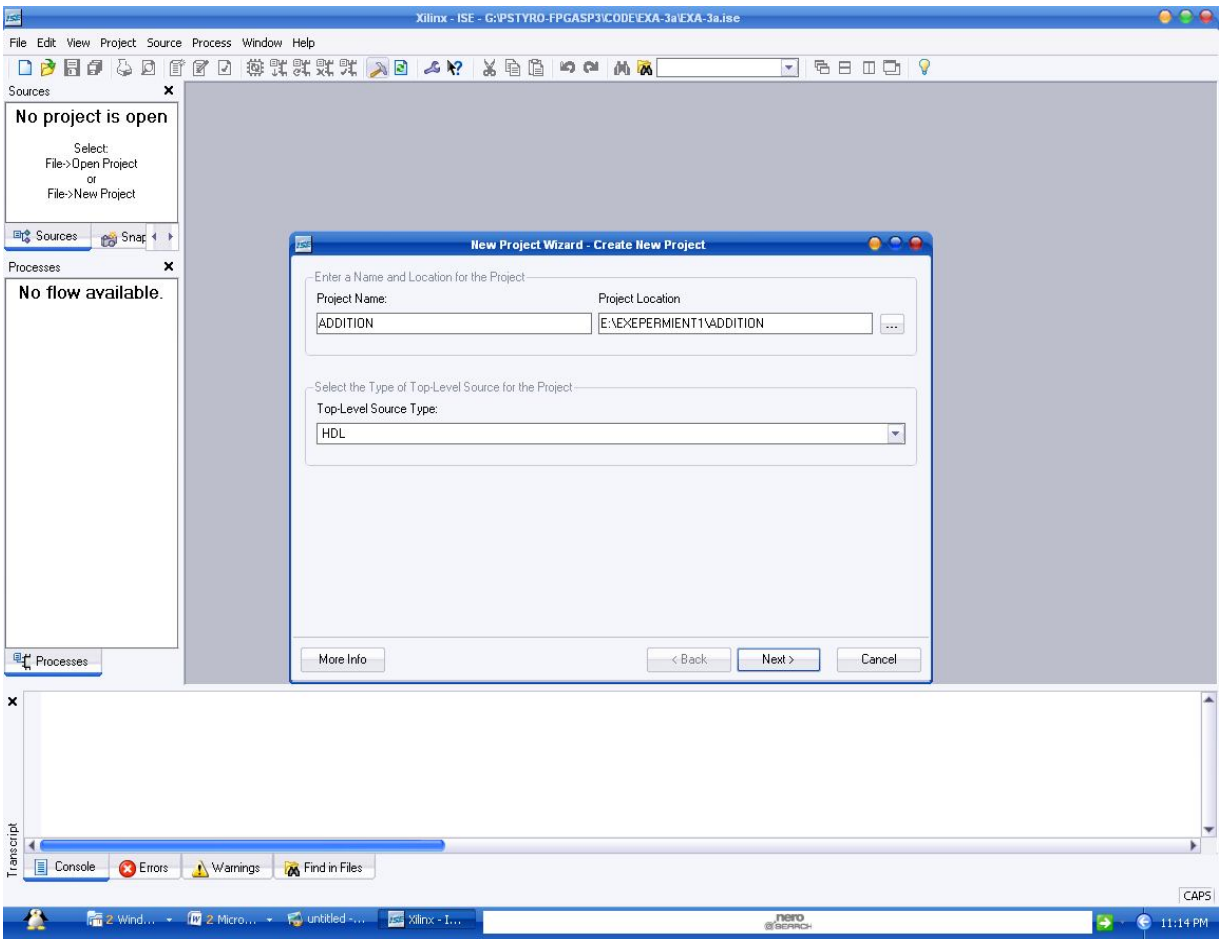


Figure 5

3. A window given in **Figure-6** will appear. Fill in the properties in the table as shown below:
   - ✓ Product Category: **All**
   - ✓ Family: **Spartan3**
   - ✓ Device: **XC3S200**
   - ✓ Package: **TQ144** selected.
   - ✓ Speed Grade: **-4**
   - ✓ Top-Level Module Type: **HDL**
   - ✓ Synthesis Tool: **XST (VHDL/Verilog)**
   - ✓ Simulator: **ISE Simulator (VHDL/Verilog)**

Then click "**Next**" (refer **Figure- 6, 7, 8)** and then "**Finish**" (refer **Figure- 9**).

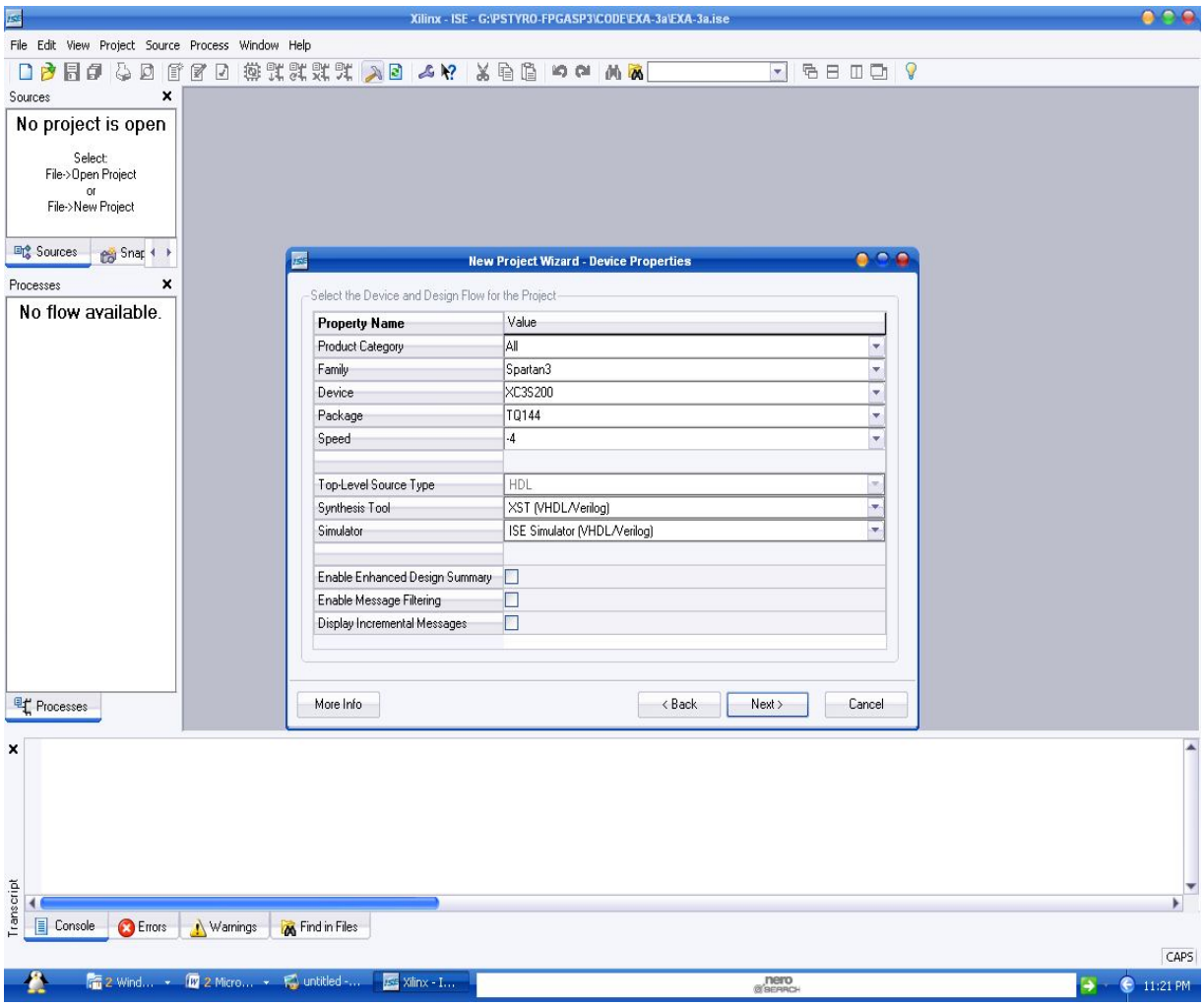

Xilinx - ISE - G:\PSTYRO-FPGASP3\CODE\EXA-3a\EXA-3a.ise
File Edit View Project Source Process Window Help
Sources
No project is open
Select:
File->Open Project
or
File->New Project
Sources
Snap
Processes
No flow available.
Processes
New Project Wizard - Device Properties
Select the Device and Design Flow for the Project
Property Name
Value
Product Category
All
Family
Spartan3
Device
XC3S200
Package
TQ144
Speed
-4
Top-Level Source Type
HDL
Synthesis Tool
XST (VHDL/Verilog)
Simulator
ISE Simulator (VHDL/Verilog)
Enable Enhanced Design Summary
Enable Message Filtering
Display Incremental Messages
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
CAPS
11:21 PM

