

User Manual

Version: 2

ni logic Pvt. Ltd, 25/B-5, Bandal Complex, Bhusari Colony, Paud Road, Kothrud, Pune – 411 038, Maharashtra, India Ph: +91-20-5286948 www.ni2designs.com

Table of Contents

- 1. Introduction
 - a. Introduction to VLSI and PLDs
 - b. Basics about kits and required features
- 2. About USDP
 - a. Product introduction.
 - b. Product Features.
 - c. Technical specification of all modules.
- 3. Product Overview, applications and examples.
- 4. System Diagrams
 - a. USDP board connectivity.
 - b. I/O Connections.
 - c. Add-on connector detail.
- 5. Pin Assignments
 - a. FPGA pin assignment.
 - b. Add-on Module pin assignment.
- 6. Jumper and Headers (Pin description and settings).
- 7. Using EDA tools
 - a. PLD Design Flow.
 - b. 89c51 controller design flow.
 - c. PIC controller design flow.
 - d. Using Xilinx ISE software.
 - e. Using Altera Quartus-II software.
 - f. Using keil compiler.
- 8. Precautions
- 9. Configuration
 - a. Configuring Xilinx & Altera Devices.
 - b. Slot selection.
 - c. Programming Modes.
 - d. Jumper setting for mode selection.
 - e. Programming 89c51RD2 microcontroller.
 - f. Programming PIC16F877 controller.
- 10. Using Add-on Modules
 - a. Xilinx FPGA Module.
 - b. Altera FPGA Module.
 - c. ADC/DAC Module.
 - d. 89c51RD2 Module.
 - e. PIC uC Module.
 - f. SRAM Memory Card.
 - g. Power Electronics Module.
 - h. General Purpose PCB.
- 11. Designing Application on USDP using provided modules
- 12. An application implementation on USDP
- 13. Sample codes
- 14. Glossary
- 15. Troubleshooting

Disclaimer

Device Overview & Datasheets

Chapter 1 Introduction to Programmable Logic

What is Programmable Logic?

In the world of digital electronic systems, there are three basic kinds of devices: memory, microprocessors, and logic. Memory devices store random information such as the contents of a spreadsheet or database. Microprocessors execute software instructions to perform a wide variety of tasks such as running a word processing program or video game. Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform.

Fixed Logic versus Programmable Logic

Logic devices can be classified into two broad categories - fixed and programmable. As the name suggests, the circuits in a fixed logic device are permanent, they perform one function or set of functions - once manufactured, they cannot be changed. On the other hand, programmable logic devices (PLDs) are standard, off-the-shelf parts that offer customers a wide range of logic capacity, features, speed, and voltage characteristics - and these devices can be changed at any time to perform any number of functions.

With fixed logic devices, the time required to go from design, to prototypes, to a final manufacturing run can take from several months to more than a year, depending on the complexity of the device. And, if the device does not work properly, or if the requirements change, a new design must be developed. The up-front work of designing and verifying fixed logic devices involves substantial "non-recurring engineering" costs, or NRE. These NRE costs can run from a few hundred thousand to several million dollars.

With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device.

Another key benefit of using PLDs is that during the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology - to change the design, the device is simply reprogrammed. Once the design is final, customers can go into immediate production by simply programming as many PLDs as they need with the final software design file.

CPLDs and FPGAs

The two major types of programmable logic devices are field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs). Of the two, FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA provides millions of "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired IP cores (such as the IBM Power PC, PCI cores, microcontrollers, peripherals, etc), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.

CPLDs, by contrast, offer much smaller amounts of logic - up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. Low power CPLDs are also available and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

The PLD Advantage

Fixed logic devices and PLDs both have their advantages. Fixed logic devices, for example, are often more appropriate for large volume applications because they can be mass-produced more economically. For certain applications where the very highest performance is required, fixed logic devices may also be the best choice.

However, programmable logic devices offer a number of important advantages over fixed logic devices, including:

- PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
- ? PLDs do not require long lead times for prototypes or production parts the PLDs are already on a distributor's shelf and ready for shipment.
- PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets PLD suppliers incur those costs when they design their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.
- PLDs allow customers to order just the number of parts they need, when they need them, allowing them to control inventory. Customers who use fixed logic devices often end up with excess inventory which must be scrapped, or if demand for their product surges, they may be caught short of parts and face production delays.
- ? PLDs can be reprogrammed even after a piece of equipment is shipped to a customer.

Conclusion

The value of programmable logic has always been its ability to shorten development cycles for electronic equipment manufacturers and help them get their product to market faster. As PLD suppliers continue to integrate more functions inside their devices, reduce costs, and increase the availability of time-saving IP cores, programmable logic is certain to expand its popularity with digital designers.

Prototyping kits for Programmable Logic devices

With the advent of any programmable device whether it be an EEPROM, PAL, PLA, microcontroller, microchip or a FPGA, the need of programmer was mandatory. The application or usage of programmer was just to download the programming file in the device. But the FPGAs were exceptional case.

Applications like prototyping, product development and learning of VLSI converted the programmers of FPGA into a more complex protoboards, where the designers can program the FPGAs, develop and verify the design and finally can go for production after satisfactory results.

Manufacturing of these trainer protoboards was earlier done by foreign companies and were costlier, but day by day Indian manufacturers developed their own protoboards for the Indian market, and thus VLSI designers, engineers and educational industry of India got easy access to these protoboards.

They are available in a wide range depending on type of device used.

Applications of FPGA/CPLD protoboards are:

- 1. ASIC prototyping
- 2. Product development
- 3. Verification of designs in CPLDs and FPGAs.
- 4. Helping users to exploit architectural features of CPLDs and FPGAs.
- 5. Performing a wide range of experiments, by actually downloading the designs into physical devices.
- 6. Understanding use of HDL's.

FPGAs and CPLDs have seen an exponential rise in their architectures and hence their applications. In mid '90s PLDs were used in packing of digital logic into them, but today devices are much more complex and available with inbuilt hard-wired microprocessors, PCI cores, peripherals, etc. Thus their protoboards/kits also need to be designed accordingly; so as to exploit the complete architecture and feature set of PLDs a designer needs some basic specifications on the protoboards

The most demanding feature set of any PLD Protoboard is as follows:

- 1. Ability to program the PLD in circuit
- 2. User inputs and output
- 3. Displays like 7-segment and LCDs
- 4. Keyboard
- 5. Clock circuit
- 6. Power ON Reset
- 7. User interface I/Os
- 8. On board power supply
- 9. Any bus interface
- 10. Easy to use and operate
- 11. etc.

Needs of today's protoboards

VLSI trainer protoboards or kits are extensively used in prototyping of FPGA based product designs, training and for experimental purposes. All of them posses feature set towards programming of FPGAs and just few user input-output facilities. Very few protoboards offer a complete solution for integration of various user modules with ease. With the advancement in the system requirements and specifications, designers are finding difficult in development of FPGA based designs.

The increase in complexity of design has made a significant change in FPGA architecture. With Virtex-II pro device from Xilinx and Stratrix/Excalibur from Altera designers can integrate a complete system in them. With millions of system gate capacity and inbuilt microprocessors they give flexibility to designers to develop smaller and more complex designs.

Technology Interface

As the FPGA designs accounts the integration of dense memories, analog interface, DSP microprocessors and microcontrollers, many vendors are unable to provide a platform where a designer can integrate all kinds of modules with the FPGA.

High speed performance

Integration of other technology modules is not only the issue with complex FPGA designs. The most important part in FPGA designs is to achieve the target speed of operation. Many designs are required to operate more than 60MHz and how many actually guarantee this speed of operation with bad PCB designs, congested boards, bad connectors, and poor power supplies.

Generally the vendors use the berg connectors for the user interface and FPGA attachment, which cannot perform more that 20-30 MHz, the signals get deteriorated and poor single integrity of design is the outcome.

FPGA stacking

Also an FPGA designer needs a freedom to use any FPGA for his application, i.e., he may opt to use FPGAs from different vendors according to the design requirements and project needs. Most of the vendors have failed to give this kind of feature. FPGA stacking feature is most demanding one in the design of an FPGA protoboard, which means a designer can interface more than one FPGA together and that too from different vendors. Even if the foreign vendors have come with feature of FPGA stacking there is limitation on number of FPGAs and generally they use BERG pins for stacking which are unable to perform at high frequencies.

Easy user interface

Designers have always faced problems now and then in attaching external world signals with the protoboards. When it comes to external design module interface the problems increases as the interface is poor and testing becomes a headache.

Easy user interface has always the key point in designing these protoboards, for this manufacturers provide options of 'D' type connectors, BERG connector, or other connectors. These connectors have problems like limitation on number of I/Os, poor signal integrity and hassle of wires.

Up gradation

Looking at the customer needs the products are continuously up graded or redesigned. The system should be capable of up grading the system components and giving the product some added features to compete in the market. Generally protoboards don't have this kind of feature with them. After couple of project completions the designers find their existing protoboards old and difficult to upgrade. Where as to be in market the protoboards should posses the ability to upgrade themselves with the latest FPGAs, CPLDs, increase in number of user I/Os, interface with new technologies, replacement of existing FPGA with higher speed device, etc.

Chapter 2 About USDP

Overview of Universal System Development Protoboard

VLSI trainer protoboards or kits are extensively used in prototyping of FPGA based product designs, training and for experimental purposes. All of them posses features towards programming of FPGAs and just few user input-output facilities. Very few protoboards offer a complete solution for integration of various user modules with ease. With the advancement in the system requirements and specifications, designers are finding difficulty in development of FPGA based designs.

As the FPGA designs accounts the integration of dense memories, analog interface, DSP microprocessors and microcontrollers, many vendors are unable to provide a platform where a designer can integrate all kinds of modules with the FPGA.

With increase in use and demand of FPGAs in the communication, telecom, power electronics, motion control and educational industries the protoboards are in huge demand, thus **ni logic** took initiative to provide an excellent prototyping platform for these industries.

Our R&D arm **ni2 designs** have come out with the complete solution towards the problems of trainer kits in DSP, VLSI and Microprocessors.

ni2 designs have designed prototyping protoboards with the concept of backplane interface of various modules, giving a complete solution for the platform integration of VLSI, DSP, microprocessors and power electronics applications.

With this product **ni logic** is aiming to give the industry a product that can be used in various applications, thus solving the problems of designers for high speed designs and platform integration. With this unique idea product, **ni logic** will stand ahead in market segment of PLD products and will prove with its sales figures in coming years.

Features of USDP

- ? Easy to use and implement system designs.
- ? Slot cards for FPGA from Altera, Xilinx and other vendors with package support up to FG256.
- ? Stacking of multiple FPGAs (can be of different vendors).
- ? 64 bit general purpose bus interface with FPGA.
- ? 77 bit bus sharing between FPGAs.
- ? High performance backplane, good frequency response upto 80MHz designs (one of the fastest protoboards available in India).
- ? On board JTAG circuit for downloading.
- ? Multiple configuration options with JTAG chaining of devices.
- ? User selectable configuration modes, using either FLASH PROM / JTAG.
- ? 32 Digital I/Ps and O/Ps, each can be configured as input or output giving flexibility to designers.
- ? On board system reset circuit.
- ? Configuration reset circuit.
- ? Four seven segment Multiplexed display.
- ? 4x4 membrane keypad.
- ? On board crystal oscillator socket (user can select his desired oscillators).
- ? General-purpose user area for interface of user clock circuit.
- ? Ability to use Clock management circuits of FPGAs.
- ? Proper configuration of FPGAS with high-speed clocks through special scheme.
- ? SMPS.
- ? Support for different I/O Standards.
- ? A complete I/O bank for user VREF interface, using DB25 connector.
- ? Parallel port interface.
- ? Easily accessible user I/Os.
- ? 3 on board 120-pin connector for Add-on card interface.
- ? Stacking of maximum three Add-on card modules possible of different technologies.
- ? 89c51 Microcontroller card for traditional 8051 applications with ISP support.
- ? PIC Microcontroller card for industrial based applications.
- ? Memory Card for data intensive applications.
- ? High performance ADC/DAC add-on card.
- ? Power module for motion control and electro-mechanical applications.
- ? Intensive user manual support with various examples and source codes.
- ? And many more.

Backplane features

- ? Supports interface with dual daughter PLD cards
- ? Supports FPGAs from reputed PLD vendors like Xilinx, Altera, etc.
- ? Stacking of maximum two PLD daughter cards possible, with on-board chaining circuit.
- ? 3 PCI connector based application specific add-on cards
- ? 64 bit bus sharing between application add-on cards and PLD daughter cards
- ? 77 bit bus sharing between PLD daughter cards
- ? 32 Digital I/Ps and O/Ps, each can be configured as input or output giving flexibility to designers.
- ? On board system reset circuit.
- ? Four seven segment Multiplexed display.
- ? 4x4 switch matrix keyboard interface header.
- ? On board crystal oscillator socket (user can select his desired oscillators).
- ? High performance backplane PCB.

PLD Daughter cards Available

Different PLD modules are available for different devices from vendors like Xilinx, Altera, etc. The user can prototype upto **1 million (10,00,000)** gate count designs.

Here is the list of PLD daughter add-on cards and their feature set;

Features of PLD Modules

Xilinx FPGA Card

- ? Supports Spartan-II device family of FPGA's.
- ? On board Regulators for VCCINT and VCCIO generations.
- ? User selectable configuration modes.
- ? Facility to program through JTAG, Slave serial and Master serial modes.
- ? Onboard PROM support.
- ? Support for different I/O Standards.
- ? High speed interface with other add-on cards.
- ? Capability to use special clock management features of Spartan-II.
- ? A complete I/O bank for user VREF interface, using DB25 connector.
- ? Parallel port interface directly from add-on card of FPGA.

Xilinx		
PLD Family/Architecture	Device Available	Package
Spartan -II	XC2S50, XC2S100, XC2S150, XC2S200	PQ208

Altera FPGA Card

- ? Supports ACEX 1K device family of FPGA's.
- ? On board Regulators for VCCINT and VCCIO generations.
- ? User selectable configuration modes.
- ? Onboard PROM support.
- ? High speed interface with other add-on cards.
- ? Parallel port interface directly from add-on card of FPGA

Altera		
PLD Family/Architecture	Device Available	Package
ACEX 1K	EP1K50, EP1K100	PQ208
Note: Customized daughter cards are also available	for other devices	

List of Application Add-on modules

Here is the list of add-on modules that are supported by "Universal System Development Platform". All cards are made on edge connector of 120 pins.

1. Micro-controller Card

- ? Philips 89C51 RD2 controller.
- ? RS-232 interface.
- ? On board serial EEPROM.
- ? On board serial RTC.
- ? In system serial programmable (ISP).
- ? All the I/Os are accessible to FPGA with connector in between.
- ? Available with standard codes of I2C, timers, data transfer, etc.

2. PIC Micro-controller Card

- ? Microchip 16F877 PIC microcontroller .
- ? In system Programmable.
- ? On board RTC and RS 232 Interface.
- ? Interrupt port.
- ? All ports accessible through edge connector.

3. Memory Add-On Card

- ? Add on module for Memory intensive applications.
- ? Total 2MB data storage capacity.
- ? Four 512KBx 2 SRAM.
- ? Direct high speed interface with FPGA modules.
- ? 70ns of access time.

4. ADC/DAC Add-On card

- ? 4-channel ADC.
- ? 2-channel DAC.
- ? Sampling rate upto 400 KSPS.
- ? 8-bit ADC resolution.
- ? 12 –bit DAC resolution.
- ? 0-5V or +/- 2.5V input voltage.

5. Power Electronics module

- ? IGBT based O/P drive, with current rating upto 10Amps.
- ? Dual high current relays.
- ? Stepper motor controller circuit.
- ? 5 optically isolated O/Ps.
- ? Step down transformer (/100) for line monitoring applications.
- ? Separately available in isolated board.
- ? High current capacity connectors.
- ? Easily interfaced with motors and other circuits.
- ? Optimum choice for Pulse Width Modulation (PWM), motion control, line monitoring & control applications.

6. General Purpose Add-On Card (Free of charge)

- ? General purpose layouts.
- ? Helps in prototyping electronic circuits, with FPGA interface.
- ? Useful in mixed signals designs.

Other Accessories Provided

- ? JTAG Cable for programming of FPGAs.
- ? Serial cable for RS-232 communication from 89c51
- ? PIC programming Cable.
- ? PIC Serial Cable for RS-232 communication.
- ? ADC/DAC cables for interfacing external world signals.
- ? SMPS.
- ? 4x4 membrane keypad with cable.
- ? 16x2 character LCD with cable.
- ? User Manual with instructions, reference designs, examples, sources codes and reference datasheets.
- ? CD-ROM containing user manual, source codes, reference designs and programming softwares.

Note: The above modules comes in a package, many modules can be optional to user.

ni logic Pvt. Ltd holds the rights to modify the product features and specification without notice.



Advantages of Universal System Development Platform over other protoboards

Here are the technological advantages of **USDP** with other protoboards.

- ? Easy system design and development.
- ? Good learning and design implementation platform.
- ? Single platform for multiple technologies.
- ? Stacking of multiple FPGAs which makes multiple FPGA designs development possible.
- ? Higher performance.
- ? Long chain of Add-on modules for all type of applications.
- ? Capable for future advancement and up gradation with new technologies.
- ? Large number of user interface.
- ? General purpose PCB for user circuit interface.