FIGURE 6

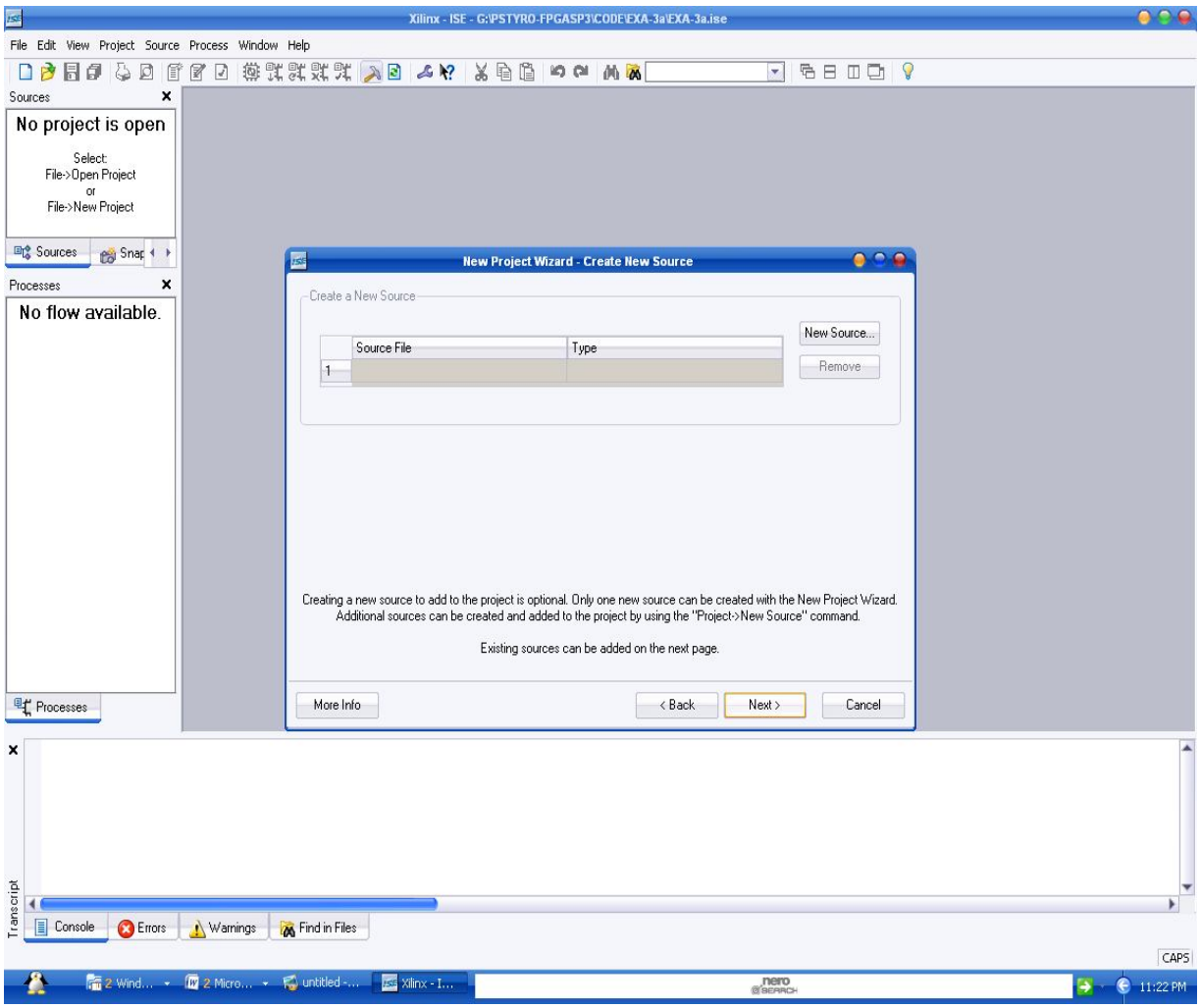

Xilinx - ISE - G:\PSTYRO-FPGASP3\CODE\EXA-3a\EXA-3a.ise
File Edit View Project Source Process Window Help
Sources
No project is open
Select:
File->Open Project
or
File->New Project
Sources
Processes
No flow available.
Processes
New Project Wizard - Create New Source
Create a New Source
Source File
Type
New Source...
Remove
Creating a new source to add to the project is optional. Only one new source can be created with the New Project Wizard.
Additional sources can be created and added to the project by using the "Project->New Source" command.
Existing sources can be added on the next page.
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
CAPS
11:22 PM

FIGURE 7

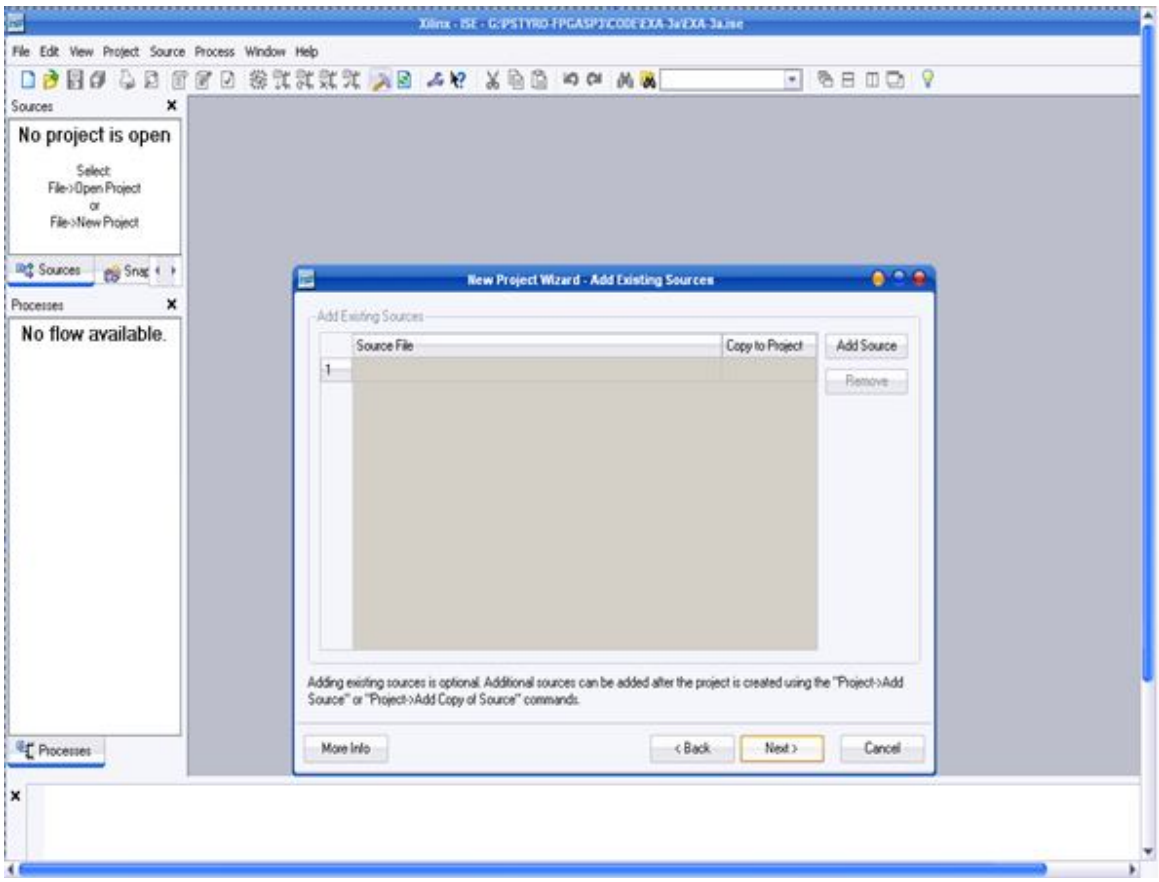

File Edit View Project Source Process Window Help
Sources
No project is open
Select
File->Open Project
or
File->New Project
Sources
Processes
No flow available.
Processes
New Project Wizard - Add Existing Sources
Add Existing Sources
Source File
Copy to Project
Add Source
Remove
1
Adding existing sources is optional. Additional sources can be added after the project is created using the "Project->Add Source" or "Project->Add Copy of Source" commands.
More Info
< Back
Next >
Cancel

FIGURE 8

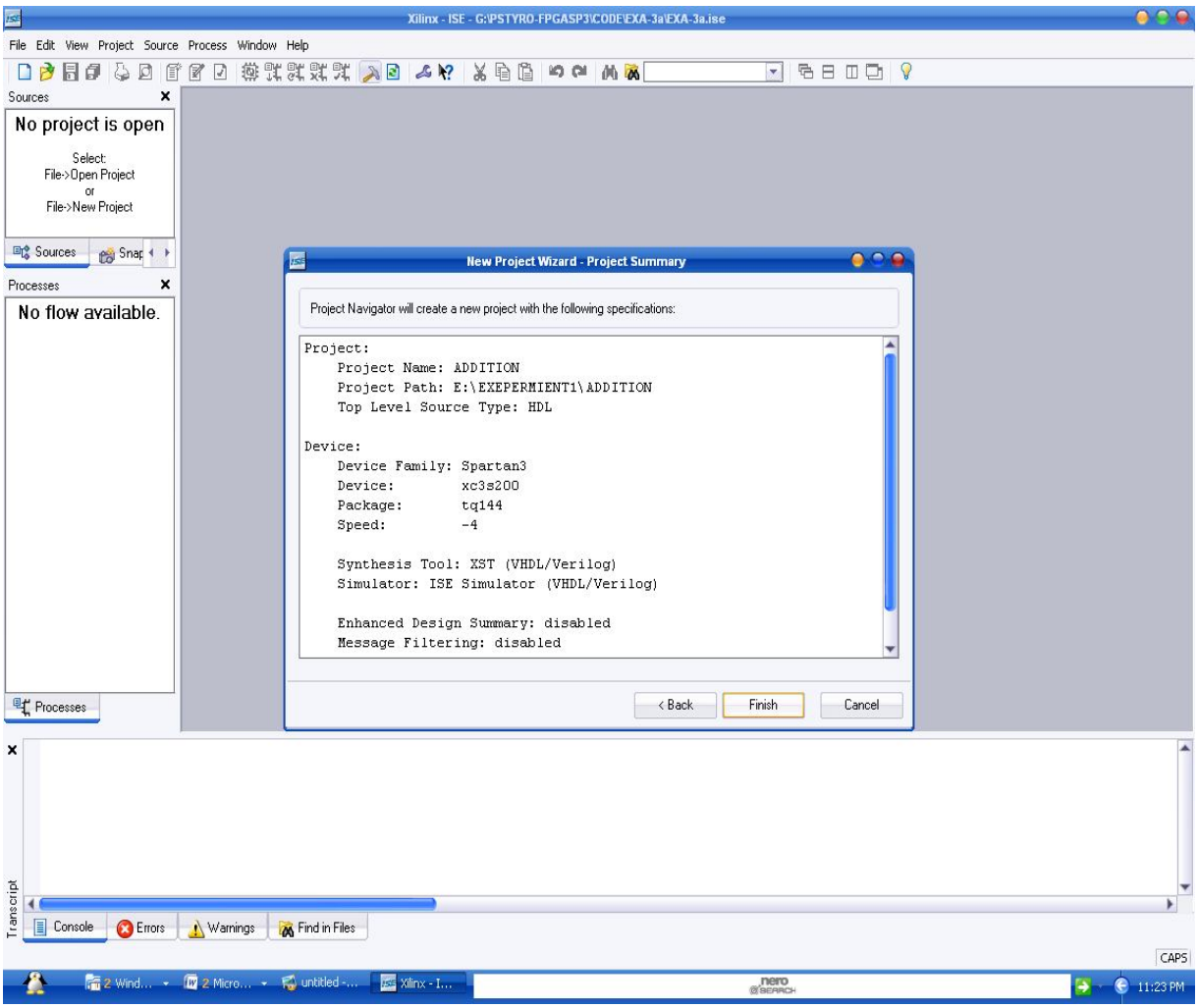


FIGURE 9

4. A window given in **Figure-10** will appear. Select **Project** menu ➔ **New Source** (refer **Figure-11**). A window given in **Figure-12** will appear. Then select **VHDL module**, and specify the file name in appropriate field as shown in figure-13 and Click **Next**. If you want you can give inputs & outputs in the appropriate positions in the window shown in **Figure -14.**You can also skip these information by simply clicking N**ext** button and click **Finish** (refer **Figure - 15)**.

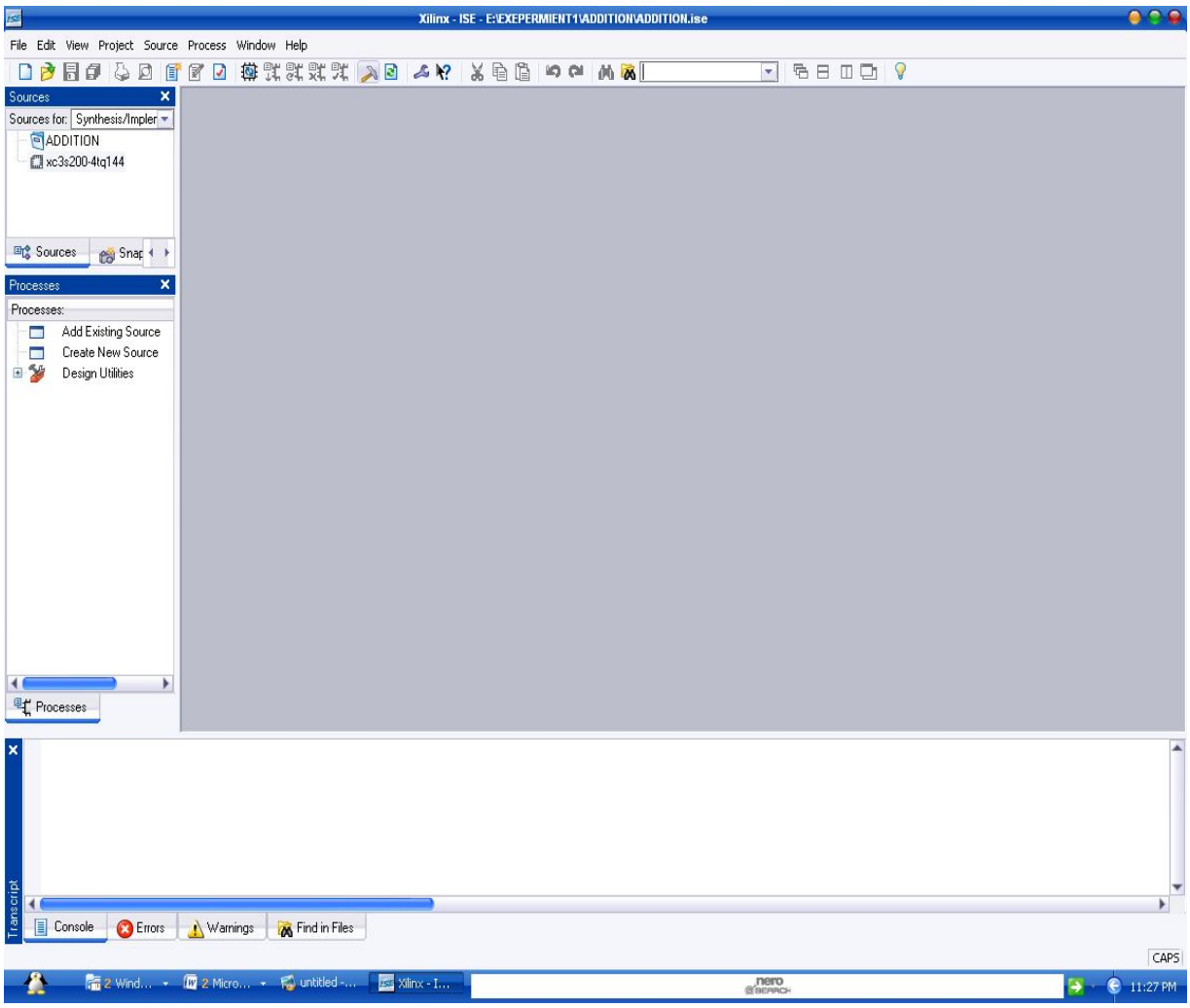

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
Sources
Snap
Processes
Processes:
Add Existing Source
Create New Source
Design Utilities
Processes
Transcript
Console
Errors
Warnings
Find in Files
CAPS
2 Wind...
2 Micro...
untitled -...
Xilinx - I...
11:27 PM

Figure-10

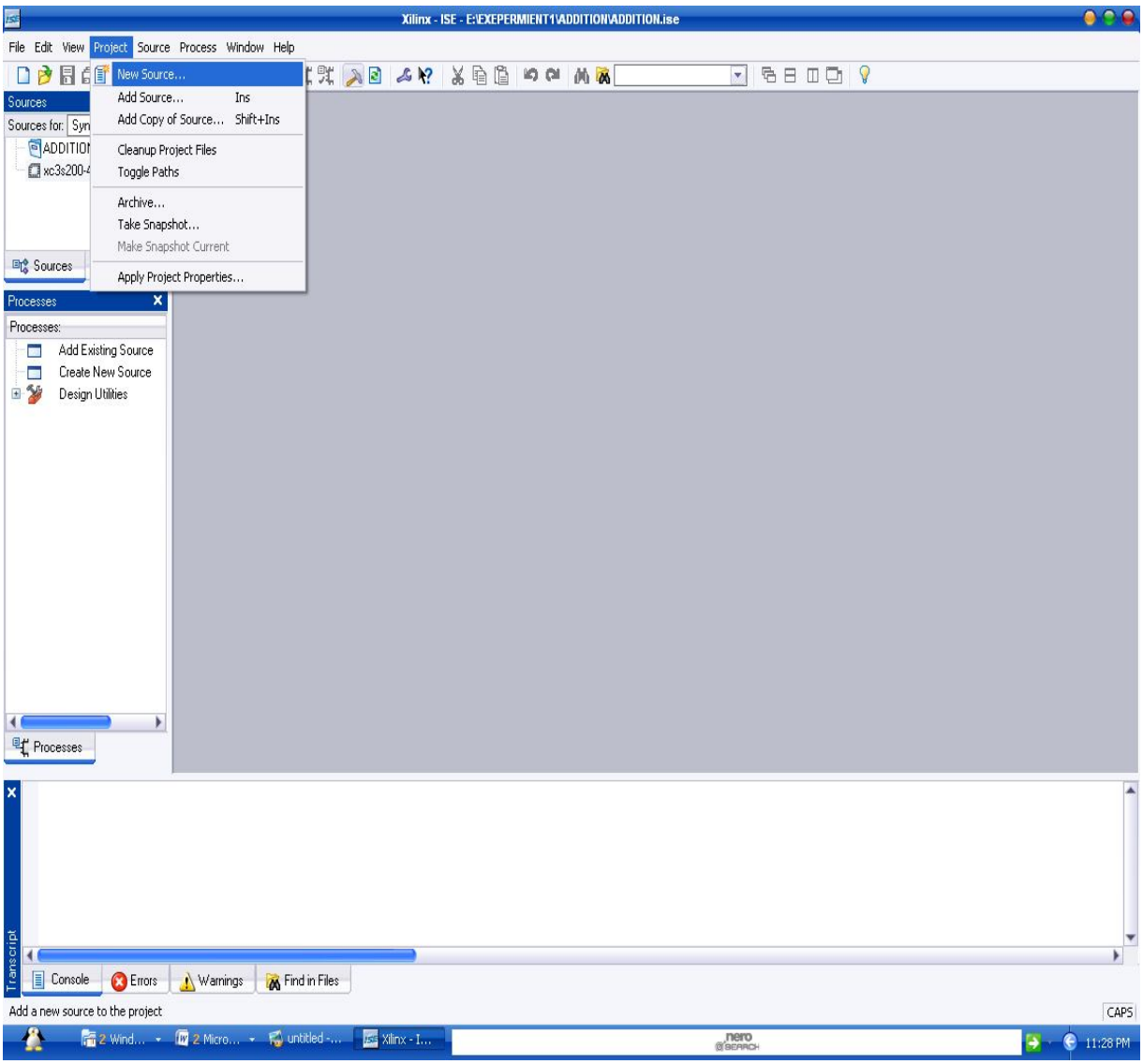

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise
File Edit View Project Source Process Window Help
New Source...
Add Source... Ins
Add Copy of Source... Shift+Ins
Cleanup Project Files
Toggle Paths
Archive...
Take Snapshot...
Make Snapshot Current
Apply Project Properties...
Sources
Processes
Add Existing Source
Create New Source
Design Utilities
Transcript
Console
Errors
Warnings
Find in Files
Add a new source to the project
CAPS
11:28 PM