Applications of USDP

- ? Prototyping of FPGA designs
- ? Research and development of high-speed FPGA designs.
- ? Design and development of FPGA based DSP designs and algorithms
- ? Understanding of various FPGA architectures
- ? To train designers how to exploit architectural features of FPGAs from different vendors
- ? Performing a wide range of experiments, by actually downloading the designs into FPGA
- ? Understanding the basics of HDL's and digital logic interface
- ? Robotics and Motion control applications.
- ? Mictrocontroller based applications.
- ? PC controlled design development.
- ? Many more...

Examples:

Temperature controller

USDP is ideal for temperature control applications. Only a temperature sensor has to be interfaced along with its signal conditioning unit(SCU) to ADC. After the digitization of temperature values, user can manipulate the values and control the process using FPGA or the microcontroller cards.

Modules Can be used

PIC card / 89C51 card, FPGA card, General Purpose PCB card, ADC/DAC Card and power module.



Motion Control

USDP is ideal for robotics applications development. As Stepper motors are widely used and plays vital role in robotics environment for implementing arms, handles, moving/rotating mechanisms.

With the use of external power module USDP can prove good platform for motion control applications. With onboard stepper motor controller circuit, AC line step down transformer, relays, optically isolated outputs, etc., power module makes USDP as a complete system development platform.

Modules used

PIC card / 89C51 card, FPGA card, General Purpose PCB card, Power module.



USDP



Infrared Communication

USDP is designed to meet a variety of application needs with distinct advantages that enable the embedded system designer to easily add infrared wireless connectivity. IRDA IC's can be easily assembled on General purpose PCB's and data communication can be achieved using the micro controller or FPGA/CPLD. **Modules Can be used**

PIC card / 89C51 card, FPGA card, General Purpose PCB card, IRDA IC's.



Access control system

Today many applications are developed for security and access controlling. The basic applications of access control can be developed and prototyped on USDP. The basic model consists of keypad interface for password entering, solonoid for door open & close which can be replaced with relays here, user display for welcome notes, menus and messages and protocol for security maintenance.

Modules Can be used

PIC card / 89C51 card, FPGA card, General Purpose PCB card, LCD, Power module and keypad.



PC based applications

Today most of the applications are having PC communication or control. For communication with PC **USDP** provides both **serial** and **parallel** communication. Parallel port connector is provided on FPGA cards, which can used to communicate with PC from its parallel port. Also every microcontroller is equipped with RS-232 communication port for serial communication with PC. Hence USDP is worth getting product for such PC controlled application development.

Modules Can be used

PIC card / 89C51 card, FPGA card.



USDP

8051 microcontroller-based applications

USDP can give excellent support of development and implementation of microcontroller based application. The user gets two top of the line microcontroller development cards based on 89c51RD2 and PIC 16F877. Using these controller cards the designer gets the choice for selecting the controller depending on the application requirement. As 89c51 is based on 8051 architecture the students feels comfortable and gets the exposure for practical design development and also the PIC 16F877 which is RISC based architecture gives them a real learning experience.

Other Digital experiments and Practicals

Design of 8/16/24/32 bit counters & shift registers, Adders and subtractors, 4-bit and 8-bit ALUs, Timer designs IC8254 & IC8253, 8255 PPI design, Micro-processor design development using HDL, All digital logic gates and functions.

Chapter 4 System Diagrams

This chapter has the required system diagrams of USDP.



USDP Base board connectivity



Note: * General purpose PCB and power module can be mapped anywhere on the addon connectors.

FPGA Connectivity

A1	LD0	A2	LD1	B1	LD2	B2	LD3
A3	LD4	A4	LD5	B3	LD6	B4	LD7
A5	LC0	A6	LC1	B5	LC2	B6	LC3
A7	LC4	A8	LC5	B7	LC6	B8	LC7
A9	LB0	A10	LB1	B9	LB2	B10	LB3
A11	LB4	A12	LB5	B11	LB6	B12	LB7
A13	LA0	A14	LA1	B13	LA2	B14	LA3
A15	LA4	A16	LA5	B15	LA6	B16	LA7
A17	100	A18	101	B17	102	B18	103
A19	IO4	A20	105	B19	106	B20	107
A21	108	A22	109	B21	IO10	B22	IO11
A23	IO12	A24	IO13	B23	IO14	B24	IO15
A25	1016	A26	1017	B25	1018	B26	1019
A27	IO20	A28	IO21	B27	1022	B28	IO23
A29	IO24	A30	IO25	B29	1026	B30	IO27
A31	IO28	A32	1029	B31	IO30	B32	IO31
A33	1032	A34	IO33	B33	IO34	B34	IO35
A35	1036	A36	IO37	B35	IO38	B36	1039
A37	IO40	A38	IO41	B37	IO42	B38	IO43
A39	IO44	A40	IO45	B39	1046	B40	IO47
A41	IO48	A42	1049	B41	1050	B42	IO51
A43	IO52	A44	IO53	B43	IO54	B44	IO55
A45	1056	A46	1057	B45	1058	B46	1059
A47	1060	A48	1061	B47	1062	B48	1063
A49	GND	A50	GND	B49	GND	B50	GND
A51	GND	A52	GND	B51	GND	B52	GND
A53	+5V	A54	+5V	B53	+5V	B54	+5V
A55	+5V	A56	+5V	B55	+5V	B56	+5V
A57	-5V	A58	-5V	B57	-5V	B58	-5V
A59	-5V	A60	-5V	B59	-5V	B60	-5V

Add-On Connector Pin Out Table

A1-A14 and B1-B14 are LEDs

IOO to **IO63** are general-purpose bus on the connector. All cards are mapped on above layout, kindly refer the below pin out detail for individual card layout.

	PIC	Memory	LEDs		PIC	Memory	LEDs	_		
A1	AN0/RA0	A16	LD0	A2	AN1/RA1	A17	LD1	$\left \right\rangle$		
A3	AN2/RA2	A18	LD4	A4	AN3/RA3	D0	LD5			
A5	AN4/RA4	D1	LC0	A6	AN4/RA5	D2	LC1			PIC, Memory, LEDs
A7	RD0	D3	LC4	A8	RD1	D4	LC5		\geq	share the connections
A9	RD2	D5	LB0	A10	RD3	D6	LB1	(
A11	RD4	D7	LB4	A12	RD5	RD/	LB5			
A13	RD6	WR/	LA0	A14	RD7	A19	LA1			
A15	MCLR	A20	LA4	A16			LA5			
	ADC/DAC				ADC/DAC					
A17	D0			A18	D1					
A19	D4			A20	D5					
A21	D8			A22	D9					$\Delta DC/D\Delta C$ Card
A23	EN1			A24	EN2			1	\geq	connections
A25	DB0			A26	DB1					CONTECTIONS
A27	DB4			A28	DB5]		
A29	INT			A30	RDY					
	89c51				89c51					
A31	AD0			A32	AD2					
A33	AD4			A34	AD6					
A35	A8			A36	A10					
A37	A12			A38	A14			(89c51 Card
A39	PORT1_1			A40	PORT1_3			1 1		connections
A41	PORT1_5			A42	PORT1_7					
A43	INT0			A44	INT1					
A45	ALE			A46	FRST			レ		
A47		1060		A48		1061				
								1		
D1	PIC	Memory	LEDS	Da	PIC	Memory	LEDS			
B1	PIC RC0	Memory A0	LEDs	B2	PIC RC1	Memory A1	LEDs			
B1 B3	PIC RC0 RC2	Memory A0 A2	LEDs LD2 LD6	B2 B4	PIC RC1 RC3	Memory A1 A3	LEDs LD3 LD7			
B1 B3 B5	PIC RC0 RC2 RC4	Memory A0 A2 A4	LEDs LD2 LD6 LC2	B2 B4 B6	PIC RC1 RC3 RC5	Memory A1 A3 A5	LEDs LD3 LD7 LC3			PIC, Memory, LEDs
B1 B3 B5 B7	PIC RC0 RC2 RC4 RC6	Memory A0 A2 A4 A6	LEDs LD2 LD6 LC2 LC6	B2 B4 B6 B8	PIC RC1 RC3 RC5 RC7	Memory A1 A3 A5 A7	LEDs LD3 LD7 LC3 LC7		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9	PIC RC0 RC2 RC4 RC6 RB0/INT	Memory A0 A2 A4 A6 A8	LEDs LD2 LD6 LC2 LC6 LB2	B2 B4 B6 B8 B10	PIC RC1 RC3 RC5 RC7 RB1	Memory A1 A3 A5 A7 A9 A11	LEDS LD3 LD7 LC3 LC7 LB3		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 DD4	Memory A0 A2 A4 A6 A8 A10	LEDs LD2 LD6 LC2 LC6 LB2 LB6	B2 B4 B6 B8 B10 B12	PIC RC1 RC3 RC5 RC7 RB1 RD3 PD5	Memory A1 A3 A5 A7 A9 A11	LEDs LD3 LD7 LC3 LC7 LB3 LB7		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 DD(Memory A0 A2 A4 A6 A8 A10 A12	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2	B2 B4 B6 B8 B10 B12 B14	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 DD7	Memory A1 A3 A5 A7 A9 A11 A13	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B8 B10 B12 B14 B16	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15 B17	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B8 B10 B12 B14 B16	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7		\succ	PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B10	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D4	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B10 B12 B14 B16 B18 B20	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7			PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B10 B12 B14 B16 B18 B20 B22	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7			PIC, Memory, LEDs share the connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B17 B19 B21 B22	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 NP2	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B24 B24	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 NP2	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B13 B15 B17 B19 B21 B23 B25 B27	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB2	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B17 B19 B21 B23 B25 B27 B20	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB6 CS	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28 B20	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 PD	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB6 CS S 90c51	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28 B28 B30	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD 90c51	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29 B21	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB6 CS 89c51 AD1	Memory A0 A2 A4 A6 A8 A10 A12 A14 -	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28 B28 B30 B22	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD B7 RD 89c51 AD3	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B13 B15 B17 B19 B21 B23 B25 B27 B29 B31 B23	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB6 CS 89c51 AD1 AD5	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B10 B12 B14 B16 B18 B20 B22 B24 B24 B26 B28 B30 B32 B32	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD B7 RD 89C51 AD3 AD7	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B17 B19 B21 B23 B25 B27 B29 B31 B33 B25	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A0	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B24 B26 B28 B30 B32 B34 B34 B26	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD B3 B7 RD 89c51 AD3 AD7 A11	Memory A1 A3 A5 A7 A9 A11 A13 A15 	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B31 B33 B35 B27	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A9 A13	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28 B28 B28 B30 B32 B34 B36 B32 B32	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD B3 DB7 RD 89c51 AD3 AD7 A11 A15	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29 B31 B33 B35 B37 B37 B29	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 D6 D10 A0 DB2 B6 CS 89c51 AD1 AD5 A9 A13 POPT1 2	Memory A0 A2 A4 A6 A8 A10 A12 A14 	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B10 B12 B14 B16 B14 B16 B20 B22 B24 B26 B28 B28 B28 B30 B32 B34 B36 B38 B30 B38 B40	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB7 ADC/DAC D3 D7 D11 A1 DB3 D7 D11 A1 DB3 DB7 RD 89c51 AD3 AD7 A11 A15 POPT1 4	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections 89c51 Card
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B27 B29 B27 B29 B31 B33 B35 B37 B39 B41	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A9 A13 PORT1_2 PORT1_2	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B10 B12 B14 B16 B14 B16 B20 B22 B24 B24 B26 B28 B28 B30 B30 B32 B34 B36 B38 B40 B42	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 D7 D11 A1 DB3 DB7 RD 89c51 AD3 AD7 A11 A15 PORT1_4 PORT1_4 PORT1_4	Memory A1 A3 A5 A7 A9 A11 A13 A15 	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections 89c51 Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29 B31 B33 B35 B37 B39 B41 B43	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A9 A13 PORT1_2 PORT1_6 T0	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B18 B20 B22 B24 B26 B28 B26 B28 B28 B30 B22 B24 B26 B28 B30 B22 B24 B26 B28 B30 B22 B24 B26 B28 B30 B22 B24 B24 B26 B28 B26 B28 B28 B20 B22 B24 B24 B24 B24 B24 B24 B24 B24 B24	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 D7 D11 A1 DB3 D87 RD 89c51 AD3 AD7 A11 A15 PORT1_4 PORT1_8 T1	Memory A1 A3 A5 A7 A9 A11 A13 A15 	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections 89c51 Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29 B31 B33 B35 B37 B39 B41 B43 B45	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RB6 ADC/DAC D2 D6 D10 A0 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A9 A13 PORT1_2 PORT1_6 T0 RD 51	Memory A0 A2 A4 A6 A8 A10 A12 A14	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B14 B20 B22 B24 B22 B24 B26 B28 B28 B30 B22 B24 B28 B28 B30 B22 B24 B28 B28 B30 B22 B24 B28 B28 B30 B22 B24 B24 B26 B28 B28 B28 B20 B22 B24 B24 B24 B24 B24 B24 B24 B24 B24	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 RB7 ADC/DAC D3 D7 D11 A1 DB3 DB7 RD B3 DB7 RD B3 CD RD B7 RD B3 DB7 RD B7 RD B3 DB7 RD B7 RD B7 RD B7 RD B3 DB7 RD B7 RD B7 RD B7 RD B3 DB7 RD B7 RD B7 RD B3 DB7 RD B7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 DB7 RD B3 RD B3 RD B3 RD B3 RD B3 RD B7 RD B3 RD B7 RD RD RD RD RD RD RD RD RD RD RD RD RD RD RD RD RD R	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections 89c51 Card connections
B1 B3 B5 B7 B9 B11 B13 B15 B17 B19 B21 B23 B25 B27 B29 B27 B29 B31 B33 B35 B37 B39 B41 B43 B45 B47	PIC RC0 RC2 RC4 RC6 RB0/INT RB2 RB4 RD0 D10 A0 D2 D6 D10 A0 DB2 DB6 CS 89c51 AD1 AD5 A9 A13 PORT1_2 PORT1_6 T0 RD_51	Memory A0 A2 A4 A6 A8 A10 A12 A14 	LEDs LD2 LD6 LC2 LC6 LB2 LB6 LA2 LA6 	B2 B4 B6 B8 B10 B12 B14 B16 B14 B20 B22 B24 B24 B26 B28 B28 B28 B30 B32 B34 B33 B34 B36 B38 B40 B42 B44 B46 B49	PIC RC1 RC3 RC5 RC7 RB1 RD3 RB5 AD2/DAC D3 D7 D11 A1 DB3 D7 D11 A1 DB3 DB7 RD 89c51 AD3 AD7 A11 A15 PORT1_4 PORT1_8 T1 WR_51	Memory A1 A3 A5 A7 A9 A11 A13 A15	LEDs LD3 LD7 LC3 LC7 LB3 LB7 LA3 LA7 			PIC, Memory, LEDs share the connections ADC/DAC Card connections 89c51 Card connections

Add-On Connector Layout Detail With Adaptor Module Connections

Chapter 5 Pin Assignment

This chapter will go through the pin assignment of FPGAs with various add-on modules. The three PCI connectors on board share the bus. All modules can be plugged on any one of the connector, as the connectors are sharing the common bus.

Here is the add-on connector pin out with Altera ACEX-1K FPGA and Xilinx Spartan-II FPGA.

	Acex-1k	Spartan-II									
A1	36	34	A2	37	35	B1	38	36	B2	39	37
A3	40	41	A4	41	42	B3	44	43	B4	45	44
A5	46	45	A6	47	46	B5	53	47	B6	54	48
A7	55	49	A8	56	57	B7	57	58	B8	58	59
A9	60	60	A10	61	61	B9	63	62	B10	64	63
A11	65	67	A12	67	68	B11	68	69	B12	69	70
A13	70	71	A14	71	73	B13	73	74	B14	74	75
A15	75	81	A16	85	82	B15	86	83	B16	87	84
A17	92	87	A18	90	86	B17	89	3	B18	88	4
A19	96	94	A20	95	90	B19	94	89	B20	93	88
A21	101	98	A22	100	97	B21	99	96	B22	97	95
A23	111	102	A24	104	101	B23	103	100	B24	102	99
A25	115	111	A26	114	110	B25	113	109	B26	112	108
A27	121	115	A28	120	114	B27	119	113	B28	116	112
A29	127	122	A30	126	121	B29	125	120	B30	122	119
A31	133	127	A32	132	126	B31	131	125	B32	128	123
A33	139	134	A34	136	133	B33	135	132	B34	134	129
A35	143	139	A36	142	138	B35	141	136	B36	140	135
A37	149	146	A38	148	142	B37	147	141	B38	144	140
A39	159	150	A40	158	149	B39	157	148	B40	150	147
A41	163	163	A42	162	162	B41	161	152	B42	160	151
A43	168	167	A44	167	166	B43	166	165	B44	164	164
A45	173	174	A46	172	173	B45	170	172	B46	169	168
A47	177	179	A48	176	178	B47	175	176	B48	174	175

Note: Both the FPGAs share the bus on add-on connector. User has to take care, that **no two pins are defined as output**, as in that case there would be short on the bus and may damage FPGA I/Os.

User can define the following combination of FPGAs shared I/Os;

FPGA1	FPGA2	
Input	Input	Allowed
Output	Input	Allowed
Input	Output	Allowed
Output	Output	Not allowed

ACEX1K FPGA Pin detail

Device: EP1kxx PQ208

Clock &	Reset
Clock	79
_	

Reset 180

Note: Reset is active LOW.