Figure-11

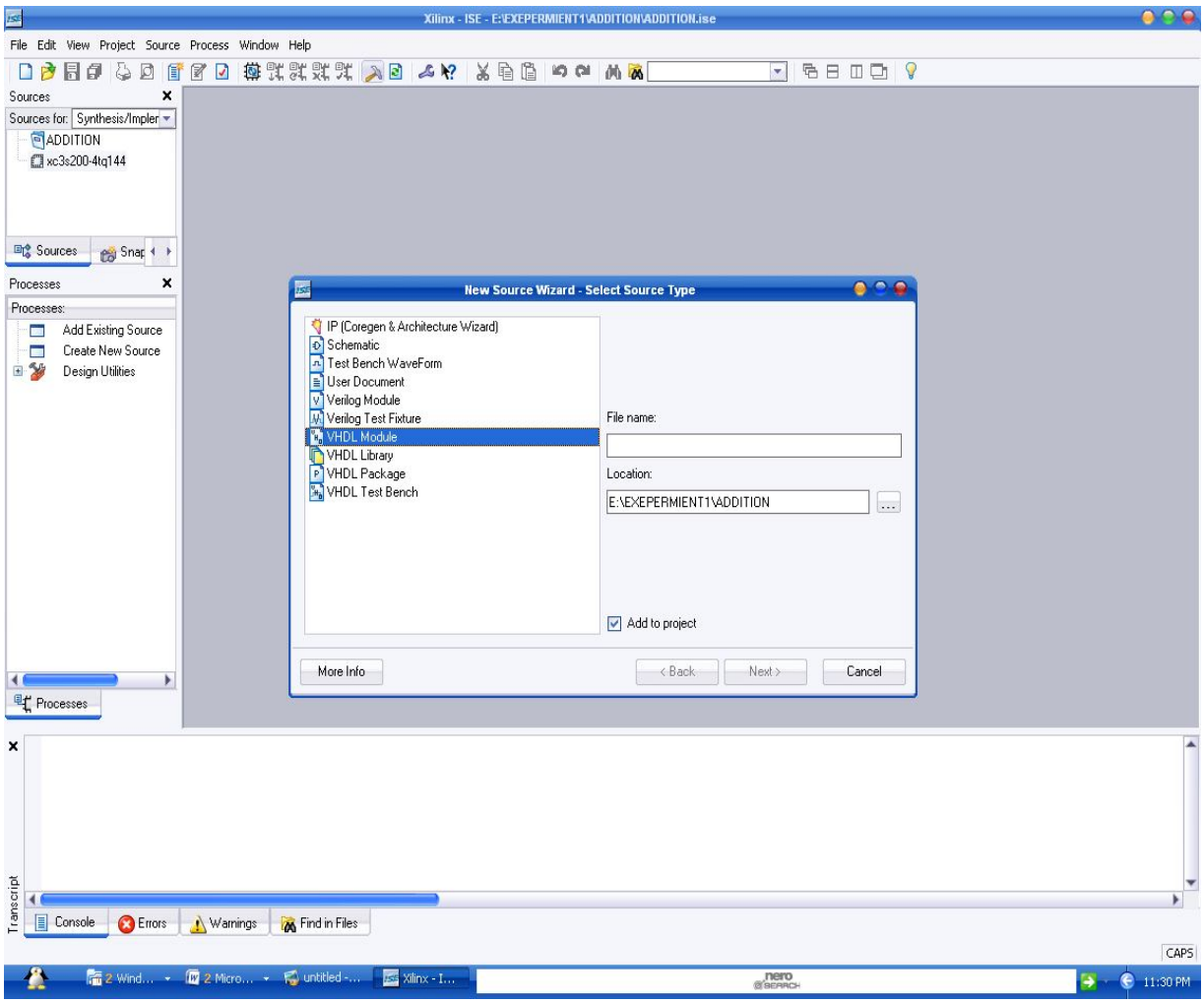

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
Sources
Snap
Processes
Processes:
Add Existing Source
Create New Source
Design Utilities
Processes
New Source Wizard - Select Source Type
IP (Coregen & Architecture Wizard)
Schematic
Test Bench WaveForm
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
File name:
Location:
E:\EXEPERMIENT1\ADDITION
Add to project
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
CAPS
11:30 PM

Figure-12

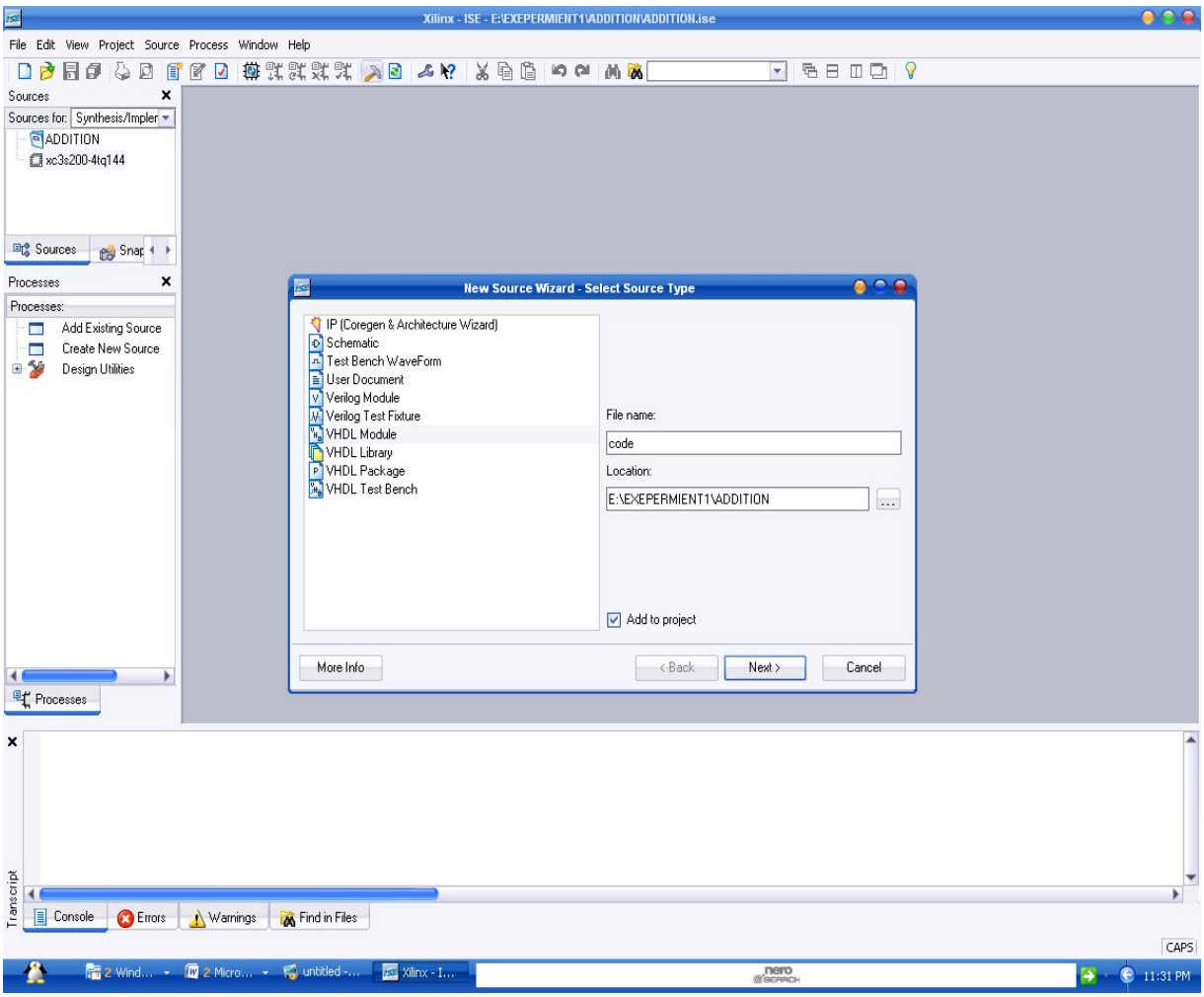

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
Sources
Snap
Processes
Processes:
Add Existing Source
Create New Source
Design Utilities
Processes
New Source Wizard - Select Source Type
IP (Coregen & Architecture Wizard)
Schematic
Test Bench WaveForm
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
File name:
code
Location:
E:\EXEPERMIENT1\ADDITION
Add to project
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
CAPS
11:31 PM

Figure-13

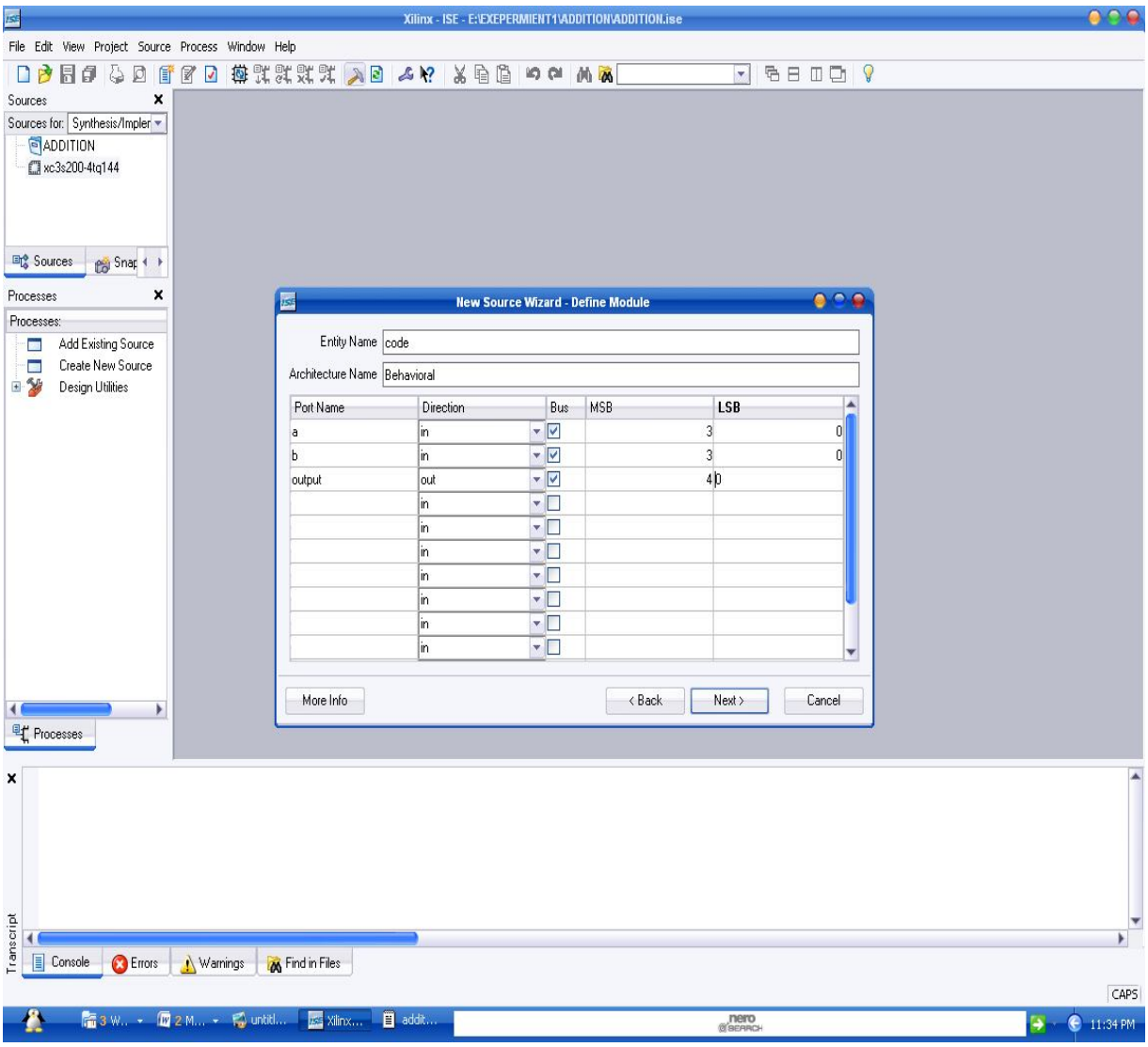

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
Sources
Snap
Processes
Processes:
Add Existing Source
Create New Source
Design Utilities
Processes
New Source Wizard - Define Module
Entity Name code
Architecture Name Behavioral
Port Name
Direction
Bus
MSB
LSB
a in 3 0
b in 3 0
output out 4 0
More Info
< Back
Next >
Cancel
Transcript
Console
Errors
Warnings
Find in Files
CAPS
11:34 PM