	Configurable I/Os						
LA7	36	LB7	46	LC7	60	LD7	70
LA6	37	LB6	47	LC6	61	LD6	71
LA5	38	LB5	53	LC5	63	LD5	73
LA4	39	LB4	54	LC4	64	LD4	74
LA3	40	LB3	55	LC3	65	LD3	75
LA2	41	LB2	56	LC2	67	LD2	85
LA1	44	LB1	57	LC1	68	LD1	86
LA0	45	LB0	58	LC0	69	LD0	87

7 Segme	ents	Display E	Keypad Header				
segA	31	DISP1	18	SL0	8	RL0	3
segB	30	DISP2	16	SL1	13	RL1	11
segC	29	DISP3	17	SL2	9	RL2	7
segD	28	DISP4	15	SL3	14	RL3	12
segE	27						
segF	26						
segG	25						
segDP	24						

Parallel Port Connector (DB-25)						
Par1	205	Par10	193			
Par2	203	Par11	192			
Par3	202	Par12	191			
Par4	200	Par13	190			
Par5	199	Par14	189			
Par6	198	Par15	187			
Par7	197	Par16	186			
Par8	196	Par17	179			
Par9	195					
Pin 18 – 2	5 of DB-25	5 connector ar	e around			

Note: Both the FPGAs share the above I/Os. User has to take care, that **no two pins are defined as output**, as in that case there would be short on the bus and may damage FPGA I/Os. Also tri-state the unused pins of FPGA by changing settings of your EDA tool while pin locking.

User can define the following combination of FPGAs shared I/Os;

FPGA1	FPGA2	
Input	Input	Allowed
Output	Input	Allowed
Input	Output	Allowed
Output	Output	Not allowed

Spartan-II FPGA Pin detail

Device: XC2Sxx PQ208

Clock and Reset				
Clock (GCK0)	80			
GCK2	182			
GCK3	185			
Reset	5			

Note: **Reset** is active **LOW**.

	Configurable I/Os						
LA7	34	LB7	45	LC7	60	LD7	71
LA6	35	LB6	46	LC6	61	LD6	73
LA5	36	LB5	47	LC5	62	LD5	74
LA4	37	LB4	48	LC4	63	LD4	75
LA3	41	LB3	49	LC3	67	LD3	81
LA2	42	LB2	57	LC2	68	LD2	82
LA1	43	LB1	58	LC1	69	LD1	83
LA0	44	LB0	59	LC0	70	LD0	84

7 Segments		Display E	Keypad Header				
segA	33	DISP1	17	SL0	8	RL0	6
segB	31	DISP2	18	SL1	15	RL1	10
segC	30	DISP3	20	SL2	9	RL2	7
segD	29	DISP4	21	SL3	16	RL3	14
segE	27						
segF	24						
segG	23						
segDP	22						

Parallel Port Connector (DB-25)						
Par1	206	Par10	192			
Par2	205	Par11	191			
Par3	204	Par12	188			
Par4	202	Par13	187			
Par5	201	Par14	203			
Par6	199	Par15	200			
Par7	195	Par16	189			
Par8	194	Par17	181			
Par9	193					
Pin 18 – 2	Pin 18 – 25 of DB-25 connector are ground					

Note: Both the FPGAs share the above I/Os. User has to take care, that **no two pins are defined as output**, as in that case there would be short on the bus and may damage FPGA I/Os.

User can define the following combination of FPGAs shared I/Os;

FPGA1	FPGA2	
Input	Input	Allowed
Output	Input	Allowed
Input	Output	Allowed
Output	Output	Not allowed

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A1	AN0/RA0	36	34	A2	AN1/RA1	37	35
A3	AN2/RA2	40	41	A4	AN3/RA3	41	42
A5	AN4/RA4	46	45	A6	AN4/RA5	47	46
A7	RD0	55	49	A8	RD1	56	57
A9	RD2	60	60	A10	RD3	61	61
A11	RD4	65	67	A12	RD5	67	68
A13	RD6	70	71	A14	RD7	71	73
A15	MCLR/	75	81				
A15	MCLR/	75	81	A14	KD7	/	15

PIC 16F877 Micro controller Pin details*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B1	RC0	38	36	B2	RC1	39	37
B3	RC2	44	43	B4	RC3	45	44
B5	RC4	53	47	B6	RC5	54	48
B7	RC6	57	58	B8	RC7	58	59
B9	RB0/INT	63	62	B10	RB1	64	63
B11	RB2	68	69	B12	RD3	69	70
B13	RB4	73	74	B14	RB5	74	75
B15	RB6	86	83	B16	RB7	87	84

MCLR/ is active LOW reset.

*Refer the device datasheet for pin description and functioning.

Note: PIC micro controller and SRAM memory module share the same bus, kindly use one module one at a time on the 3 slots.

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A1	A16	36	34	A2	A17	37	35
A3	A18	40	41	A4	D0	41	42
A5	D1	46	45	A6	D2	47	46
A7	D3	55	49	A8	D4	56	57
A9	D5	60	60	A10	D6	61	61
A11	D7	65	67	A12	RD/	67	68
A13	WR/	70	71	A14	A19	71	73
A15	A20	75	81				

SRAM Module Pin details*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B1	A0	38	36	B2	A1	39	37
B3	A2	44	43	B4	A3	45	44
B5	A4	53	47	B6	A5	54	48
B7	A6	57	58	B8	A7	58	59
B9	A8	63	62	B10	A9	64	63
B11	A10	68	69	B12	A11	69	70
B13	A12	73	74	B14	A13	74	75
B15	A14	86	83	B16	A15	87	84

*Refer the device datasheet for pin description and functioning.

Note: PIC micro controller and SRAM memory module share the same bus, kindly use one module one at a time on the 3 slots.

ADC/DAC Module Pin Detail*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A17	D0	92	87	A18	D1	90	86
A19	D4	96	94	A20	D5	95	90
A21	D8	101	98	A22	D9	100	97
A23	EN1	111	102	A24	EN2	104	101
A25	DB0	115	111	A26	DB1	114	110
A27	DB4	121	115	A28	DB5	120	114
A29	INT	127	122	A30	RDY	126	121

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B17	D2	89	3	B18	D3	88	4
B19	D6	94	89	B20	D7	93	88
B21	D10	99	96	B22	D11	97	95
B23	A0	103	100	B24	A1	102	99
B25	DB2	113	109	B26	DB3	112	108
B27	DB6	119	113	B28	DB7	116	112
B29	CS	125	120	B30	RD	122	119

ADC header Detail

DAC O/P (J1)

Pin1	DAC Channel 1
Pin 2	DAC Channel 2
Pin 3	Ground

ADC I/P (J2)

	1 /
Pin1	ADC Channel 1
Pin 2	ADC Channel 2
Pin 3	ADC Channel 3
Pin 4	ADC Channel 4
Pin 5	Ground

*Refer the device datasheet for pin description and functioning.

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A31	AD0	133	127	A32	AD2	132	126
A33	AD4	139	134	A34	AD6	136	133
A35	A8	143	139	A36	A10	142	138
A37	A12	149	146	A38	A14	148	142
A39	PORT1_1	159	150	A40	PORT1_3	158	149
A41	PORT1_5	163	163	A42	PORT1_7	162	162
A43	INT0	168	167	A44	INT1	167	166
A45	ALE	173	174	A46	FRST	172	173

89c51 Module Pin Detail*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B31	AD1	131	125	B32	AD3	128	123
B33	AD5	135	132	B34	AD7	134	129
B35	A9	141	136	B36	A11	140	135
B37	A13	147	141	B38	A15	144	140
B39	PORT1_2	157	148	B40	PORT1_4	150	147
B41	PORT1_6	161	152	B42	PORT1_8	160	151
B43	T0	166	165	B44	T1	164	164
B45	RD_51	170	172	B46	WR_51	169	168

FRST is active HIGH reset.

*Refer the device datasheet for pin description and functioning.

Power Module Pin Detail

Power module is independent of USDP baseboard, and can me mapped to any of the I/Os of the adaptors available.

Here by are the header details of the power module.

JP1 (AC I/P)		
Pin No. Signal		
1	Line	
2	Neutral	

JP2 (DC O/P)			
Pin No. Signal			
1	VDC		
2	AGND (Analog ground)		

JP4 (AC Line, transformer)		
Pin No. Signal		
1	Primary1	
2	Primary2	

JP5 (Step O/P / 100)			
Pin No. Signal			
1	Secondary1		
2	Common		
3	Secondary2		

JP8 (Stepper Motor)			
Pin No.	Signal		
1	W1		
2	W2		
3	W3		
4	W4		
5	DGND (digital ground)		

JP6 (Optically Isolated O/Ps)			
Pin No.	Signal		
1	OP1		
2	OP2		
3	OP3		
4	OP4		
5	OP5		
6	IGND (Isolated ground)		
7	DGND (Digital ground)		

JP7 (I	Relay1)	JP9 (Relay2)	
Pin No.	Signal	Signal	
1	Common	Common	
2	NO	NC	
3	NO	NC	

J2 (IGBT)				
Pin no.	Signal	Description		
1	G	Gate		
2	С	Collector		
3	E	Emitter		

JP10 (FPGA Interface)						
Pin No.	Signal	Acex 1K	Spartan-II	Description		
1	IP1	205	206	Isolated I/P 1		
2	IP2	203	205	Isolated I/P 2		
3	IP3	202	204	Isolated I/P 3		
4	IP4	200	202	Isolated I/P 4		
5	IP5	199	201	Isolated I/P 5		
6	RELAY1	198	199	Relay control 1		
7	RELAY2	197	195	Relay control 2		
8	W2	196	194	Winding 2 control		
9	W1	195	193	Winding 1 control		
10	W4	193	192	Winding 4 control		
11	W3	192	191	Winding 3 control		
12-16	DGND (Digital ground)					

Note: The above pin assignment is for the power electronics module cable provided along with USDP. Kindly use the same cable to interface the power electronics module.

Chapter 6 <u>Jumpers and Headers</u> Pin description and Settings

In this chapter we will have a look on the jumper settings of all the USDP modules.

USDP Baseboard

Clock (J	IP1)
----------	------

Pin 1	Clock
Pin 2	GCK0
Pin 3	Ground

To use the oscillator clock short pin 1 and 2. It is recommended to ground the GCK0 during programming.

JPDIS

These four jumpers are used to disable the 7-segment displays. Just remove the jumpers to isolate to displays from FPGAs.

JPA, JPB, JPC & JPD

These jumpers are used to select the I/O mode of the LEDs.

Total 32 configurable I/Os are provided on the USDP baseboard. To configure them as I/P user has to insert the jumper in between, and remove the jumper to make them as O/P.

The switch configured as input to FPGA should not be connected to pin which is configure as output in FPGA, else there would be a hardware short, and may cause damage to the device.

User has to take this care during VHDL coding and pin assignment.

Power Headers (J8 & J9)

There are power supply headers; users can connect the SMPS to any one of these headers. Two headers are provided just for the sake of convenience for connecting the SMPS from any of the directions.

SLOT SEL (JP3)

This is slot selector header. User can use the provided selector card for selecting the slot.

Slot 1	Slot 2
Short 1-2	Short 2-3

PLD SEL (JP2)

This is PLD vendor selector header. User can use the provided selector card for selecting the PLD vendors.

Xilinx	Altera
Short 1-2	Short 2-3

J7 (Keypad)

Pin No.	Signal	Pin No.	Signal
1	VCC	2	GND
3	SL3	4	SL2
5	SL1	6	SL0
7	RL3	8	RL2
9	RL1	10	RL0

Programming Port (J6)

User has to connect provided programming cable to this port and PC's parallel for programming of FPGAs.

Xilinx FPGA module

Programming Mode selection Switch (U7)

Modes	MO	M1	M2
ITAG	1	0	1
Slave Serial	1	1	1
Master Serial	0	0	0

Mode selection jumpers (JP4, JP5, JP6, JP7)

Serial	JTAG	
Short 1-2	Short 2-3	

PROM BYPASS (J3 & J4)

To Chain PROM	To Bypass PROM
Short 1 –2	Short 2-3

Shorting 1-2 will bring the PROM in chain with FPGA. In this case user can configure the PROM and use for programming file storage.

Shorting 2-3 will remove the PROM from chain, and only FPGA would be connected to programming port.

Global Clock Buffers

Pin 1	GCK2	GCK3
Pin 2	GND	GND

Note: These buffers are extra from GCK0.

Altera FPGA Module

Programming Mode selection jumpers (J3, J8)

	MSEL1	MSEL0
Passive Serial (PS)	0	0
JTAG	0	1

For logic low short 2-3 and for logic short 1-2 for logic High

Mode selection jumpers (J4, J5, J6, J7)

Serial	JTAG
Short 2-3	Short 1-2

PROM BYPASS (J1 & J2)

To Chain PROM To Bypass PROM

Short 1 – 2 Short 2-3

Shorting 1-2 will bring the PROM in chain with FPGA. In this case user can configure the PROM and use for programming file storage.

Shorting 2-3 will remove the PROM from chain, and only FPGA would be connected to programming port.

89c51 Module

Square Wave o/p (JP3)

Pin1	1Hz clock from RTC.
Pin 2	Ground

Interrupt/Timer (JP4)

Pin1	Interrupt 0 (INT0) I/P
Pin 2	Interrupt 1 (INT1) I/P
Pin 3	Timer 1 O/P
Pin 4	Timer 0 O/P

JP1 & JP2

JP1	Port1_1	SDA
JP2	Port1_2	SCLK

JP1 & JP2 are selection jumper for serial clock and data from RTC and EEPROM. Both these signals are shared with the base bus of USDP. So if you are these two ports of 89c51 with the base bus then remove the jumpers from the JP1 & JP2 to isolate them from the base bus.

PIC Microcontroller Module

Programming jumper

JP9, JP10 & JP11 are programming selection jumpers. User has to short 1-2 during programming and after programming the PIC controller, short 2-3.

JP1

Pin1	1Hz clock from RTC.
Pin 2	Ground

JP2	
Pin1	Ra4
Pin 2	Rb0/int
Pin 3	Ground

JP3	
D' 1	

Pin1	PCRX
Pin 2	PCTX
Pin 3	Ground

RX2 (JP5)

Port RC7 is used for RX channel of RS-232, also this port pin is shard with base bus and by removing this jumper you can isolate the RS-232 from the base bus.

TX (JP6)

Port RC6 is used for TX channel of RS-232, also this port pin is shard with base bus and by removing this jumper you can isolate the RS-232 from the base bus.

SDA (JP7)

Port RC4 is used for serial data for RTC, also this port pin is shard with base bus and by removing this jumper you can isolate the RTC from the base bus.

SCLK (JP8)

Port RC3 is used for clock for RTC, also this port pin is shard with base bus and by removing this jumper you can isolate the RTC from the base bus.

Pin1AN0Pin 2AN1Pin 3AN2Pin 4AN3Pin 5AN4Pin 6AN5Pin 7AN6Pin 8AN7Pin 9Ground

Analog I/Ps (Header for inbuilt ADC I/Ps for PIC controller)

JP9, JP10 & JP11

During programming the PIC controller, **short pins 2-3**, and after programming the controller, **short pins 1-2** for using with FPGA.

ADC/DAC Module

DAC O/P (J1)

· · · ·		
Pin1	DAC Channel 1	
Pin 2	DAC Channel 2	
Pin 3	Ground	

ADC I/P (J2)

Pin1	ADC Channel 1
Pin 2	ADC Channel 2
Pin 3	ADC Channel 3
Pin 4	ADC Channel 4
Pin 5	Ground

JP2

This jumper is for setting the reference voltage. This module has onboard reference voltage generation, to use that short 2-3.

Power Electronics Module

As there is no settings in this module, for pin description and header information, please refer pin assignment chapter

General purpose PCB

All the I/Os of base bus are brought on this PCB. User can build their custom circuit on the board and interface with any of the module pins with the help of headers provided on the PCB. The I/O numbering has been indicated nearby the headers, users can refer the pin assignment chapter for further information about the bus description and numbering.

Also this PCB contains headers for +5V, -5V and ground signals onboard.

JP6 (LCD Header)

Pin No.	Signal	Pin No.	Signal
1	GND	2	+5V
3	NC	4	RS
5	NC	6	EN
7	D0	8	D1
9	D2	10	D3
11	D4	12	D5
13	D6	14	D7
15	NC	16	NC

JP1 (LCD I/Ps)

Pin No.	Signal	Pin No.	Signal
1	D0	2	RS
3	D2	4	EN
5	D4	6	D1
7	D6	8	D3
9	D7	10	D5

Note: The LCD module pins are brought on a separate header (JP1), users can interface LCD module to any of the modules I/Os with the provided cable.

Chapter 7 Precautions

Please follow the guidelines below while using USDP and take the mentioned precautions.

General

- ? Verify the power ON LED status after applying power to the trainer.
- ? Connect the 25 pin D connector of the cable to the trainer only after confirming the above
- ? Insert the modules vertically in the slots; slowly and with care, as inserting the modules in angular form will cause damage to connector pins.
- ? Do not press the modules after inserting them on board, else this can cause damage to connector pins.
- ? During downloading make sure that the jumper selections are proper
- ? Before implementation, it is necessary to lock the pins in user constraint file (UCF) as per the mode selected.
- ? For downloading the bit stream, the downloading circuit requires a stable supply; hence it is recommended to use the power supply supplied with the trainer only.

FPGA modules

- ? Insert the module vertically in the slots.
- ? Do not touch the FPGA while handling the module, as it may get damage due to static charge on your body.
- ? Do program the FPGA as per the Configuration chapter.
- ? While stacking both FPGA modules, do not make the same pin location/bus as outputs.

Memory module

? As the memory module shares the bus with PIC controller, it is recommended **not to use** PIC controller while using memory module.

Power Electronics module

- ? If working on the high voltages, then it is recommended to use gloves while handling module.
- ? Keep the hands off the terminal blocks (AC & DC) as you may get shock from it.
- ? Do not exceed the ratings of devices, else it may cause damage to them.

ADC/DAC module

- ? Apply the analog signals in the range of specifications.
- ? Step down the high voltage/current signals.

Chapter 8 Using EDA tools

USDP can works on various technologies altogether. The currently supported technologies are VLSI, microcontroller and DSP.

All these technologies have different design flows and software tools for development purposes. For VLSI technology, USDP works on Altera and Xilinx devices right now, and RISC & 89c51 controllers.

To work on the following provided devices, this chapter will cover the design flow and methodology of EDA tools required for building applications on USDP.

We will cover the design flow following technologies;

- ? PLD design flow
- ? Micro-controller design flow

PLD Design Flow


Microcontroller Design Flow



Using Xilinx ISE Series Software

In this section we will see how a project can be created in Xilinx and Altera EDA tools, and how we can proceed to use USDP to perform our experiments using FPGAs. We take the example of 3:8 decoder with enable inputs and implement on both vendor devices.

Design flow for Xilinx ISE series softwares

Step 1: Open ISE webpack software.

Step 2: Create new project

New Project		×
Project Name: decoder	Project Location: E:\kit_test_projects\USE)P_Demo
Property Name		Value
Device Family	Spartan	2
Device	xc2s200	I
Package	pq208	
Speed Grade	-5	
Design Flow	XST VH)L
	OK Cancel	Help

Step 3: Go to project menu and select new source



Step 4: Select VHDL source file, name it decoder, click next, and enter entity I/Os as Add, en1, en2, en3 & Y.

efine ¥HDL Source				X
Entity Name dec	coder			
Architecture Name Bel	navioral			
Port Name	Direction	MSB	LSB	
Add	in	2	0	
en1	in			
en2	in			
en3	in			
Υ	out	7 🚖	0	
	in			-
			1	
<	Back Next>	Cance	el Help	

Step 5: Write VHDL code for decoder.



Step 6: Create new source file for implementation constraint file. Name it decoder_UCF, and associate with the corresponding design file.

New	×
User Document VHDL Module Coregen IP Schematic VHDL Library VHDL Package VHDL Test Bench Test Bench Waveform BMM File MEM File Implementation Constraints File State Diagram	File Name: decoder_uclf Location: E:\kit_test_projects\USDP_Der
< Back Next >	Cancel Help

Step 7: To assign the pin location of the design entity, open the UCF file, which in turn will open the constraint editor where we have to lock the I/Os of design to a particular pin number.

💫 Xilinx - Project Navigator - E:\kit_test_projects\USDP_Demo\decoder\decoder.npl - [decoder]		_ 8 ×
File Edit View Project Source Process Window Help		» <u>– b ×</u>
	1 % % % S	
Sources in Project 1 libbrary IEEE; Ibbrary IEEE; 1 libbrary IEEE; Image:	address i/ps active low enable active low enable active high enable decoder o/p, active high	n
Processes for Current Source: Properties Properties Properties Processes for Current Source: Processes for Current)' and en3='l') else	×
JHDPARSE - VHDL/Verilog Parser. ISE 5.11 Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.		4
Scanning decoder.vhd Scanning decoder.yhd Writing decoder.jhd. JHDPARSE complete - 0 errors, 0 warnings.		
		¥
	Le opication	
Upen the selected source	LN 22 COI 35	

Step 8: Once the constraint editor is opened, goto ports tab; and assign the pins by referring the Pin assignment chapter.

Eg: net add<0> loc =p84;

dd<0> dd<1>	INPUT	1 70			
dd<1>		p/U		N/A	
	INPUT	p69		N/A	
dd<2>	INPUT	p68		N/A	
11	INPUT	p57		N/A	
n2	INPUT	p58		N/A	
13	INPUT	p59		N/A	
<0>	OUTPUT	p84	N/A		
<1>	OUTPUT	p83	N/A		
<2>	OUTPUT	p82	N/A		
⇒	OUTPUT	p81	N/A		
<4>	OUTPUT	p75	N/A		
\$	OUTPUT	p74	N/A		
<6>	OUTPUT	p73	N/A		
<7>	OUTPUT	p71	N/A		
I/O Configuration Options Prohibit I/O Locations	Group Name:		Create Group Pad to Setup Clock to Pad		
Global Ports	Advanced	Misc			
ET "add<1>" LOC = "p69"; ET "add<0>" LOC = "p70";					
ET "en1" LOC = "p57";					
ET "enz" LOC = "p58";					

Step 9: Save the UCF file and come back to project navigator. Now selecting the **decoder** design file, run the **synthesis** process, there after **Implementation**, which in turn will place and route the design on the target FPGA.



Step 10: Now we need to generate the programming file to configure the FPGA. The Xilinx FPGA supports, JTAG, Slave-serial, and Master-serial (through PROM) for configuration. User has to set proper mode during the programming file generation, else FPGA won't be configured.

For this right click on the Generate Programming File option and click properties on the opened menu.



Step 11: In the opened window, go to startup-options, and select JTAG clock in FPGA Start-Up Clock option. This in turn will generate the programming file for JTAG mode programming.

To program the FPGA in **Slave Serial** mode, user has to select **CCLK clock** instead of JTAG Clock in **the startup-options**.

Property Name	Va	alue
FPGA Start-Up Clock	JTAG Clock	-
Enable Internal Done Pipe	CCLK	
Done (Output Events)	User Clock	
Enable Outputs (Output Events)	JTAG Clock	
Release Set/Reset (Output Events)	 Default (6)	
Release Write Enable (Output Events)	Default (6)	
Release DLL (Output Events)	 Default (NoWa	it)
Drive Done Pin High		

Step 12: Now run the **Generate Programming File** option, which will in turn generate the programming file for FPGA configuration. Left click on the **Generate Programming File** option, which will show the programmer below, named as Configure Devices (iMPACT).

File Edit View Project Source Process Window Help Sources in Project	🗞 Xilinx - Project Navigator - E:\kit_test_projects\USDP_Demo	\deco	oder\decoder.npl - [decoder]	_ 8 ×
Image: Sources in Project Image: Sources in Project Image: Sources in Project Image: Source in Source Image: Source in Project Image: Source in Studie Image: Source in Studie Image: Source in Studie Image: Source in Source Image: Source	File Edit View Project Source Process Window Help			» <u>- 8 ×</u>
<pre>Sources in Project Gecoder Gecode</pre>		ъ 🛛 🖁	? №	
	Sources in Project:	1 2 3 4 6 7 8 9 0 10 11 12 13 14 15 16 17 18 19 20 21 22	<pre>library TEEE; use IEEE.STD_LOGIC_l164.ALL; use IEEE.STD_LOGIC_ONSIGNED.ALL; entity decoder is Port (Add : in integer range 0 to 7; address i/ps en1 : in std_logic; active low enable en2 : in std_logic; active low enable en3 : in std_logic; active low enable en3 : in std_logic; active low enable en3 : in std_logic; active high enable r : out bit_vector(7 downto 0)); decoder o/p, active high end decoder; architecture Behavioral of decoder is begin Y <= "000000001" sll add when (enl='0' and en2='0' and en3='1') else (others=>'0'); end Behavioral;</pre>	×
	Launching Application for process "Conf: Launching Application for process "Conf:	.gure .gure	= Device (iMPACT)". = Device (iMPACT)".	-
Launching Application for process "Configure Device (iMPACT)". Launching Application for process "Configure Device (iMPACT)".	Console Find in Files			
Launching Application for process "Configure Device (iMPACT)". Launching Application for process "Configure Device (iMPACT)".	For Help, press F1		In 16 Col 36	

Step 13: Open the properties of iMPACT programmer, and set the **port** as **LPT1**, **configuration mode** as **Boundary Scan** for JTAG or PROM configuration. **Slave Serial** for serial programming. Leave the other options in default, and click OK and come back to main window.

Process Properties	×
iMPACT Programming Tool Properties	
Property Name	Value
Port to be used	None
Baud rate	None
Configuration Mode	Boundary Scan 💌
Configuration Filename	None
	Boundary Scan
	Slave Serial
	Select MAP
	Desktop Configuration
OK Cancel	Default Help

Step 14: Run the iMPACT programmer.



Step 15: Right click to add the device-programming file, browse to your project folder, and select the currently generated BIT file.



Step 16: Select the device file (the color would go green). Right click on the device file and select program option.

untitled [Configuration Mode] - iMPACT File Edit Mode Operations Operations Options Output	View Help				<u>_ </u>
	■ №				
Boundary Scan Slave Serial Select Man Deskton	Configuration				
TDI xc2s200 decoder.bit TDO					
'1': Loading file 'F:\1_USDP_Demo_Codes\PIC_Tests\decoder.bit'					<u> </u>
done.					
Device #1 selected Device #1 selected '1': Loading file 'F:\1_USDP_Demo_Codes\PIC_Tests\decoder.bit' done. 					
4					
For Help, press F1	Configuration Mode	Boundary-Scan	Parallel/PC3	lpt1	1

Step 17: Programming would start after clicking OK, if the programming is successful then the message would display on the screen.



Now check the functionality on the board and verify it by applying different inputs.

Design Flow for Quartus-II series of software of Altera.

Install Quartus-II (version 3.0 & above) software on your machine, the supported platforms are windows NT/XP/2000.

We take the same **decoder** example for implementing on the Altera ACEX1K FPGA.

Step 1: Start Quartus-II (version 3.0 & above) software.

Step 2: For new project creation, go to **File** option and select new project wizard. In the opened window, specify project location and design and entity name. For eg. Entity name **decoder**, and top design name also **decoder**. **Click "next"**.

c:\quartus\demo\decode	a./	
What is the name of this <u>p</u> top-level design entity.	roject? If you wish, you can use the name of the project's	
decoder		
What is the name of the to automatically create Comp n this wizard. After you cr Compiler and Simulator set	op-level design entity in your project? The Quartus II softw viler and Simulator settings for the top-level entity you speci reate a project, you can add more top-level entities and creatings for them with commands on the Assignments menu.	vare will ify eate
What is the name of the to automatically create Comp n this wizard. After you cr Compiler and Simulator set decoder	op-level design entity in your project? The Quartus II softw oiler and Simulator settings for the top-level entity you speci reate a project, you can add more top-level entities and cre ttings for them with commands on the Assignments menu.	vare will ify eate
What is the name of the to automatically create Comp n this wizard. After you cr Compiler and Simulator set decoder	op-level design entity in your project? The Quartus II softw viler and Simulator settings for the top-level entity you speci reate a project, you can add more top-level entities and cre ttings for them with commands on the Assignments menu.	vare will ify eate
What is the name of the to automatically create Comp n this wizard. After you cr Compiler and Simulator set decoder	op-level design entity in your project? The Quartus II softw viler and Simulator settings for the top-level entity you speci reate a project, you can add more top-level entities and cre ttings for them with commands on the Assignments menu.	vare will ify eate

Step 3: Click "next" button till you reach EDA tool settings window, there keep all options as none, which in default will select the inbuilt design tools and softwares for the design processing. **Click "next"**.

Tool type		Tool name	,		
Design entry/sy	nthesis	<none></none>			
Simulation		<none></none>			
i iming analysis. Board-level		<none></none>			
Formal verificati	on	<none></none>			
Resynthesis		<none></none>			
Tool settings-					
Tool type:	Design ent	try/synthesis			
Tool name:	<none></none>				•
🗖 Run this to	ol automatica	ally to synthesis	ze the current	design	Settings
					Advanced
					Advanced

Step 3: Select ACEX1K device family in the next window. Click "next".

w Project Wizard: Device	e Family [page	e 4 of 6]		x
Which device family do you	wish to target?			
Eamily: ACEX1K		J	-	
Do you want to assign a sp	ecific device?			
• <u>Y</u> es				
O No, I want to allow the I	Compiler to choo	ise a device		

New Project Wizard: Select a Target Device [page 5 of 6] х Use the Filters settings to control the devices that are displayed in the "Available devices" list. Select a device in the list, and click Next to continue. Available devices: Filters EP1K30TC144-3 * • Any EP1K50FC256-1 Package: EP1K50FC256-2 • EP1K50FI256-2 Any Pin count: EP1K50FC256-3 Speed grade: Any • EP1K50FC484-1 EP1K50FC484-2 Voltage: 2.5V EP1K50FC484-3 EP1K50QC208-1 EP1K50QC208-2 EP1K50QI208-2 EP1K50QC208-3 EP1K50TC144-1 EP1K50TC144-2 • CD1/COTI144 Next Back Finish Cancel

Step 4: In the next window, select the device as EP1k50QC2083. Click "next".

Step 5: Click "Finish". And the new project would be created. Now we need to make and add new design file in the project. So goto "File" menu, and click "New", and select VHDL File, in the "device design files" tab. Click "OK".

New	×
Device Design Files Software Files	Other Files
AHDL File Block Diagram/Schematic File EDIF File Verilog HDL File	
VHDL File	
	OK Cancel

Quartus II - c:\quartus\demo\decode	er\decoder - [decoder.vhd]	_ 8 ×
ab File Edit View Project Assignments	Processing Tools Window Help	_ 8 ×
📙 🗅 🚅 🖬 🎒 🐇 🖷 🛍 🖕 o		
🛛 💥 🏶 🔟 🔁 📾 🔜 🎨 🔮	e (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	
Courier New 💌 10 💌 🚧 🛟	準 岸 🔺 涔 尨 📲 🛛 💈 🔯 🔤	
	1 library IEEE;	
🔁 Files	2 use IEEE.STD_LOGIC_1164.ALL;	
🗄 💼 Device Design Files	3 use IEEE.STD_LOGIC_ARITH.ALL;	100
	4 use IEEE.STD_LOGIC_UNSIGNED.ALL;	
Other Files	5	
	6 entity decoder is	
	7 Port (Add : in integer range 0 to 7; address i/ps	
	8 en1 : in std_logic; active low enable	
	9 en2 : in std_logic; active low enable	
	10 en3 : in std_logic; active high enable	
	11 Y : out bit_vector(7 downto 0)); decoder o/p, active high	
📲 Hierarchies 🖹 Files 🗗 Design Units	12 end decoder;	
	13	
×	14 architecture Behavioral of decoder is	
Module Progress % Time 🔕	15	
	16 begin	
	17	
	18 Y <= "00000001" sll add when (en1='0' and en2='0' and en3='1') else	
	19 (others=>'0');	
	20	
	21 end Behavioral;	
Constitut (Cimulate (

Step 6: Write the VHDL code for 3:8 decoder, and save the file as "deoder.vhd"

Step 7: Now goto processing menu- then - start -, click start analysis and synthesis.



Step 8: Read the synthesis reports

 File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Tools Window Help File Edit View Project Assignments Processing Files Files Files Edit Prove Editors Files Software Files Files Analysis & Synthesis Summary Files Analysis & Synthesis Status Successful - Mon Oct 20 23:57:50 2003 Compiler Setting Name decoder Compiler Setting Name decoder 	💐 Quartus II - c:\quartus\demo\decoder	\decoder - [decoder Compilation Report]			_ 8
Image: Software Files Image: Software F	Edit View Project Assignments	Processing Tools Window Help			_ 8
Image: Second	🛯 🗗 🖬 🚭 🐇 🖻 💼 🗠 🗠	· 🕅 🕨 😓 🧶 🧶 🗞			
Analysis & Synthesis Status Successful - Mon Oct 20 23:57:50 2003	💓 🕸 💟 😰 📾 🔜 😌 🗐 🎨 😂	• • • • • • • • • • • • • • • • • • •			
Image: Section State Image: Section Secti	Software Files	Compilation Report Compilation Report Elegal Notice Flow Summary Flow Settings Analysis & Synthesis Summary Flow Elapsed Time Analysis & Synthesis Summary Flow Elapsed Time Analysis & Synthesis Summary Flow Elapsed Time Analysis & Synthesis Settings Flow Elapsed Time Analysis & Synthesis Resource Utilization by Er Flow Danalysis & Synthesis Resource Utilization by Er Flow Danalysis & Synthesis Resource Utilization by Er Flow Danalysis & Synthesis Resource Utilization by Er	Analysis & Synthesis Summary Analysis & Synthesis Status Compiler Setting Name Top-level Entity Name Family Total logic elements	Successful - Mon Oct 20 23:57:50 2003 decoder decoder ACEX1K 10	
Image: Synthesis Image: Synthesis <t< td=""><td>Module Progress % Time Processing Total 100 % 00:00 Analysis & Synthesis 100 % 00:00</td><td></td><td>Total pins Total memory bits Total PLLs</td><td>14 0 0</td><td></td></t<>	Module Progress % Time Processing Total 100 % 00:00 Analysis & Synthesis 100 % 00:00		Total pins Total memory bits Total PLLs	14 0 0	

Step 9: For performing simulation, we need to create stimuli file from where we can apply input signals and watch the o/p waveforms.

Goto file menu, and click new file, goto other files tab, and select "vector waveform file" option.