Figure-14

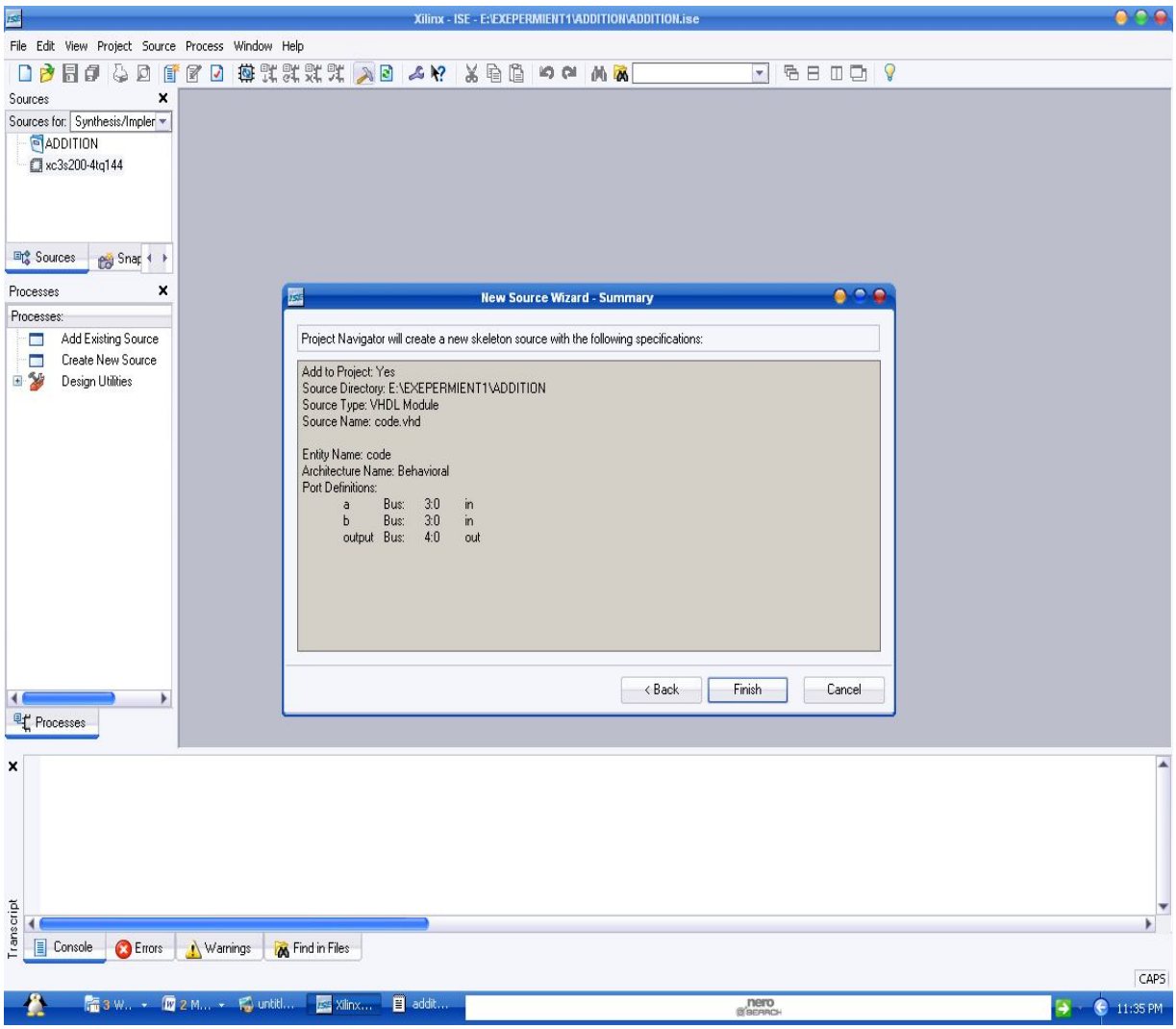


Figure-15

5. A window shown in **Figure-16** will appear. You can type your VHDL code in the right side of window and save it by clicking on the **Save** button (refer **Figure-17**). Now in the "**Processes:**" window double click **Synthesize-XST** as shown in **Figure-18**.

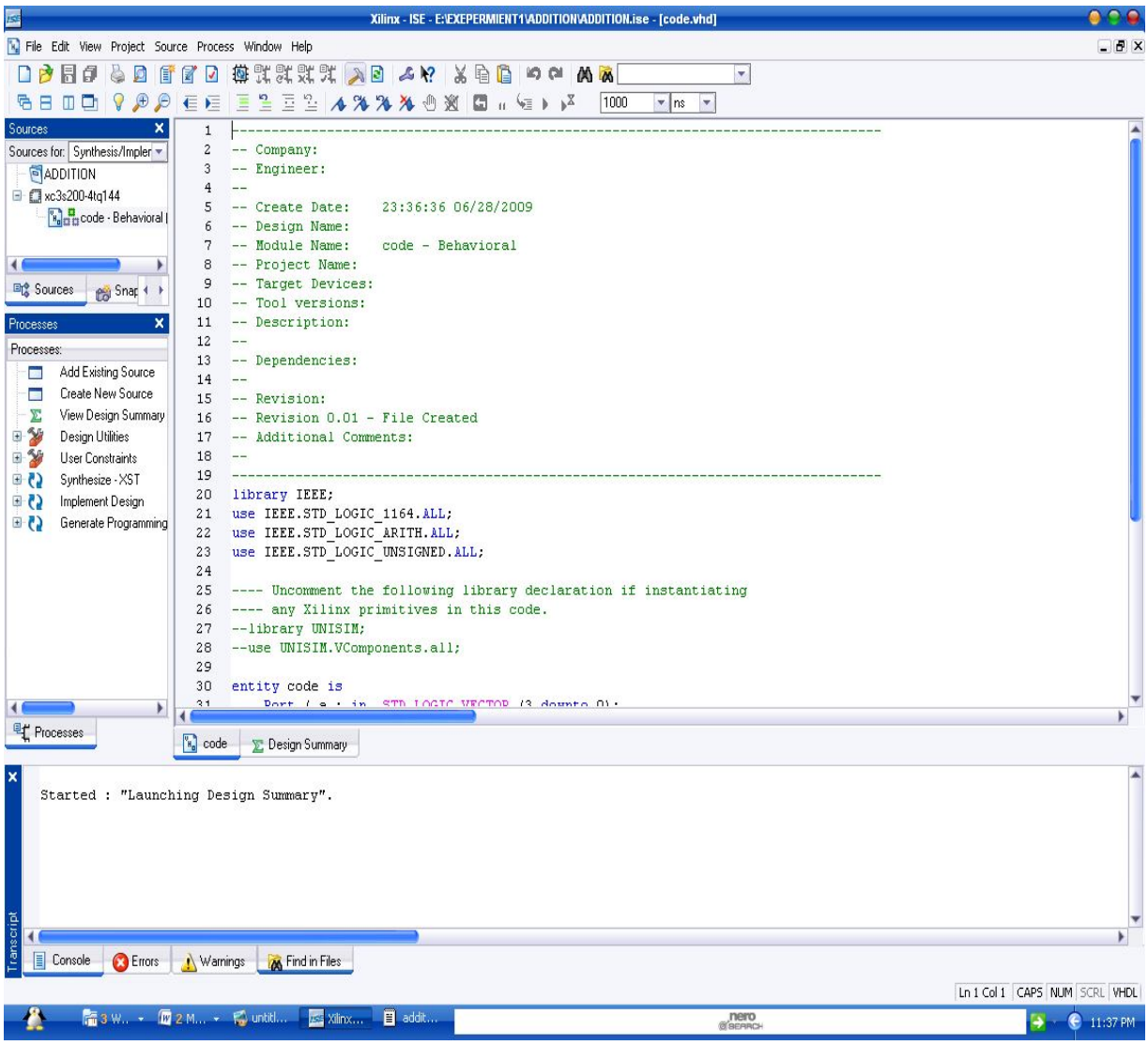

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise - [code.vhd]
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
code - Behavioral
Processes
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming
1 ----------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 23:36:36 06/28/2009
6 -- Design Name:
7 -- Module Name: code - Behavioral
8 -- Project Name:
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity code is
code
Design Summary
Started : "Launching Design Summary".
Transcript
Console
Errors
Warnings
Find in Files
Ln 1 Col 1
CAPS
NUM
SCRL
VHDL
11:37 PM