New	×
Device Design Files Software Files Other Files	1
AHDL Include File Block Symbol File Chain Description File Hexadecimal (Intel-Format) File Memory Initialization File SignalT ap II File Tcl Script File	
Text File Vector Waveform File	
OK Cancel	

Step 10: Add the entity signals in the waveform window, and apply different sets of value to check the functionality. Save the file as the same name of entity, **decoder.vwf**

Quartus II - c:\quartus\demo\decode	r\decoder	r - [de	coder.vwf*]							_ 8 ×
Eile Edit View Project Assignments	Processing	<u>I</u> 00	ls <u>W</u> indow <u>H</u> e	elp						_ 8 ×
D 🗳 🖬 🚭 X 🖻 💼 🗠 🤉	× \ ?	► ₿	: 🔭 🗇 🍕) 🧛 😽	🗞 🚸					
🛛 💢 🚸 🔟 🎦 📾 📰 🎨 🗍 🎨 🗳	• 🗣 🌚 •	Ŷ	2 3 🧳 6	6	ا 🕸 🕅					
Files	A	Mas	ter Time Bar:	15.0 ns	Pointer:	20.23 ns	Interval: 5.23 ns	s Start:	End:	
🗄 🗇 Device Design Files	₩.) (alua at	0 ps	10.0 ns	20.0 ns	30.0 ns	40.0 ns	50.0 ns
			Name	15.0 ns		15.0) ns J	·		
I I	44 12	Ď	🗉 Add	U 2		1)	2	× 3)	(4	X 5
I I			en1	BO						
I I	Xā 🔆		en2	BO						
I I	압사		en3	BO						
	<u>z</u>)	\bigcirc	ΞY	B XXX			XXX	****		
📲 Hierarchies 🖹 Files 🧬 Design Units	XI XH									
	XOC INV									
Module Progress % Time	Xc X0									
Processing Total 100 % 00:00	1/2									
Analysis & Synthesis 100 % 00:00										
	Pr 2↓									
Compile Simulate					•	1			I I	►

Step 11: In Altera Quartus-II software you can perform **Functional** and **Timing** simulation. For simulation mode settings, go to **assignments** menu, and click **settings**.

🐇 Quartus II - c:\quartus\demo\decoder\decoder - [decoder.vwf							_ 8 X
File Edit View Project Assignments Processing Tools Window	Help						_ 8 ×
🗋 🖻 📕 🎒 🕺 📴 🐓 Device	0 0 0	8 🗞 🚷					
₩ 43. Nm Pm		a an 🖂 📩					
📙 🔀 🐨 💟 🖾 🖾 🖬 🦉 Timing Settings							
EDA Tool Settings	15.0 ns	♦ Pointer:	267 ps	Interval: -14.73 r	ns Start:	End:	
Files Secongs Correspondence		0 ps	10.0 ns	20.0 ns	30.0 ns	40.0 ns	50.0 ns
Software Files Timing Wizard	Value at 15.0 ns		15.	0 ns			
Other Files Compiler Settings Wizard				Į			
Simulator Settings Wizard	02			<u> </u>	<u>, 3</u>)	4	<u>x 5</u>)
Software Build Settings Wizard	BU D0						
Assignment Editor Ctrl+Shift+A	BO	-					
Remove Assignments	BXX.						
Paulianuskia, Ph. Clas. 190		-					
Back-Annotate Assignments							
Import MAX+PLUS II Assignments							
Module Progres Progressing Tatal See & Last Compilation Floorplan							
Initialization							
🖻 Simulator Total 🛛 22 🐼 Chin Editor							
- Netlist Builder 52							
Simulator Simulator							
ist Import LogicLock Regions							
Compile A Simulate		1					•
					1		

Step 12: Now goto simulator setting, then to mode, and in the right-hand side window, select the simulation mode to Functional.

Settings	X
Settings Category: Files & Directories Add/Remove User Libraries Toolset Directories HDL Input Settings Verilog HDL Input Deal Tool Settings	Mode Select the simulation mode. Changes apply to Simulator settings 'decoder' Simulation mode: Timing Description: Functional Timing
Default Logic Option Settings Default Parameter Settings Timing Settings Other Requirements & Options Timing Analysis Reporting Ser Compiler Settings General Mode Options Time/Vectors Software Build Settings	In timing simulation, the Simulator uses a compiled netlist that includes estimated or actual timing information. This netlist can be a post-synthesis netlist that includes only estimated timing, or a fully compiled netlist that includes actual timing information. You can use Tcl commands and scripts to control simulation and to provide vector stimuli. You can also provide vector stimuli in a Vector Waveform File (.vwf), a text-based Vector File (.vec), or a Power Input File (.pwf). This type of simulation also allows you to check setup and hold times, detext gliches, and check simulation coverage (the ratio of nodes simulated to the total number of nodes in the design).
≟ Stratix GX Registration	▼ □K Cancel

Step 13: After clicking OK, come back to main window, and goto processing window, and click start simulation, the Quartus-II will start the simulation the result would appear in couple of minutes. Observe t he results, if found bugs, then change VHDL code and start simulation again.

🐇 Quartus II - c:\quartus\de	🖞 Quartus II - c:\quartus\demo\decoder\decoder - [decoder Simulation Report]											_	B×				
🕒 File Edit View Project Assignments Processing Tools Window Help											_	₽×					
👿 🖗 🔟 🔁 📾 📰 🌏	0	de 🖓 🕹 🙎 💈 🖉 🖉 🕼		🕘 📉 👱													
××		Simulator Report	Sim	ulation Wavef	orms												
Files	NA NA	🗃 El Legal Notice Ange Project Settings	Mas	ter Time Bar:	15.0 ns 🔄	Point	er: 3	31.18 n	s Inte	erval:	16.18	ns S	tart:		End:		
Software Files		🚑 🔁 Results for "decoder" Simulator Se				0 ps		20.0 n	s	40	.0 ns		60.0 m	ŝ	80.0 n	s 10)0.0 ns
🗄 💼 Other Files		Summary		Name	15.0 ns	15.0 ns											
	🚧 🖓	Simulation Waveforms		I∎ ∆dd	11.2	10Y	i V	- 2	ý á	-ý-	À Ý F	ÝŔ		; ;	ó ý	i v	3
	X0 .X	- 🚑 🐌 Messages		en1	B 0	P^	1		1 t	<u>-</u>	T-A	<u>^</u>		÷	Ť	† † †	1
	.0. <u>1</u>			en2	BO				1					1			
	ZVW			en3	B 1												
	Vī VH		1	ΞY	B 0000	(00)(00	0001)	000010	0001	OC <mark>X</mark> OO1	1000 0	0)(100	<u> 00x00</u>		00 <mark>×0000</mark>	0001,0	00000
			۲	-Y[7]	BO												
	Vr Va		0	Y[6]	BO												
•	\ <u>√</u> 2 \√2				BO									++			
	<u>^:</u>		0	-Y[4]	BO									++			
Module Progress	Br∺ 2+				B U B 1									++-			
Processing Total 1002			0	- Y[1]	BO						++			++			
- Initialization 100.2			0		BO	F			1-1		1						-
Simulator Total																	
Simulator																	
Compile A Simulate																	

Step 14: Once the simulation results are found correct, then we need to implement the design in the target device. For this we need to lock our design I/Os with the Kit I/O pin details. Goto **assignment** menu, click **"assign pin"** option.

🖞 Quartus II - c:\quartus\demo\decoder\decoder - [decoder Simulation Report]												X					
File Edit View Project Assignments Processing To	ools Window Help															- 8	×
🗋 🖆 🗐 🎒 🐰 📴 🐓 Device		1	۵ 📀														
Assign Pins		= 70	100 1 tot	1													
🗍 💢 🤎 🔟 🔁 🕼 📰 1 🐓 Timing Settings	S 2	9 📓	🗢 🛛 🖉														
EDA Tool Settings		Sin	nulation Wavef	orms													
Files Settings C	trl+Shift+E	Ma	ster Time Bar:	15.0 ns 🔹	Point	er:	1.68 ns	Interv	/al: -1	3.32 ns	Sta	rt:		Ene	d: 🗌		
Software Files Timing Wizard	mulator	5e 📃		Mahun at	t 0 ps 20,0 ns 40,0 ns 60,0 ns 80,0 ns 15,0 ns								Ins	100.0 ns	.0 ns		
Other Files <u>Compiler Settings Wizard.</u>			Name	15.0 ns											_		
Simulator Settings Wizard	ns		∃ Add	U 2	σx	τĪ	ż	<u>x</u> ż	X Å	<u> </u>	6	χŻ	X	όx	1	X	ź
Sortware build Settings w	12di u		en1	BO													
Assignment Editor C	trl+Shift+A		en2	BO													
Remove Assignments			en3	B 1										1		L	
E Demote Assignments		6	ΞY	B 0000	00,00	0001	000010	000100	00100	<u>X 00 X 00</u>	10000	0,000	00,00	0000X	00000	1,000	000
Back-Annotate Assignmen	nts		- Y[7]	BO													
Import MAX+PLUS II Assi	gnments	•	- Y[6]	BO													
📕 🗗 🕞 🕞	1	•	- Y[5]	BO													
Timing Closure Eloorplan				BO													
			-Y[3]	BO					1								
Module Progres Chip Editor			-Y[2]	B 1				<u> </u>				_					
Processing Lotal LogicLock Regions Window	v Alt+L		-Y[1]	BO					<u> </u>							7	
E- Simulator Total			L-Y[0]	BO								_					
Netlist Builder 100 Sta Export LogicLock Regions																	
Simulator 100 %																	
Compile Simulate																	

Step 15: Looking at the pin assignment chapter, lock the ACEX 1K FPGA I/O with the particular pin no., for this select the I/O number on the LHS, name the design I/O in the bottom pin name option, and then click add, the particular signal will be locked to that pin number

Assign Pins								x
Select a device pin and the type of Floorplan Editor. You can reserve u You must perform a smart compilatio	assignment y inused pins o on on the des	you wish to make on a device-wide sign before routin	: You can basis with t g SignalProl	also make pin assignm he Unused Pins tab in be signals.	nents in the A n the Device	Assignment I & Pin Option	Editor and the ns dialog box.	
Changes apply to Compiler settings Available Pins & Existing Assignmen	'decoder'							
Nu Name:	1/0 Bank:	1/0 Standard	Tune:	SignalProbe Sour	ce Name	Enabled	Status	
7 en1	0	LVTTL/LVC	Row 1/0,.			Off		
8	0	LVTTL/LVC	Row I/O			Off		
9	0	LVTTL/LVC	Row I/O			Off		
11	0	LVTTL/LVC	Row I/O			Off		
12	0	LVTTL/LVC	Row I/O			Off		
Show 'no connect' pins			Г	Show current and po	otential Signa	alProbe pins		
Assignment								
Pin <u>n</u> ame: en2			Si	gnalProbe source:				
			г	SignalProbe enable				
[/O standard:n/a in ACEX1K				, Giginaji tobo onabio				
E Basarua pin fauan if it daas n	at avriat in the	design file):	Ľ	DCK:				
As insult bi stated	ocexist in the	e design niej.						
As input tri-stated				egiacera:				
	Add	Dele	ie	Enable All SignalPro	be Routing	Disa	ele All SignalPi	robe Routing
							ОК	Cancel

Step 16: Once the pin assignment is over, come back to main window. Now we need to implement the design on the particular device. So goto **processing** menu, and click **start compilation** process. Which will place & route the design in FPGA and generate the programming file.

🐇 Quartus II - c:\quartus\der	mo\decoder\decoder - [decoder.vhd]	_ 8 ×
• File Edit View Project A	issignments Processing Tools Window Help	_ 8 ×
] D 🛎 🖬 🚭 X 🖻 I	🖹 📭 c 🕮 Stop Processing Ctrl+Shift+C 🦉 🗞 🗞	
Courier New 10 10 4	Image: Start Compilation Ctrl+L Image: Start Compilation Report Ctrl+L Image: Start Compilation Report Ctrl+R Image: Start Compilation & Simulation L; Image: Start Simulation & Simulation L; Image: Start Simulation & Simulation L; Image: Start Simulation & Simulation Ctrl+R Image: Start Simulation & Ctrl+Simulation Ctrl+R Image: Start Simulation Debug Image: Start Simulation Ctrl+Simulation Image: Start Simulation Report Ctrl+Simulation Image: Start Software Build Ctrl+Q Image: Start Software Build Ctrl+Q	*
	<pre>10 logic; active high enable 11 Y : out bit_vector(7 downto 0)); decoder o/p, active high 12 end decoder; 13 14 architecture Behavioral of decoder is</pre>	
••••	15 16 begin	
Module Progressing Total Processing Total 100 2 Initialization 100 2 Simulator Total 100 2 Nettist Builder 100 2 Simulator 100 2	<pre>17 18 Y <= "00000001" sll add when (en1='0' and en2='0' and en3='1') else 19 (others=>'0'); 20 21 end Behavioral;</pre>	
Compile Simulate	× ·	•

Step 17: Once the compilation process is over, user can check the reports and see the floorplan window.

Quartus II - c:\quartus\demo\decoder\decoder - [decoder Compilation Report] File Edit View Project Assignments Processing Tools Window Help		_ B ×
Image: Software Files Book of the files	Flow Summary Flow Status Flow Status Successful - Tue Oct 21 00:11:23 2003 Compiler Setting Name decoder Top-level Entity Name decoder Family ACEX1K Device EP1K50TC144-3 Total logic elements 10 / 2,880 (< 1 %) Total pins 14 / 102 (13 %) Total memory bits 0 / 40,960 (0 %) Total PLLs 0 / 1 (0 %)	

Step 18: Now we need to program CPLD, for this goto tools menu, and click programmer. Which will open the programmer; the software will automatically add the programming file (decoder.pof). In the opened window select the **program/configure** option. Now we need to select the programming hardware, for which click the hardware tab on the top LHS of programming window.

Quartus II - c:\guartus\demo\dec Image: State of the state of t	oder\de	ecoder - [decoder.cdf*] cessing Tools Window He	elp								_ 8 × _ 8 ×
		₩? ► 🔭 📩 🎟 🖉									
j 💥 🍄 🔟 🔁 📾 🗮 🕹 🗍 🏵		> © • • • • • • • • • •) 5 😫 📎 ×	<u></u>					-		
Files	M	Hardware ByteBlaster		fode: JTAG			▼ Pr	ogress:		0%	
Device Design Files Software Files	الله الله	File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	
⊡ Other Files		1o\decoder\decoder.sof	EP1K50T144	00011CD0	0000007F						
		Hardware Setup					×				
	1	Hardware Settings JT/	AG Settings				1				
		Select a programming h hardware setup applies	hardware setup to use w s only to the current prog	nen programmin ammer window.) devices. This pr	ogramming					
		Currently selected hard	lware: No Hardware								
	1	Hardware	Server	Port	Selec	ct Hardware					
	-				Add	Hardware					
Module Progress ?	4				Remo	ve Hardware					
Processing Total 100 %											
- Analysis & Synthesis 100 %											
Fitter 100 %											
Assembler 100 %											
Tilling Analyzei						Llose					
٠ •											
Compile Simulate											

Step 19: In the opened window Click add hardware tab, and select the hardware type as ByteBlasterMV or ByteBlaster II and port as LPT1.

💔 Quartus II - c:\quartus\demo\dec 🌇 File Edit View Project Assignme	oder\a	lecoder - [decode ocessing Tools W	er.cdf*] 'indow Help									_ & ×
		N? 🕨 🔭 📩	• • •	🨻 🗞 📎								
💢 🏶 🔟 🔁 📾 🗮 🎨 🗍 🎨	€ ♦	🔊 🔮 🤨 🖉	3 🗳 🔒 🖻	5 😰 📎 🖄	(<u>1</u>							
Files	₩	Hardware	ByteBlaster (LPT1)		Mode: JTAG			▼ Pi	ogress:		0%	
Device Design Files Software Files	- Market (1997)	File	Devi	ice	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	
± Other Files	×	1o\decoder\de	ecoder.sof EP1K	50T144	00011CD0	0000007F						
		Hardware Set	up					×				
	1	Hardware Sel	ttings JTAG Sett	ings								
		Select a pro hardware se	gramming hardwai etup applies only to	re setup to use () the current pro	vhen programming grammer window.	devices. This pr	ogramming					
		Currently s	Add Hardware				×					
		- Available Hardw	Hardware type:	ByteBlaster	MV or ByteBlaster	II <u>-</u>	ardware					
•••••			Port	LPT1		•	dware					
Module Progress ?	1		Baud rate:			<u>_</u>	Hardware]				
Processing Total 100%				OK	Cance							
Full Compilation		-										
Fitter 100 %												
Timing Analyzer							Close					
Compile / Simulate /												

-										(=1)
Quartus II - c:\quartus\demo\deco	der\decoder - [decoder.cdl*]	Halo								
	Is Processing roots window i									-0-
	ା 🖓 🕨 🚼 🗱 🔍 🖓	♥ 옆 �� � �								
🛛 💥 🕹 🔟 🔁 📾 🗮 🎨 🗍 🕸	n 🖓 🖓 🕹 🖉 🖉	8 🗟 🖉 👌	k 😐 📃							
	Hardware ByteBlaste	er (LPT1)	Mode: JTAG	i		T Pr	ogress:		0%	
Original Device Design Files Software Files	File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	
± Other Files	P 1o\decoder\decoder.se	of EP1K50T144	00011CD0	0000007F						
Image: Second state of the se	Hardware Setup Hardware Setungs Hardware Settings Select a programming hardware setup applies Currently selected ha Available hardware ByteBlaster	TAG Settings g hardware setup to use les only to the current pro ardware: No Hardwa e items: Server Local	when programmin grammer window. re Port LPT1	g devices. This p	rogramming ct Hardware Hardware we Hardware Close	X				

Step 20: Come back to hardware setup window, and click the select hardware tab and close the window.

Step 21: Now click the **start-programming** button (play symbol) on the top LHS of the programming window (keep the program/configure option selected).

🏀 Quartus II - c:\quartus\demo\deco	oder\d	ecoder - [decoder.cdf*]								
	(a)	k? > 🔭 🔭 👜 🍥	🕘 🦁 🗞 🔗							
🕺 🕹 💟 😨 🔊 🗮 🕹 🛛 🕸	و چ	• • • • • • • • • • • • • • • • • • •	5 🛯 🖉 📉							
		Hardware ByteBlasterM	V [LPT1] M	ode: JTAG		•	Progress:		0%	
Compilation Hierarchies	St	art Programming File	Device	Checksum	Usercode	Program/ Configure	Blank-	Examine	Security Bit	
	*	1o\decoder\decoder.sof	EP1K50T144	00011CD0	0000007F					
	×									
	2 2									
	2									
T	8									
∎₽	1									
	-									
Module Progress % Time Q	4									
Compile / Simulate /										

Check the design functionality on the board, by applying signal from switches or other points.

Using Keil Compiler

Designers can use compiler from **Keil**, one of the compilers available in market. **Keil** is a cross compiler supporting the 8051 based architecture controllers from various vendors. Compiler support the source codes written in 'C' and assembly languages.

For starting up with USDP, we have made a design flow guide for using the **keil** compiler, but for more information users can surf the help index of keil compiler,

Install the **Keil** compiler from provided CD-ROM; you can use the evaluation version at start up, which supports the program code upto 2KB, which is sufficient for small development purposes.

 Image: Construction of the set of the

Step 1: Run the keil compiler EXE, which in turn will open the compiler.

Step 2: Go to project menu and left click the new project option.

🏨 µ¥ision2	
Eile Edit View Project Debug Peripherals Tools SVCS Window !	Help
Image: Second state of the second s	
Iargets, Groups, Files Select Device for Target Remove Item Options Alt+F7 Clear Group and File Options	
Build target F7 Bebuild all target files Translate Ctrl+F7 Stop byId	
1 C:\Keil\shail\SWITCH.uv2 2 C:\Keil\serial.Uv2	

Step 3: In the opened window user has to give project name and select the folder for the project creation, for example, name it **keil demo**. Click OK, and in the next window, you have to select the device to work on, the vendor name is Philips, and the device number is 8xC51RD2. The window will look like as below.

∰keil demo –µ¥ision2			_ 8 ×
]] Eile Edit View Project Debug Periphera	als <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp		
🎦 🗲 🖬 🎒 👗 🖿 😭 🗩 😂	: 幸 幸 ゐ % % % 🙀 📃		
🖉 🕮 🎬 👗 🕺 Target 1	T		
	Select Device for Target 'Target 1'	<u>?</u> ×	
I Target 1	CPU		
	Vendor: Philips		
	Device: 8xC51RD2	Use Extended Linker (LX51) instead of BL51	
	Family: MCS-51	Use Extended Assembler (AX51) instead of A51	
	Data base	Description:	
		8051 based CMOS controller with PCA, Dual DPTR, WDT, 32 I/J0 lines, 3 Timers/Counters, 7 Interrupts/4 priority levels 64 K ISP FLASH EPROM, 256 Bytes on-chip RAM, additional 768 Bytes X	
		OK Cancel	
Files B			

Step 4: Click ok on the above window, and the project would be created. Now the user has to add or create the source code for the controller. For example for now designer can use the source code provided along with the USDP code examples. So right click on **source group** and click **add files** to group.

🦉 keil demo - µYision2	
Eile Edit View Project Debug Peripherals Tools SVCS Window Help	
12 🚅 🖬 🕼 🔍 🕮 卓 🤌 洗 🦄 🦓 🦓 🔽 🗹 🏘 💹 🖉 🖉 🖉 💆 🖉 🕅 🖉	
● ②	
Image: Source Group 1 Select Device for Target !Target 1' Options for Group 'Source Group 1' Open File Image: Select Device for Target !Target 1' Options for Group 'Source Group 1' Image: Select Device for Target !Target 1' Options for Group 'Source Group 1' Image: Stop by: Add Files to Group 'Source Group 1' Iargets, Group, Files Remove Group 'Source Group 1' and it's Files	

i∰keil demo - μ¥ision2		_ 8 ×
Eile Edit View Project Debug Peripherals Tools SVCS Window Help		
🎦 😅 🖬 🕼 ふ 🖻 🛍 ユ ユ 卓 卓 み % % % 🖗 🥅	🗹 🛤 🐚 🗁 🔍 🖪 🗖 🖑 🕷 🕅	
🔗 🕮 🚈 🌋 Target 1		
Add Files to Group 1'Source Group 1' Look in: Seria_transmit.c File name: seria_transmit.c Files of type: C Source file (*.c) Files of type: C Source file (*.c)	? × ▼ ← E → E → E	

Step 5: Browse for the demo source code serial_transmit.c and click add to insert it in the project.

Step 6: User can have a look on the demo source code, which takes data from Port 0 and sends it serially to PC. The source code is in 'C' language; users can also use assembly code for designing.

🌉 keil demo - µ¥ision2 - [E:\ni Logic Pvt	. Ltd\ni2 Designs\Almighty Kit\Keil Demo\serial_transmit.c]	_ 8 ×
Eile Edit View Project Debug Periph	nerals <u>T</u> ools <u>S</u> VCS <u>W</u> indow <u>H</u> elp	_ <u>8 ×</u>
) 🎦 🗃 🖬 🌒 X 🖣 🛍 그 오	· 存存人%%%%%	
🛛 🅸 🏝 🎽 👗 🖍 Target 1		
Target 1	<pre>/* This Program takes data from PORT 0 and sends it on the serial PORT * The data is sent only when there is change on the port 0 data. * This program runs with FFGA only. * This code is written for USDP */ #include <intrins.h> #include <ireq52.h> #define TIMER_RELOAD (-921) typedef unsigned int UINT; typedef unsigned int UINT; typedef unsigned long ULONG; UCHAR mult;</ireq52.h></intrins.h></pre>	
Files Regs W Books	<pre>void serial (void) interrupt 4 using 2 { if(RI) RI = 0; } void init_hw(void) { TMOD = 0x21; IE = 0x92; //1001 0010 SCON = 0x50; //0101 0000 TL0 = TIMER_RELOAD; TH0 - TIMER_DELOAD; X </pre>	×

Step 7: For building the program code or HEX code, we need to set some parameters in the compiler. Go to project options, and click **option for target**.

🏨 keil demo 🛛 - µ¥ision2 - [E:\ni Logic Pvt. Ltd\ni2 Designs\Almighty Kit\Keil Demo\serial_transm	it.c]
Eile Edit View Project Debug Peripherals Tools SVCS Window Help	
Import µVision1 Project Import µVision1 Project Open Project	19 5 9 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Close Project Ele Extensions, Books and Environment Ele Extensions, Books and Environment	ds it on the serial PORT on the port 0 data.
Source (Iargets, Groups, Files Select Device for Target Target 1' Remove File Senial_transmit.c' Options for Target 1' Clear group and File Options	_
E Build target	F7
L Et(ni Logic Pvt. Ltd(ni2 Designs\Almighty Kit\Keil Demo\keil demo.uv2 C:\Keil\shall\SWITCH.uv2 G:\Keil\serial.Uv2	
<pre></pre>	

Step 8: In the opened window, go to target tab, and in the window, set the Xtal frequency as 11.0592 MHz.

Options for Target 'Target 1'	<u>? ×</u>
Target Output Listing C51 A51 BL51 Locate BL	51 Misc Debug
Philips 8xC51RD2	
<u>X</u> tal (MHz): 11.0592 Memory Model: Small: variables in DATA ✓ Code Rom Size: Large: 64K program ✓ Operating system: None	 Use On-chip ROM (0x0-0xFFFF) Use multiple DPTR registers Use On-chip XRAM (0x0-0x2FF)
Off-chip Code memory Eprom Eprom Eprom Eprom	Off-chip Xdata memory Start: Size: Ram Ram Ram Ram
Code Banking Start: End: Banks: 2 Bank Area: 0x0000 0x0000	 'far' memory type support Save address extension SFR in interrupts
OK Car	Defaults

Step 9: Now goto output tab and set the name of executable file as serial_transmit; set the option create executable; and set create HEX file option. Finally click OK on the bottom side and come out of the window.

Options for Target 'Target 1'	? ×
Target Output Listing C51 A51 BL51 Locate BL51 Misc Debug	
Select Folder for Objects Name of Executable: serial_transmit	
Create Executable: ./serial_transmit	
Debug Information	
Create HEX File HEX Format: HEX-80	
C Create Library: .\serial_transmit.LIB	
C After Make	
Beep When Complete Start Debugging	
Run User Program #1: Browse	.
Run User Program #2: Browse	.
OK Defaults	

Step 10: Now to build the HEX file, goto project menu, and click the build target option.

👿 keil demo 🕘 µ¥isio	n2 - [E:\ni Logic Pvt. Ltd\ni2 Designs\Almighty Kit\Keil Demo\serial_transmit.	c]	_ 8 ×
Eile Edit View	Project Debug Peripherals Tools SVCS Window Help		_ 8 ×
	<u>N</u> ew Project Import µVision1 Project Open Project	12 3 3 4 5 3 5	
	Glose Project Ejle Extensions, Books and Environment	ds it on the serial PORT on the port O data.	
i - 🚑 Source (Iargets, Groups, Files Select Device for Target 1' Remove file Serial_transmit.c' Options for Target 1'arget 1' Clear Group and File Options		
	Euld target F7 ∰ Bebuild all target files ∯ Translate E:\ni Logic Pvt. Ltd\ni2 Designs\Almighty Kit\Kell Demo\serial_transmit.c ∭ Stop bylld		
	1 E:\ni Logic Pvt. Ltd\ni2 Designs\Almighty Kit\Keil Demo\keil demo.uv2 2 C:\Keil\shail\SWITCH.uv2 3 C:\Keil\serial.Uv2		
	<pre>{ if(RI) RI = 0; } void init_hw(void) { TMOD = 0x21; IE = 0x92; //1001 0010 SCON = 0x50; //0101 0000 TL0 = TMPE PEIOD: TL0 = TMPE PEIOD: SCON = 0x50; //0101 0000 TL0 = TMPE PEIOD: SCON = 0x50; //0101 0000 TL0 = TMPE PEIOD: SCON = 0x50; //0101 0000 TL0 = TMPE PEIOD: SCON = 0x50; //0101 0000 SCON</pre>		
Files Regs	Books THO - TIMER DELOND N 8.		▼ ▶

This will generate the HEX file in the project folder, which you can use to program the controller using the provided Flash Magic programming software.

Using PIC C Lite Compiler

Designers can use **PIC C Lite** compiler from **HI-TECH Software**, one of the compilers available in market.

PIC C Lite is a DOS based cross compiler supporting Microchip PIC microcontrollers.

Compiler support the source codes written in 'C' and assembly languages, and works on Win 98/2000 platforms.

The choice of compiler is solely dependent on the user; they can also use the compilers available from other vendors, like MPLAB or others.

For starting up with USDP, we have made a design flow guide for using the **PIC C Lite** compiler, but for more information users can surf the help index of the compiler,

Install the **PIC C Lite** compiler from provided CD-ROM; you can use the evaluation version at start up, which supports the program code upto 2KB, which is sufficient for small development purposes.

Step 1: Run the PIC C Lite compiler EXE, which in turn will open the compiler.





Step 2: To create new project goto Make menu, and select New project option.

Step 3: After giving the project name and location, you need to select the device. The provided PIC module consists of **16F877** processor.

MTLPI	C							8 ×
(O) I	ile Edit	Op <mark>t</mark> ions C	omp <mark>i</mark> le	Make Rur	Utility	Help		
unti	tled —							
	Select Mid > 12F629 > 12F675 > 16C84	range proces () 16F627 () 16F84 () 16F84A	sor = (*) 16F () 16F	877 877A				
[K	OK – En Help – F1	ter]	K	Cancel -	Esc >		<pre></pre>	
Line	1/1	Col 1	Insert	C mode	K Search F	2 >		

Step 4: Select the float type according to your design requirement, else select 24 bit double float file format.



Step 5: In the next window select the output file format as Intel HEX file.



Step 6: Select the compiler optimization according to your requirement; by default you can select Global Optimization.



Step 7: Now user has to select mapping and symbol file option. User can select the options according to his requirement, or select for source level debug info.



Step 8: Now add the source files in the project. Right now, add the provided **disp_pic.c** source file provided along with the USDP. Browse to the folder by pressing FILE option and add the said source file in the list.

C:N	HTLPIC									_ ₽ ×
	«» File	Edit	Op t ions	Compile	Make	Run	Utility	Help		
Γ										Ċ
										1
Ш	DISP_PIC.	C								
						• • • • • • • • • • •				••••••••
										111
Ш										
Ш										- 10 C
Ш										
Ш										
Ш										
										111
										111
Ш										
Ш										
Ш										
Ш										111
Ш										
Ш										
Ш										
Ш										111
Ш										
L	< DONE	ESC		ELETE	F9 >	< IN	SERT F1	0 >	< ABS/REL	<u>F2</u>
	K FILE New entr	F8 9		ANK Ctrl- athnames:	<mark>F9 ></mark> relat	<mark>< PU</mark> ive	T Ctrl-F1	2 >		ţ

Step 9: Now the project has been created. User can edit the source file in the editor or check the code.

HTLPIC	×
Some set to the set of the set	
DISP_PIC.C /* USDP Sample Code Display "Hi Iam USDP" on 7-segment displays. Requires FPGA as feed through in between display and PIC. */	\rightarrow
#include <pic1687x.h></pic1687x.h>	
typedef unsigned int UINT; typedef unsigned char UCHAR; typedef unsigned long ULONG;	
#define TIMER1_RELOADH (ØxfC) #define TIMER1_RELOADL (Øx66)	
<pre>static volatile bit SELØ @ (unsigned)&PORTC×8+0; static volatile bit SEL1 @ (unsigned)&PORTC×8+1; static volatile bit SEL2 @ (unsigned)&PORTC×8+2; static volatile bit SEL3 @ (unsigned)&PORTC×8+5; static volatile bit TEST @ (unsigned)&PORTC×8+4;</pre>	
UINT count; bit secflag; UCHAR digit; UINT digbuf[4]; UCHAR hi[4], iam[4], usdp[4];	
void inithw(void) <	
ADCON1 = 0×06; INTCON = 0×00; OPTION = 0×00;	
$ \begin{array}{rcl} \text{TRISC} &= & 0 \times 00 \ ; \\ \text{TRISD} &= & 0 \times 00 \ ; \end{array} $	
TMR1H = TIMER1_RELOADH; TMR1L = TIMER1_RELOADL; Line 2/132 Col 1 Insert C mode < Search F2 >	† IIIIIII

Step 10: The next step is to compile the source code. For compilation and linking the source code, go to **Compile** menu and select **Compile and Link option**.

MTLPIC		_ B ×
↔ File Edit Options	Compile Make Run Utility He	1p
/* USDP Sample Code Display "Hi Iam USDP" on Requires FPGA as feed thr */	Compile and linkF3Compile to OBJshift-F3Compile to ASctrl-F3Preprocess only to .PRE	
<pre>#include <pic1687x.h> typedef unsigned int UINT typedef unsigned char UCH typedef unsigned lang ULO</pic1687x.h></pre>	Stop on warnings Warning level alt-W Optimization alt-Z Identifier length Disable non-ANSI features	
#define TIMER1_RELOADH (#define TIMER1_RELOADH (Pre-process assembler files Generate assembler listings Generate C source listings	
static volatile bit SELØ static volatile bit SEL1 static volatile bit SEL2 static volatile bit SEL3 static volatile bit TEST	<pre>@ (unsigned)&PORIC=*8+0; @ (unsigned)&PORIC=*8+1; @ (unsigned)&PORIC=*8+2; @ (unsigned)&PORIC=*8+5; @ (unsigned)&PORIC=*8+4;</pre>	
UINT count; bit secflag; UCHAR digit; UINT digbuf[4]; UCHAR hi[4], iam[4], usdp yoid inithw(yoid)	[4];	
ADCON1 = 0×06; INTCON = 0×00; OPTION = 0×00;		
$\begin{array}{rcl} TRISC &= & 0 \times 00; \\ TRISD &= & 0 \times 00; \end{array}$		
TMR1H = TIMER1_RE TMR1L = TIMER1_RE Line 2/132 Col 1	LOADH; LOADL; Insert C mode < Search F2 >	•

Step 11: If the compilation process is over then the HEX will be generated in the project location, on the name of source code.. If user wants to run the **Make** process then they can go to **Make** menu and select Make option. This will create the HEX file on the project name.