Figure-16

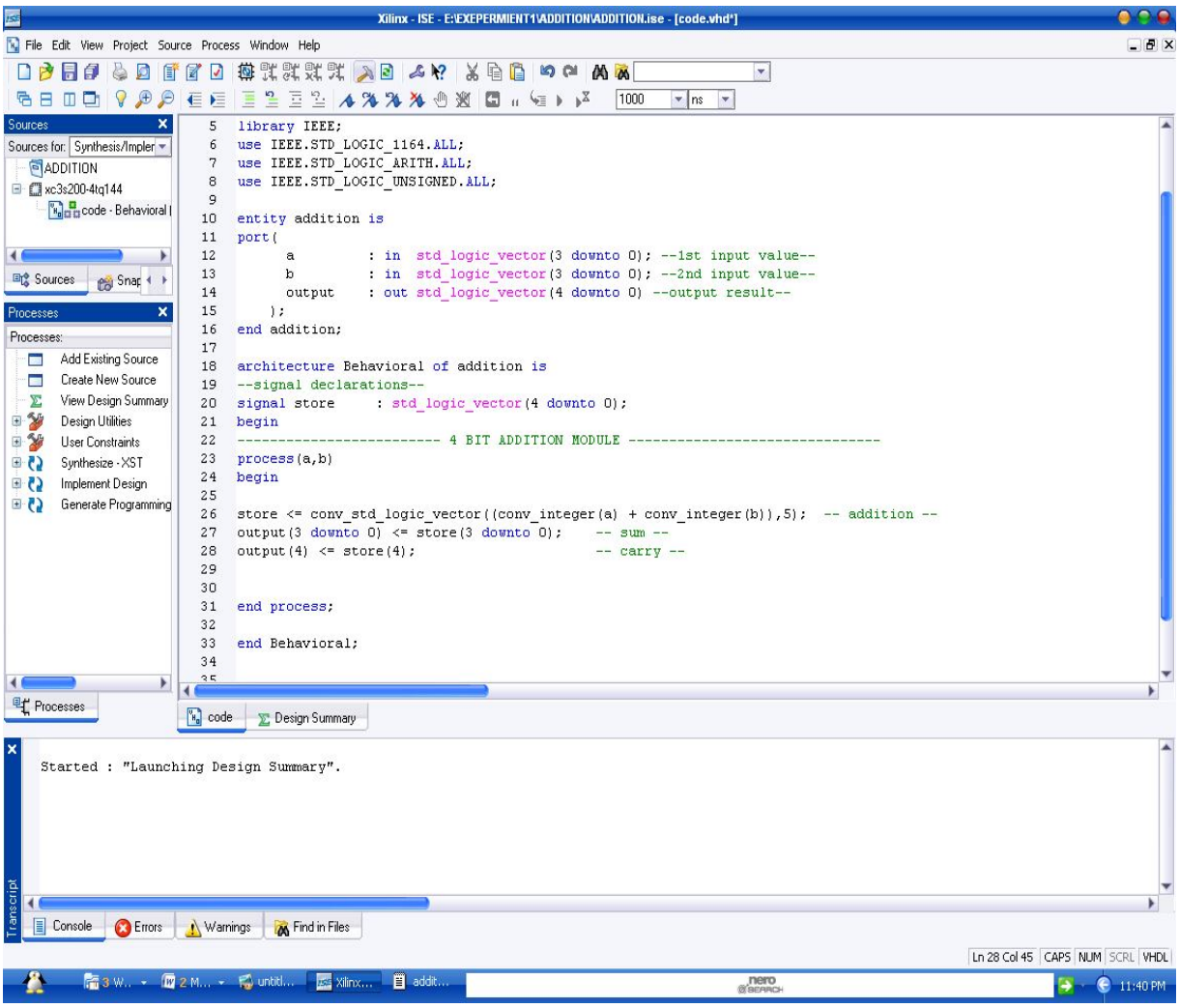
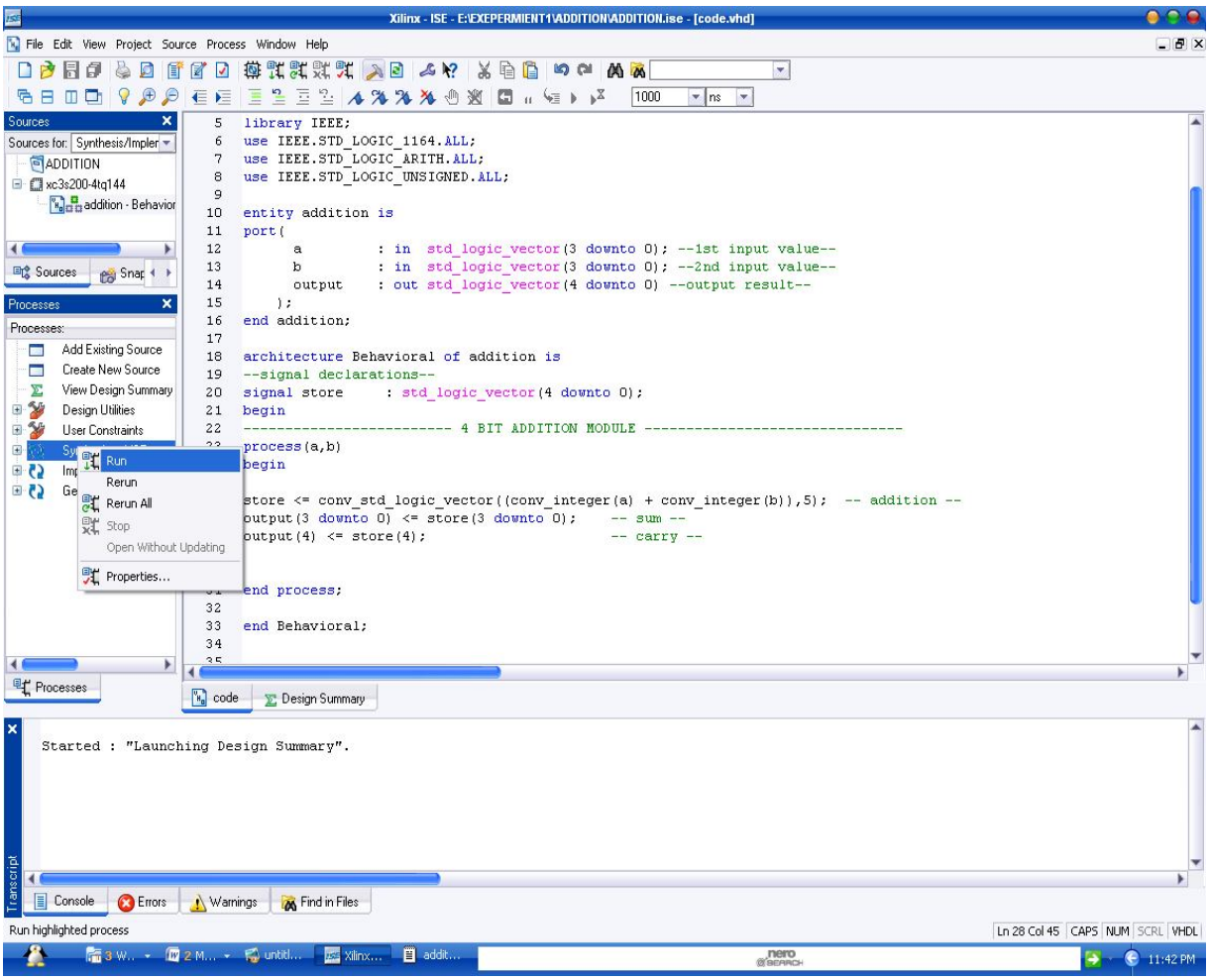

Xilinx - ISE - E:\EXEPERMIENT1\ADDITION\ADDITION.ise - [code.vhd*]
File Edit View Project Source Process Window Help
Sources
Sources for: Synthesis/Impler
ADDITION
xc3s200-4tq144
code - Behavioral
Processes
Processes:
Add Existing Source
Create New Source
View Design Summary
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity addition is
port(
a : in std_logic_vector(3 downto 0); --1st input value--
b : in std_logic_vector(3 downto 0); --2nd input value--
output : out std_logic_vector(4 downto 0) --output result--
);
end addition;
architecture Behavioral of addition is
--signal declarations--
signal store : std_logic_vector(4 downto 0);
begin
------------------------- 4 BIT ADDITION MODULE ------------------------------
process(a,b)
begin
store <= conv_std_logic_vector((conv_integer(a) + conv_integer(b)),5); -- addition --
output(3 downto 0) <= store(3 downto 0); -- sum --
output(4) <= store(4); -- carry --
end process;
end Behavioral;
code
Design Summary
Started : "Launching Design Summary".
Console
Errors
Warnings
Find in Files
Transcript
Ln 28 Col 45
CAPS
NUM
SCRL
VHDL
11:40 PM

Figure-17




```
5   library IEEE;
6   use IEEE.STD_LOGIC_1164.ALL;
7   use IEEE.STD_LOGIC_ARITH.ALL;
8   use IEEE.STD_LOGIC_UNSIGNED.ALL;
9
10  entity addition is
11  port(
12        a         : in  std_logic_vector(3 downto 0); --1st input value--
13        b         : in  std_logic_vector(3 downto 0); --2nd input value--
14        output    : out std_logic_vector(4 downto 0) --output result--
15      );
16  end addition;
17
18  architecture Behavioral of addition is
19  --signal declarations--
20  signal store     : std_logic_vector(4 downto 0);
21  begin
22  ------------------------- 4 BIT ADDITION MODULE ------------------------------
23  process(a,b)
24  begin
25
26  store <= conv_std_logic_vector((conv_integer(a) + conv_integer(b)),5);  -- addition --
27  output(3 downto 0) <= store(3 downto 0);     -- sum --
28  output(4) <= store(4);                       -- carry --
29
30
31  end process;
32
33  end Behavioral;
34
```

Figure-18

✍**Note:** You must correct any errors found in your source files. You can check for errors in the Console tab of the Transcript window. If you continue without valid syntax, you will not be able to simulate or synthesize your design.

6 .Assigning Pin Location Constraints

Specify the pin locations for the ports of the design so that they are connected correctly on the Spartan-3 TYRO/PREMIER board.

To constrain the design ports to package pins, do the following before that refer the LED & SWITCH PIN DETAILS in before pages or see board for giving the pin location.

- Verify that **ADDITION is** selected in the Sources window.