TLPIC	
Some state with the second	Make Run Utility Help
/* USDP Sample Code Display "Hi Iam USDP" on 7-segment	MakeF5Re-makectrl-F5Re-linkshift-F5
<pre>#equires FFGH as feed through in b #/ #include <pic1687x.h></pic1687x.h></pre>	Load project alt-P New project Save project Rename project
typedef unsigned int UINI; typedef unsigned char UCHAR; typedef unsigned long ULONG;	Output file name Map file name Symbol file name
#define TIMER1_RELOADL (0x66)	Source file list Object file list
static volatile bit SEL0 @ (unsi static volatile bit SEL1 @ (unsi static volatile bit SEL2 @ (unsi static volatile bit SEL3 @ (unsi static volatile bit TEST @ (unsi	Library file list CPP pre-defined symbols CPP include paths Linker options Objtohex options
UINT count; bit secflag; UCHAR digit; UINT digbuf[4]; UCHAR hi[4], iam[4], usdp[4]; yoid inithw(void)	
ADCON1 = 0×06; INTCON = 0×00; OPTION = 0×00;	
TRISC = 0x00; TRISD = 0x00;	
TMR1H = TIMER1_RELOADH; TMR1L = TIMER1_RELOADL; Line 2/132 Col 1 Insert	C mode < Search F2 >

Users can now use this HEX code to program the PIC controller using provided **PROGPIC** programming software.

Chapter 9 Configuration

This chapter covers the configuration process required for the programming/configuration of FPGAs and microcontrollers.

USDP supports multiple FPGA stacking with the PLD connectors (Slot1 and Slot2). This gives the flexibility for implementing multiple FPGA based designs, or users can also use FPGA independently. Users can use the following combination of FPGAS.

Stacking Options				
	Slot 1	Slot 2		
Choice 1	Xilinx	-		
Choice 2	Altera	-		
Choice 3	Xilinx	Xilinx		
Choice 4	Altera	Altera		
Choice 5	Xilinx	Altera		

Note: Both the PLD slots share the bus, users has to take care about the pin mode while using the both slots together. No two pins should be output; else there may be chances of hardware short.

Till now the user is familiar with the Xilinx and Altera EDA tools for development of HDL based designs. Now further more we will cover the necessary steps required to configure the devices.

Configuring Xilinx FPGAs

- ? Select the configuration mode and generate the programming file accordingly.
- ? Set the mode selection switch position on Xilinx module with reference to your selected programming mode.
- ? Check the PROM jumper setting for using or bypassing the PROM.
- ? Set the jumper selection for JTAG/serial mode.
- ? Inset the Xilinx module on any one of the PLD slots (Slot1 or Slot2).
- ? Connect the programming cable.
- ? Short the 1-2 connections of PLD sel header on the baseboard, to select the Xilinx programmer.
- ? Select the slot you want to program.
- ? Turn on the board supply.
- ? Run the programmer on the Xilinx ISE series software.

Note: For jumper/header setting refer chapter Jumper Setting, and for Xilinx ISE flow refer chapter EDA tools.

Configuring Altera FPGAs

- ? Select the configuration mode and generate the programming file accordingly.
- ? Set the mode selection jumpers on Altera module with reference to your selected programming mode.
- ? Check the PROM jumper setting for using or bypassing the PROM.
- ? Set the jumper selection for JTAG/serial mode.
- ? Inset the Altera module on any one of the PLD slots (Slot1 or Slot2).
- ? Connect the programming cable.
- ? Short the 2-3 connections of PLD sel header on the baseboard, to select the Altera programmer.
- ? Select the slot you want to program.
- ? Turn on the board supply.
- ? Run the programmer on the Altera Quartus series software.

Note: For jumper/header setting refer chapter jumper setting, and for Altera Quartus flow refer chapter EDA tools.

Slot selection

There are total five slots on the baseboard of USDP. The first two (Slot1 & Slot2) are PLD connector slots where user can insert provided FPGA modules. The remaining PCI based connector slots (Slot3, Slot4 & Slot5) are used for add-on module insertion.

FPGA module can be inserted in Slot1 & Slot2, but for programming the FPGAs, user has to select the Slot which he has used and the device vendor he intends to program; this can be done by using the selector cards provided along with the USDP board. The PLD selector card can be used to select the PLD vendor and the Slot selector card can be used to select the slot. Refer the table below for the combination.

PLD Selector				
Altera Xilinx				
1-2	2-3			

Slot Selector				
Slot 1 Slot 2				
1-2	2-3			

Programming Modes

Both the PLD vendors Xilinx and Altera modules support various programming modes to download the configuration file in FPGA.

Here is the table for programming modes for Xilinx and Altera modules.

For Xilinx FPGA module

Modes	MO	M1	M2
JTAG	1	0	1
Slave Serial	1	1	1
Master Serial	0	0	0

The same table is also printed on the module, which the users can refer while programming. To apply logic high and low on the mode pins, users have to use the DIP switch provided. By turning ON, the switch will apply logic HIGH on the mode pin and by turning OFF, the switch position will put logic LOW on the mode pins.

Note: The fourth switch is Not Connected (NC) and is not in use.

For Altera FPGA Module

	MSEL1	MSEL0
Passive Serial (PS)	0	0
JTAG	0	1

Users can use the selection jumpers (J3 & J8) provided on module to select the configuration mode. For logic LOW short 2-3 and for logic short 1-2 for logic HIGH.

Jumper setting for mode selection

Apart from mode selection pin settings, user also has to select the programming pins for programming.

For Xilinx FPGA module

Jumpers	Serial	JTAG
JP4, JP5, JP6, JP7	Short 1-2	Short 2-3

For Altera FPGA Module

Jumpers	Serial	JTAG	
J4, J5, J6, J7	Short 2-3	Short 1-2	

Programming 89c51RD2 microcontroller

Using the Flash Magic programmer, designers can program the 89c51 controller. After building the HEX file from compiler, designers can refer the below shown steps to program the controller.

Insert the controller module on baseboard; connect the serial cable provided to module and PC's serial port. Turn ON the power supply; now run the EXE of Flash Magic programmer on your PC. The below shown window will open up.

📕 Flash Magic		
File ISP Options Help		
1 COM Port: COM 1 Baud Rate: 9600 Device: 89C51RA2xx Oscillator Freq. (MHz):	2 Erase block 0 (0x0000-0x0FFF) Erase block 1 (0x1000-0x1FFF)	
3 Hex File: Last Modified: Unknown	Browse Size: Unknown	
4 5 Verify after programming Set Security Bit 1 Fill unused Flash Set Security Bit 2 Generate checksums Set Security Bit 3 Execute 6 clks/cycle		
Your Training or Consulting Partner: Embedded 9 <u>www.esacademy.com</u>	Systems Academy 0	

The above software is been divided in 5 sections.

- 1 For device selection and settings.
- 2 For program memory block erasure selection.
- 3 Selection of HEX file.
- 4 Programming options, and security settings.
- 5 Starting programming.

So, one bye one we will look on all the sections and settings required to work on this particular controller module.

Flash Magic			
File ISP Options Help			
_1 Reset2			
Advanced Options			
Erase block U (UXUUUU-UXTFFF)			
Baud Rate: 9600 Erase block 2 (0x4000-0x7FF)			
Device: 89C51RD2Hxx Erase block 3 (0x8000-0xBFFF) Erase block 4 (0xC000-0xFFFF)			
Oscillator Freq. (MHz): 11.0592 Erase all Flash+Security			
3			
Jun Film La Later? Design Mahrisha Kitter? Descala sid here a here			
Hex File: Vt. Ltd\ni2 Designs\Almighty Kit\Kell Demo\serial_transmit.hex			
Last Modified: 11/7/2003 4:14:11 PM Size: 302 bytes			
-4			
Verifu after programming D Set Securitu Bit 1			
Fill unused Flash			
🗖 Generate checksums 🔲 Set Security Bit 3			
Execute 6 clks/cycle Start			
On-Line training classes for microcontrollers and embedded networking and Internetworking			

Step 1: Before the settings of all 5 sections, go to option menu and click advance options.

Step 2: In the opened window, go to communications tab, and set the option High Speed Communication and 6-clock part.

1–	2	
A	dvanced Options	×
0s 3 - He	Communications Hardware Config Security Just In Time Cod High Speed Communications 6 clock part 12 clock part Half-duplex Communications	de Misc
4	Cancel	ок
	erify after programming 🔲 Set Security Bit 1	

Step 3: Now go to hardware config tab, and set the option Use DTR and RTS to control RST and PSEN. Also set the T1 value by 100 ms and T2 by 200 ms. now click OK and go back to main window.

File IS	h Magic P Options Help			_ 🗆 🗙
-1-		2		
A	Advanced Options			
	Communications	Hardware Config Security	Just In Time Code	Misc
	✓ Use DTR and RTS to control RST and PSEN			
Os	Os 🔽 Keep RTS asserted while COM Port open			
2	T1: 100 ms T2: 200 ms			
3		IDTO 11 COMP .		
He Assert DTR and RTS while COM Port open				
4			Cancel	OK
	erify after programm	ing 🥅 Set Security Bit 1		
Fi	ll unused Flash enerate checksum:	Set Security Bit 2		
E:	enerate checksums (ecute	6 clks/cycle		Start
On-Line training classes for microcontrollers and embedded networking and Internetworking www.esacademy.com/fag/classes				

In case there is problem in connecting the device, then change the T1 & T2 values by 50 & 100 respectively.

Step 4: In the main window, for the 1st section, use the following settings. COM port = COM1; Baud rate = 9600; Device = 89C51RD2Hxx; Oscillator Freq.(MHz) = 11.0592.

🗮 Flash Magic 📃 🗆 🗶				
File ISP Options Help	0			
1		2		
COM Port:	СОМ 1 🗾	Erase block 0 (0x0000-0x1FFF) Erase block 1 (0x2000-0x3FFF)		
Baud Rate:	9600 💌	Erase block 2 (0x4000-0x7FFF)		
Device:	89C51RD2Hxx 💌	Erase block 4 (0xC000-0xFFFF)		
Oscillator Freq. (MHz):	89C61X2 89C51RB2Hxx 89C51RC2Hxx	Erase all Flash+Security		
-3	89C51RD2Hxx			
Hex File:	89C662	Browse		
Last Modified	89C668	Size: Unknown		
.45				
✓ Verify after programming Set Security Bit 1 Fill unused Flash Set Security Bit 2				
Image: Generate checksums Image: Set Security Bit 3 Image: Generate checksums Image: Generate checksums Image: Generate checksums Image: Generate checksums				
Microcontrollers from Philips Semiconductors Main web page at:				
Step 5: In the **2**nd section, select the number of block to be erased; generally up till block 1 or 2 are needed for small code of program. These are code memory locks which needs to be erased to over write the new program on to them.

Flash Magic	
File ISP Options Help	
1 COM Port: COM 1 Baud Rate: 9600 Device: 89C51RD2Hxx Oscillator Freq. (MHz): 11.0592	2 Erase block 0 (0x0000-0x1FFF) Erase block 1 (0x2000-0x3FFF) Erase block 2 (0x4000-0x7FFF) Erase block 3 (0x8000-0x8FFF) Erase block 4 (0xC000-0xFFFF) Erase all Flash+Security
Hex File:	Browse
, Last Modified: Unknown	Size: Unknown
Verify after programming Set Security Fill unused Flash Set Security Generate checksums Set Security Execute G clks/cycle	Bit 1 Bit 2 Bit 3 Start
I echnical on-line articles about 8051 and XA pro www.esacademy.com/faq/docs	ogramming

Step 6: In the **3**rd section, you have to give the programming **HEX** file; so browse to the folder containing the HEX file and click OK. You can browse for sample HEX file provided along with the USDP, **serial_transmit.hex**.

🛱 Flash Magic		
File ISP Options Help		
1 COM Port: COM 1 Image: COM 1		
Hex File: vt. Ltd\ni2 Designs\Almighty Kit\Keil Demo\serial_transmit.hex Browse		
Last Modified: 11/7/2003 4:14:11 PM Size: 302 bytes		
4 Set Security Bit 1 Fill unused Flash Set Security Bit 2 Generate checksums Set Security Bit 3 Execute 6 clks/cycle Start		
Technical on-line articles about 8051 and XA programming		

Step 7: In the **4**th section set the **Execute** option and clear all other options. This option is to execute the code after programming the controller.

🛱 Flash Magic		
File ISP Options Help		
_1	2	
COM Port: COM 1 Baud Rate: 9600 Device: 89C51RD2Hxx Oscillator Freq. (MHz): 11.0592	Erase block 0 (0x0000-0x1FFF) Erase block 1 (0x2000-0x3FFF) Erase block 2 (0x4000-0x7FFF) Erase block 3 (0x8000-0x8FFF) Erase block 4 (0xC000-0xFFFF) Erase all Flash+Security	
Hex File: vt. Ltd\ni2 Designs\Almighty Kit\Kei	il Demo\serial_transmit.hex Browse	
Last Modified: 11/7/2003 4:14:11 P	M Size: 302 bytes	
4 5 □ Verify after programming □ Set Security Bit 1 □ Fill unused Flash □ Set Security Bit 2 □ Generate checksums □ Set Security Bit 3 ☑ Execute □ 6 clks/cycle		
Technical on-line articles about 8051 and XA pro	ogramming	
www.esacademy.com/faq/docs		

Step 8: Now in the 5th section press the **start** button, which in turn start the programming of controller. After displaying of **Finished** message on the bottom side of window, start working on your application.

Note: If any modification is there in the source code, then regenerate the HEX file and repeat steps 6 to 8. The above program works on windows 98/2000/XP platforms.

The above sample program works in conjunction with FPGA, so after programming the 89c51 controller configure the FPGA with the sample code provided for it. For more information refer the chapter **Using Add-on Modules**.

Programming PIC16F877 controller

Using the **ProgPIC** programmer, designers can program the PIC16F877 controller module. After building the HEX file from compiler, designers can refer the below shown steps to program the controller.

Insert the controller module on baseboard; connect the serial cable provided to module and PC's parallel port. Connect the +18V adaptor to Casio plug on the module. This adaptor is used to generate high voltage pulse to reset the PIC, remove this adaptor after programming the PIC controller.

Turn ON the power supply; now run the EXE of **ProgPIC** programmer on your PC. The below shown window will open up.

PIC16Fxxx - Progr File Port Erase	ammer TEST Options Help	
Action Read all Write Verify Code	 ✓ Write Code ✓ Write Data ✓ Write Config.Word ✓ Erase bef. Write ✓ Low-Volt-Prog. 	Controller PIC16C71 DEV-ID Config. Change 3FFF
Current File:	Activity:	Exit

The designer needs to set the programming port, device, configuration word, and couple of option to use the above programmer, once looking at the pictorial steps below, it will become more easy to use the programmer.

Step 1: Go to Port menu, and select the PP 0378H option to use the parallel port for programming.

PIC1	6Fxxx - Prog	ammer TEST	
PIC1 File	6Fxxx - Progr Port Erase PP 0278 PP 0378 PP 038C COM1 ✓ COM2 COM3 COM4 COM5	Ammer TEST Options Help Vite Code Vite Data Vite Config.Word Erase bef. Write	Controller PIC16C71 DEV-ID Config.
ſ	COM6 COM7 COM8	Activity:	Exit

Step 2: Now go to File menu and click **open file** option, there after you have to browse to your programming HEX file. You can select the sample program provided along with the USDP, **disp_PIC.hex**

PIC16Fxxx - Program	mer TEST	
File Port Erase Op	tions Help	
Open File Close File Exit Write	 Write Code Write Data Write Config.Word Erase bef. Write Low-Volt-Prog. 	Controller PIC16C71 DEV-ID Config. Change 3FFF
Current File:	Activity:	Exit

Step 3: Go to Controller option on main window, and select PIC part number as 16F877.

PIC16Fxxx - Program	nmer TEST	
File Port Erase Op	tions Help	
Action		Controller
Read all	🔽 Write Code	PIC16C71 -
	🔽 Write Data	PIC16F83
Write	🔲 Write Config.Word	PIC16F84A
	🔲 Erase bef. Write	Co PIC16F870 PIC16F871
Verify Code	🗖 Low-Volt-Prog.	PIC16F872 PIC16F873
		PIC16F876
Current File:	Activity:	PIC16F877
disp_PIC.HEX		Eixit

Step 4: The PIC controller works on the configuration word setting; the configuration word can be modified by clicking the change button on the Config. option. This will open a new window for setting the configuration word. For the sample program the configuration word value is **3D32 (in HEX)**, which you can fill in the window and press write to write in the controller. User can also make changes according to their design for changing the configuration word.

Config-	Word				×
Cor	nfig Word				
	Bit 13 🔽 CP1 Bit 12 🔽 CP0 Hex Valu	Bit 11 I DEBUG Bit 10 I - Bit 9 I WRT Bit 8 I CPD	Bit 7 LVP Bit 6 BODEN Bit 5 CP1 Bit 4 CP0 Read Write	Bit 3 I /PWRTE Bit 2 I WDTE Bit 1 I FOSC1 Bit 0 I FOSC0	
ID's ID 0	s 0 ID1 ID2 ID3 0 0 0	Read Write		Done	

Note: The configuration word can also be included in the source code.

Step 5: Press **Done** on the above window, and come back to main window. Here set the following option shown in the bottom picture. (Write code; write data; write config word; Erase before write).

PIC16Fxxx - Program	ner TEST	
File Port Erase Opti	ons Help	
Action Read all Write Verify Code	 Write Code Write Data Write Config.Word Erase bef. Write Low-Volt-Prog. 	Controller PIC16F877 DEV-ID Config. Change 3D32
Current File: disp_PIC.HEX	Activity:	Exit

Now the required settings are made to program the PIC microcontroller. Now press the **write** button to start programming the device. After display of **OK** on the activity window, start working on your application.

Note: The above program works on windows 98, to work on windows 2000/XP install the provided **DiportIO** driver.

The above sample program works in conjunction with FPGA, so after programming the PIC controller configure the FPGA with the sample code provided for it. For more information refer the chapter **Using Add-on Modules**.