- Double-click the **Assign Package Pins** process found in the User Constraints process group. The Xilinx Pin out and Area Constraints Editor (PACE) opens.(**refer figure 19 & 20)**

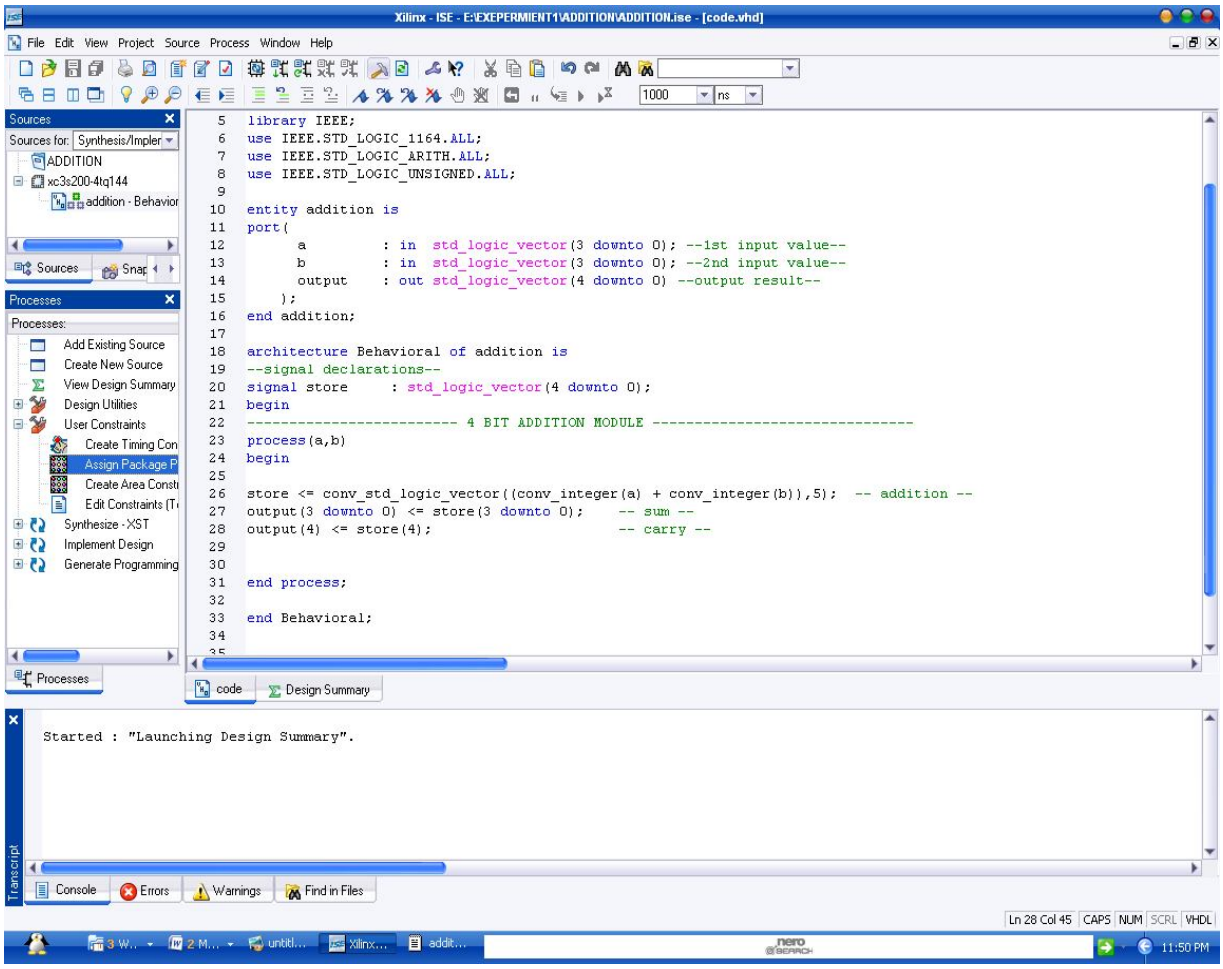


FIGURE 19

- Select the **Package View** tab.
- In the Design Object List window, enter a pin location for each pin in the **Loc** column using the seeing on board LED & SWITCHES or refer before pages in manual.

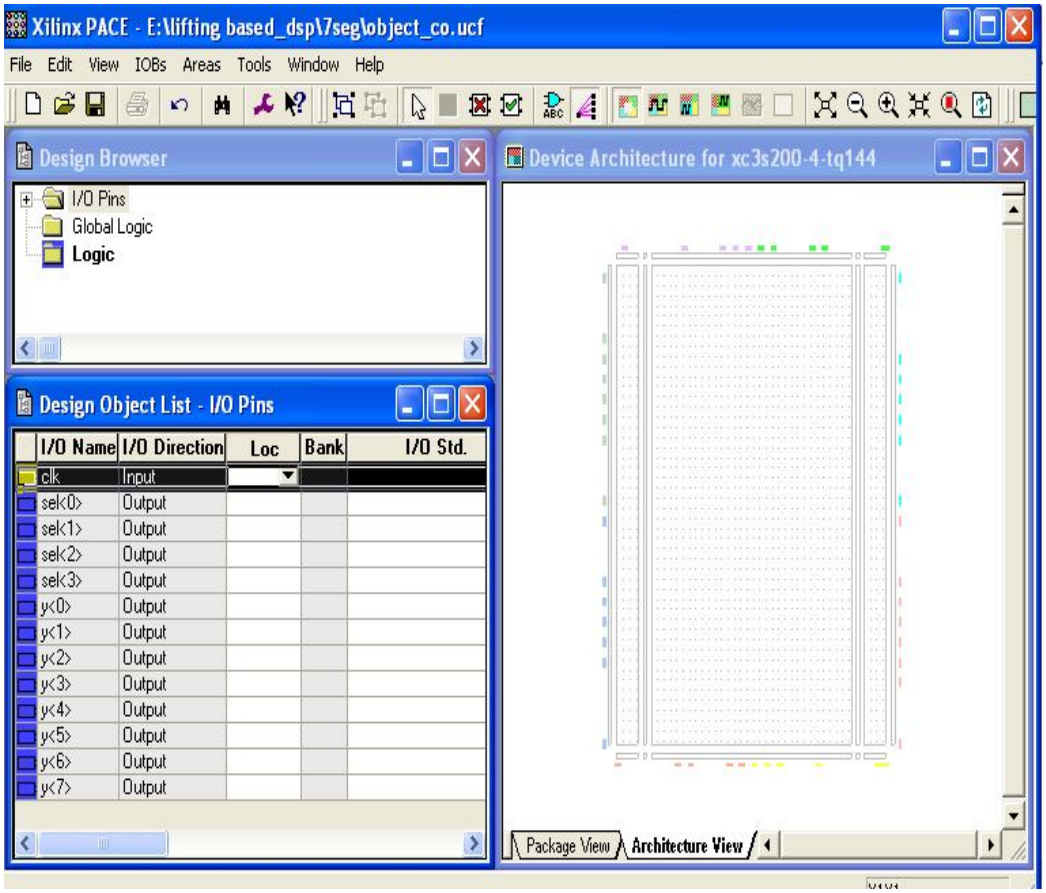


Figure 20

- Select **File → Save**. You are prompted to select the bus delimiter type window will open. Select synplify VHDL/exemplar default [] and click **OK**.(refer FIGURE 21)
- Close PACE.

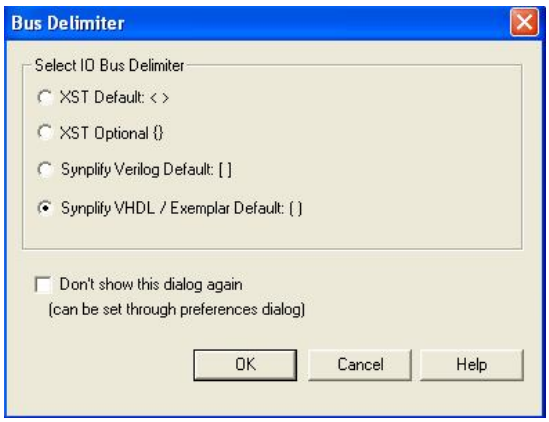


FIGURE 21

8. There are two ways of downloading the program into the target hardware, one is generating the**.bit file** for downloading into the **FPGA** device and the other is generating the.**mcs file** for downloading into the **PROM** device.

9. To download the program as **.bit** file in **Boundary Scan mode**, do the following steps:
i) Shunt of Jumper **J2** of must be in **S3** position for **FPGA** selection.
ii) In the **Project Navigator** window and click **Implement design** in **Processes:** category (refer **Figure-22**).

✍ Notice that after Implementation is complete, the Implementation processes have a green check mark next to them indicating that they completed successfully without Errors or Warnings.

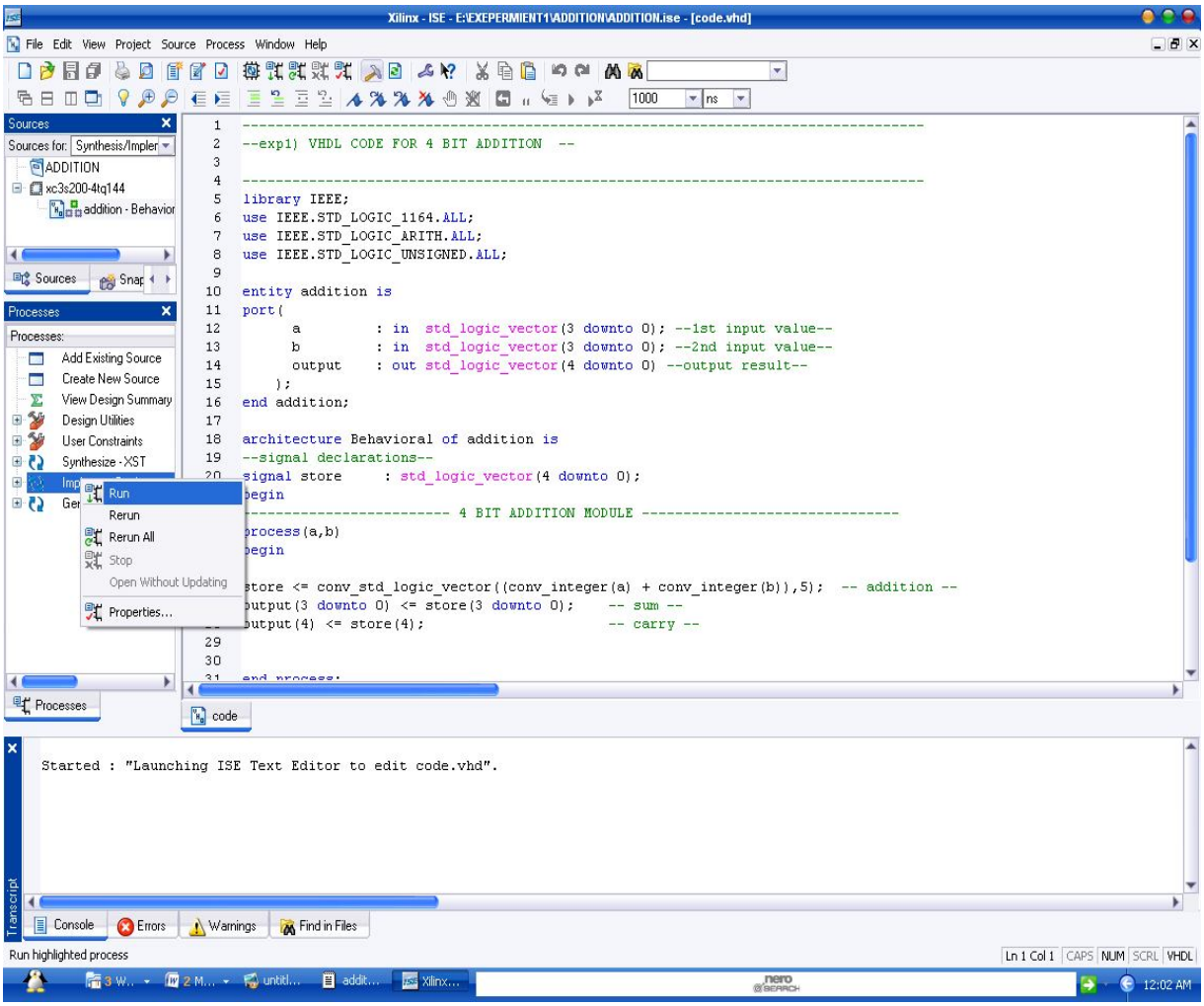


Figure-22

10. After implement design has become successful (refer **Figure-23**), click **Generate Programming File** (refer **Figure- 24**). After generate programming file has been completed successfully click **Manage Configuration Project (IMPACT) (**refer **Figure-25)**.

11. Download Design to the Spartan™-3 TYRO/PRIMER Kits.

This is the last step in the design verification process.

- Connect the 9V DC power cable to the power input on the TYRO/Primer Kits. (J1).
- Connect the download cable between the PC and development board (J4).
- Check **Synthesis/Implementation** from the drop-down list in the Sources window (refer figure 26).

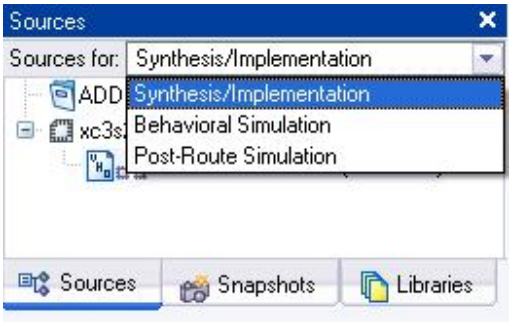


Figure 26

- Select **addition** in the Sources window (refer figure 27).

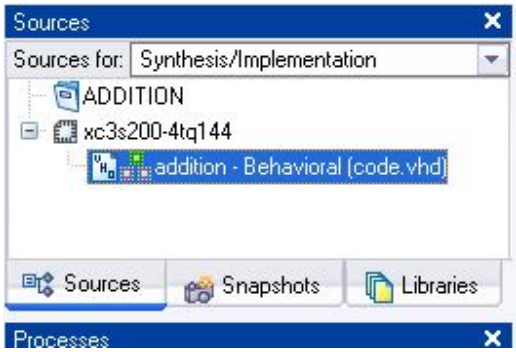


Figure 27

- In the Processes window, click the "**+**" sign to expand the **Generate Programming File** processes.
- Double-click the **Configure Device (iMPACT)** process (Refer FIGURE 28)

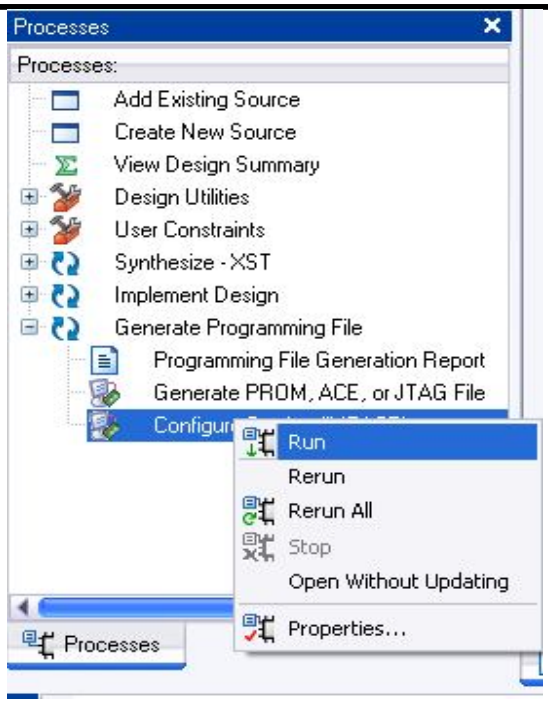
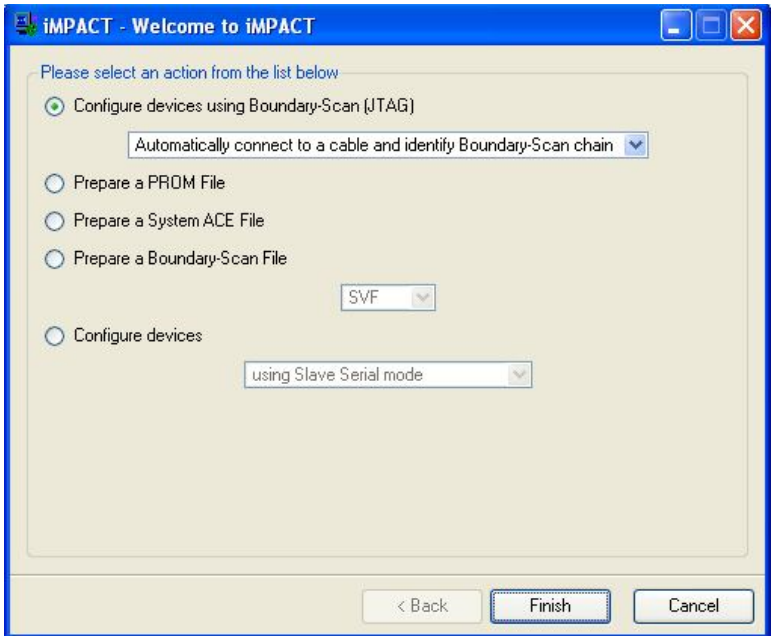
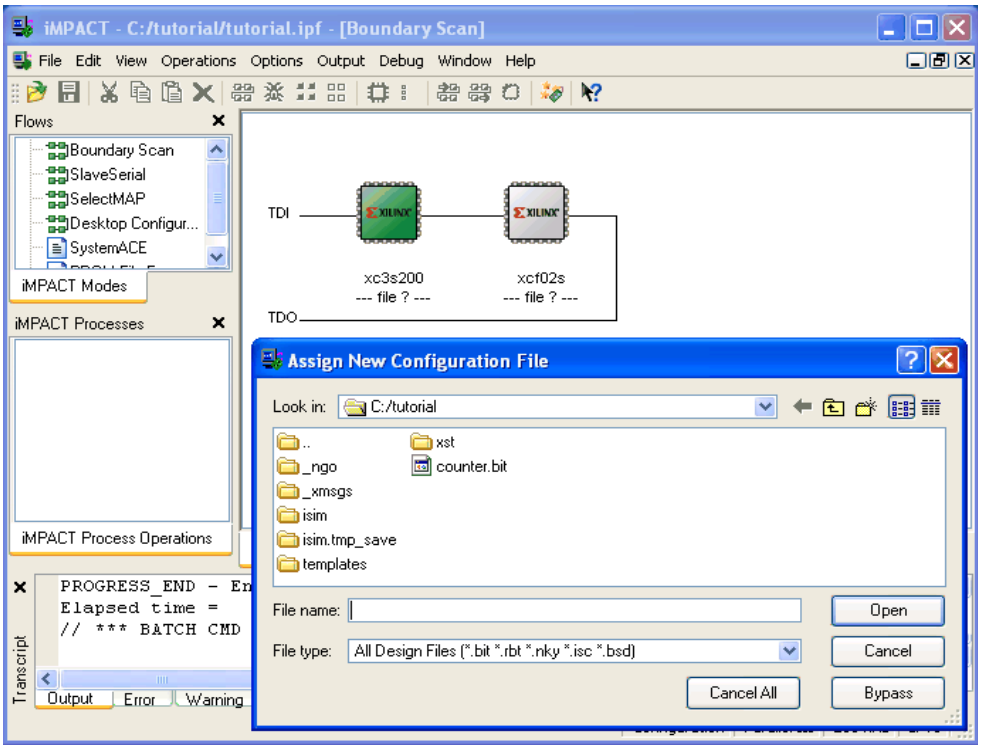


FIGURE 28



- In the Welcome dialog box, select **Configure devices using Boundary-Scan (JTAG)**.
- Verify that **Automatically connect to a cable and identify Boundary-Scan chain** is selected.
- Click **Finish**.

- If you get a message saying that there are two devices found, click **OK** to continue.
- The devices connected to the JTAG chain on the board will be detected and displayed in the iMPACT window
- The **Assign New Configuration File** dialog box appears. To assign a configuration file to the xc3s200 device in the JTAG chain, select the addition.bit file and click **Open.**



- If you get a Warning message, click **OK**.
- Select **Bypass** to skip any remaining devices.
- Right-click on the xc3s200 device image, and select **Program...** The **Programming Properties** dialog box opens.
- Click **OK** to program the device.

When programming is complete, the Program Succeeded message is displayed.

- Close iMPACT without saving.

  After successful configuration you can check the output on your target hardware.