Chapter 10 Using Add-on Modules

In this chapter user will cover the basics on using the add-on modules provided along with the USDP. The motive of chapter is to let the user know about the module, executions steps, block diagrams and details for using the modules.

Here is the list of modules to be covered under this chapter;

- 1. USDP Base Board
- 2. Xilinx FPGA Module.
- 3. Altera FPGA Module.
- 4. ADC/DAC Module.
- 5. 89c51RD2 Module.
- 6. PIC uC Module.
- 7. SRAM Memory Card.
- 8. Power Electronics Module.
- 9. General Purpose PCB.

1. USDP Base Board

USDP baseboard is used to interface the various modules with each other. Designers can use this baseboard for developing their application with switches, -segment displays, keypad and connectors provided.

Jumper and Header settings

Clock (JP1)

Pin 1	Clock
Pin 2	GCK0
Pin 3	Ground

To use the oscillator clock short pin 1 and 2. It is recommended to ground the GCK0 during programming.

JPDIS

These four jumpers are used to disable the 7-segment displays. Just remove the jumpers to isolate to displays from FPGAs.

JPA, JPB, JPC & JPD

These jumpers are used to select the I/O mode of the LEDs.

Total 32 configurable I/Os are provided on the USDP baseboard. To configure them as I/P user has to insert the jumper in between, and remove the jumper to make them as O/P.

Power Headers (J8 & J9)

There are power supply headers; users can connect the SMPS to any one of these headers. Two headers are provided just for the sake of convenience for connecting the SMPS from any of the directions.

SLOT SEL (JP3)

This is slot selector header. User can use the provided selector card for selecting the slot.

Slot 1	Slot 2
Short 1-2	Short 2-3

PLD SEL (JP2)

This is PLD vendor selector header. User can use the provided selector card for selecting the PLD vendors.

Xilinx	Altera
Short 1-2	Short 2-3

J7 (Keypad)

	/		
Pin No.	Signal	Pin No.	Signal
1	VCC	2	GND
3	SL3	4	SL2
5	SL1	6	SL0
7	RL3	8	RL2
9	RL1	10	RL0

Programming Port (J6)

User has to connect provided programming cable to this port and PC's parallel for programming of FPGAs.

Board Idents

In this section, idents of all boards are shown; users can refer this section for any component listing.



2. Xilinx FPGA Module

The Xilinx FPGA module contains a Spartan-II family FPGA with a choice of 50,000 gates or 200,000 gates FPGA. With a total pins 208 and maximum 147 user I/O pins. Users can develop designs requiring high density and higher number of I/Os with this FPGA.

There are total 96 I/O pins brought down to the base bus for add-on module connection. User can use this bus for their application design with the add-on module. Also 32 bits out this bus are been shared with configurable I/Os.

The bank (0) is been brought on DB-25 connector with its VREF pins. Designers can use this bank for different voltage signaling. Also this DB-25 connector can be used to interface with the PC's parallel port for communication.

We have provided a sample program, which implements the hardware of 3:8 decoder IC 74xx238, which is active high decoder with 3 enable signals.

Here is the procedure for using the above code with this FPGA module.

Procedure for using module

- ? Write source code in VHDL/Verilog using the Xilinx ISE series software.
- ? Pin lock the entity I/Os with configurable I/Os.
- ? Run the FPGA design flow, from synthesis to implementation of design on FPGA.
- ? Set the programming mode JTAG or slave serial in generate programming file option.
- ? Generate the BIT file.
- ? Insert the FPGA module on slot 1 or slot 2.
- ? Set the PROM bypass jumpers to position 2-3.
- ? Set the programming mode in **JTAG** or **slave serial**.
- ? Set the jumper position of configurable I/Os for input or output for entity pins.
- ? Connect the programming cable.
- ? Turn on the **USDP** power supply.
- ? Run the programming tool impact.
- ? Program the FPGA.
- ? Check the application.

Source code for FPGA

Source code name = decoder.vhd Pin lock file name = decoder.ucf

Note: For further information on Xilinx Spartan-II FPGAs, kindly refer the device datasheet.

Jumper Settings

Programming Mode selection Switch (U7)

Modes	MO	M1	M2
JTAG	1	0	1
Slave Serial	1	1	1
Master Serial	0	0	0

Mode selection jumpers (JP4, JP5, JP6, JP7)

Serial	JTAG
Short 1-2	Short 2-3

PROM BYPASS (J3 & J4)

To Chain PROM To Bypass PROM

Short 1 – 2 Short 2-3

Shorting 1-2 will bring the PROM in chain with FPGA. In this case user can configure the PROM and use for programming file storage.

Shorting 2-3 will remove the PROM from chain, and only FPGA would be connected to programming port.

Global Clock Buffers

Pin 1	GCK2	GCK3
Pin 2	GND	GND

Note: These buffers are extra from GCK0.



Xilinx FPGA Adaptor Board Ident

Spartan-II FPGA Pin detail

Device: XC2Sxx PQ208

Clock and Re	eset
Clock (GCK0)	80
GCK2	182
GCK3	185
Reset	5
N	

Note: **Reset** is active **LOW**.

	Configurable I/Os						
LA7	34	LB7	45	LC7	60	LD7	71
LA6	35	LB6	46	LC6	61	LD6	73
LA5	36	LB5	47	LC5	62	LD5	74
LA4	37	LB4	48	LC4	63	LD4	75
LA3	41	LB3	49	LC3	67	LD3	81
LA2	42	LB2	57	LC2	68	LD2	82
LA1	43	LB1	58	LC1	69	LD1	83
LA0	44	LB0	59	LC0	70	LD0	84

7 Segme	ents	Display Enable		Keypad Header		er	
segA	33	DISP1	17	SL0	8	RL0	6
segB	31	DISP2	18	SL1	15	RL1	10
segC	30	DISP3	20	SL2	9	RL2	7
segD	29	DISP4	21	SL3	16	RL3	14
segE	27						
segF	24						
segG	23						
segDP	22						

Parallel Port Connector (DB-25)				
Par1	206	Par10	192	
Par2	205	Par11	191	
Par3	204	Par12	188	
Par4	202	Par13	187	
Par5	201	Par14	203	
Par6	199	Par15	200	
Par7	195	Par16	189	
Par8	194	Par17	181	
Par9	193			
Pin 18 – 25 of DB-25 connector are ground				

Note: Both the FPGAs share the above I/Os. User has to take care, that **no two pins are defined as output**, as in that case there would be short on the bus and may damage FPGA I/Os.

User can define the following combination of FPGAs shared I/Os;

FPGA1	FPGA2	
Input	Input	Allowed
Output	Input	Allowed
Input	Output	Allowed
Output	Output	Not allowed

3. Altera FPGA Module

The Altera FPGA module contains an ACEX 1K family FPGA with a choice of 50,000 gates or 100,000 gates FPGA. With a total pins 208 and maximum 147 user I/O pins, users can develop designs requiring high density and higher number of I/Os.

There are total 96 I/O pins brought down to the base bus for add-on module connection. User can use this bus for their application design with the add-on module. Also 32 bits out this bus are been shared with configurable I/Os.

Few I/Os are also brought on DB-25 connector. Designers can use this connector to interface with the PC's parallel port for communication.

We have provided a sample program, which implements the hardware of 3:8 decoder IC 74xx238, which is active high decoder with 3 enable signals.

Here is the procedure for using the above code with this FPGA module.

Procedure for using module

- ? Write source code in VHDL/Verilog using the Altera Quartus series software.
- ? Pin lock the entity I/Os with configurable I/Os.
- ? Run the FPGA design flow, from synthesis to implementation of design on FPGA.
- ? Generate the BIT file for JTAG or serial mode.
- ? Insert the FPGA module on slot 1 or slot 2.
- ? Set the programming mode in **JTAG** or **slave serial**.
- ? Set the PROM bypass jumpers to position 2-3.
- ? Set the jumper position of configurable I/Os for input or output for entity pins.
- ? Connect the programming cable.
- ? Turn on the **USDP** power supply.
- ? Program the FPGA.
- ? Check the application.

Source code for FPGA

Source code name = decoder.vhd Pin lock file name = decoder.ucf

Note: For further information on Altera ACEX 1K FPGAs, kindly refer the device datasheet.

Jumper Settings

Programming Mode selection jumpers (J3, J8)

	MSEL1	MSEL0
Passive Serial (PS)	0	0
JTAG	0	1

For logic low short 2-3 and for logic short 1-2 for logic High

Mode selection jumpers (J4, J5, J6, J7)

Serial	JTAG
Short 2-3	Short 1-2

PROM BYPASS (J1 & J2)

To Chain PROM To Bypass PROM

Short 1 – 2 Short 2-3

Shorting 1-2 will bring the PROM in chain with FPGA. In this case user can configure the PROM and use for programming file storage.

Shorting 2-3 will remove the PROM from chain, and only FPGA would be connected to programming port.



ACEX FPGA Adaptor Board Ident

ACEX1K FPGA Pin detail

Device: EP1kxx PQ208

Clock &	Reset
Clock	79
Reset	180

Note: **Reset** is active **LOW**.

	Configurable I/Os						
LA7	70	LB7	60	LC7	46	LD7	36
LA6	71	LB6	61	LC6	47	LD6	37
LA5	73	LB5	63	LC5	53	LD5	38
LA4	74	LB4	64	LC4	54	LD4	39
LA3	75	LB3	65	LC3	55	LD3	40
LA2	85	LB2	67	LC2	56	LD2	41
LA1	86	LB1	68	LC1	57	LD1	44
LA0	87	LB0	69	LC0	58	LD0	45

7 Segme	ents	Display E	nable	Keypad Header			er
segA	31	DISP1	15	SL0	8	RL0	3
segB	30	DISP2	16	SL1	13	RL1	11
segC	29	DISP3	17	SL2	9	RL2	7
segD	28	DISP4	18	SL3	14	RL3	12
segE	27						
segF	26						
segG	25						
segDP	24						

Parallel Port Connector (DB-25)				
Par1	205	Par10	193	
Par2	203	Par11	192	
Par3	202	Par12	191	
Par4	200	Par13	190	
Par5	199	Par14	189	
Par6	198	Par15	187	
Par7	197	Par16	186	
Par8	196	Par17	179	
Par9	195			
Pin 18 – 25 of DB-25 connector are ground				

Note: Both the FPGAs share the above I/Os. User has to take care, that **no two pins are defined as output**, as in that case there would be short on the bus and may damage FPGA I/Os.

User can define the following combination of FPGAs shared I/Os;

FPGA1	FPGA2	
Input	Input	Allowed
Output	Input	Allowed
Input	Output	Allowed
Output	Output	Not allowed

4. ADC/DAC Module

The ADC/DAC module provided along with the USDP has 4 channel ADC, 2 channel DAC on board. This module is directly connected to the FPGAs, and does not have direct connections with other modules. To use this module user has to design ADC/DAC controller in the FPGA to get the data from ADC and to put data on DAC channels.



Block diagram of the ADC

The above diagram shows how the interconnection can be done with the ADC/DAC module. The sampled data from ADC would be latched inside the FPGA, and as the module has on board latches for DAC, designer can control these latches and store data over there for DAC conversion.

Designer can use the sampled value from ADC for his signal processing and after that he can reconstruct the wave by storing the data in the on board latches for DAC.

We have provided a sample controller for ADC/DAC module that works as feed though circuit between ADC and DAC. User can use this source code for the check of module.

Procedure for using module

- ? Add the provided source code Xilinx ISE or Altera Quartus series software.
- ? Run the procedure for using FPGA module (refer previous pages).
- ? Insert the ADC/DAC module in Slot 3, Slot 4 or Slot 5.
- ? Connect the signal generator or analog source to ADC's channel 0.
- ? Connect the CRO probe to DAC's channel 0.
- ? Turn on the **USDP** power supply.
- ? Program the FPGA.
- ? Check the application.

Source code for FPGA

Source code name	= ADC_DAC_feed.vhd
Pin lock file name	= ADC_DAC_feed.ucf

For further working of ADC and DAC, kindly refer the datasheet provided along with the protoboard.

87

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A17	D0	92	87	A18	D1	90	86
A19	D4	96	94	A20	D5	95	90
A21	D8	101	98	A22	D9	100	97
A23	EN1	111	102	A24	EN2	104	101
A25	DB0	115	111	A26	DB1	114	110
A27	DB4	121	115	A28	DB5	120	114
A29	INT	127	122	A30	RDY	126	121

ADC/DAC Module Pin Detail*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B17	D2	89	3	B18	D3	88	4
B19	D6	94	89	B20	D7	93	88
B21	D10	99	96	B22	D11	97	95
B23	A0	103	100	B24	A1	102	99
B25	DB2	113	109	B26	DB3	112	108
B27	DB6	119	113	B28	DB7	116	112
B29	CS	125	120	B30	RD	122	119

Header Details

DAC O/P (J1)

D! 1	DAC Channel 1
PINI	DAC Channel I
Pin 2	DAC Channel 2
Dia 0	
PIN 3	Ground

ADC I/P (J2)

Pin1	ADC Channel 1
Pin 2	ADC Channel 2
Pin 3	ADC Channel 3
Pin 4	ADC Channel 4
Pin 5	Ground

JP2

This jumper is for setting the reference voltage. This module has onboard reference voltage generation, to use that short 2-3.

Note: ADC/DAC work in linear monotonic format with -VREF = 00000000 and +VREF = 11111111. So take care of numbering system while designing the logic.



ADC/DAC Card Board Ident



MAX154 ADC Pin out



ADC7541A DAC Pin out

USDP User Manual

5. 89c51RD2 Module

This module is based on renowned industry standard 8051 architecture based controller. The uC is 89c51RD2 controller from Philips, which is **In System Programmable (ISP)**.

All the I/Os have been brought on the edge connector and are connected with the FPGAs, user can program the uC and use it with the FPGA. The timer & interrupt pins are brought on the separate headers, which can be used to interface with the external world.

The module also contains on board RTC and EEPROM, which user can use for his application.

Procedure for using module

- ? Write source code in C or assembly language in keil compiler or some other compiler.
- ? Generate the HEX file.
- ? Connect the serial cable.
- ? Turn on the **USDP** power supply.
- ? Program the microcontroller using the Flash Magic programmer.
- ? Configure the FPGA for microcontroller interface logic.
- ? Reset the controller through FPGA.
- ? Check the application.

We have provided a sample program, which takes data from FPGA on one port and sends it serially through the RS-232 port.

For further information on 89c51RD2, kindly refer the datasheet provided along with the protoboard.

Source code for FPGA

Source code name = feed_89c51.vhd Pin lock file name = feed_89c51.ucf

Source code for 89c51

Source code name = serial_transmit.c Programming file name = serial_transmit.hex

Note: For further information on using Keil compiler and Flash Magic software, kindly refer the chapters Using EDA tools & Configuration.

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A31	AD0	133	127	A32	AD2	132	126
A33	AD4	139	134	A34	AD6	136	133
A35	A8	143	139	A36	A10	142	138
A37	A12	149	146	A38	A14	148	142
A39	PORT1_1	159	150	A40	PORT1_3	158	149
A41	PORT1_5	163	163	A42	PORT1_7	162	162
A43	INT0	168	167	A44	INT1	167	166
A45	ALE	173	174	A46	FRST	172	173

89c51 Module Pin Detail*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B31	AD1	131	125	B32	AD3	128	123
B33	AD5	135	132	B34	AD7	134	129
B35	A9	141	136	B36	A11	140	135
B37	A13	147	141	B38	A15	144	140
B39	PORT1_2	157	148	B40	PORT1_4	150	147
B41	PORT1_6	161	152	B42	PORT1_8	160	151
B43	T0	166	165	B44	T1	164	164
B45	RD_51	170	172	B46	WR_51	169	168

Jumper & Header settings

Square Wave o/p (JP3)

Pin1	1Hz clock from RTC.
Pin 2	Ground

Interrupt/Timer (JP4)

Pin1	Interrupt 0 (INT0) I/P
Pin 2	Interrupt 1 (INT1) I/P
Pin 3	Timer 1 O/P
Pin 4	Timer 0 O/P

JP1 & JP2

JP1	Port1_1	SDA
JP2	Port1_2	SCLK

JP1 & JP2 are selection jumper for serial clock and data from RTC and EEPROM. Both these signals are shared with the base bus of USDP. So if you are these two ports of 89c51 with the base bus then remove the jumpers from the JP1 & JP2 to isolate them from the base bus.



89c51 Microcontroller Card Board Ident



89c51RD2Hxx Pin out

6. PIC uC Module

This module use RISC based architecture controller. The PIC16F877 controller is from Micro Chip, which is **In System Programmable (ISP).** This is leading controller used in industries for product design and is widely used for his ease of use and enriched features.

All the I/Os of controller have been brought on the edge connector and are connected with the FPGAs; user can program the uC and use it with the FPGA. The interrupt pins are brought on the separate headers, which can be used to interface with the external world.

The module also contains on board RTC which user can use for his application.

Procedure for using module

- ? Write source code in C or assembly language in compiler.
- ? Generate the HEX file.
- ? Connect the PIC parallel cable.
- ? Turn on the **USDP** power supply.
- ? Program the microcontroller using the PIC PRO programmer provided (also use the +18V adaptor).
- ? Remove the +18V adaptor after programming the PIC.
- ? Configure the FPGA for microcontroller interface logic.
- ? Reset the controller through FPGA.
- ? Check the application.

We have provided a sample program, which runs a message on the 7-segment display though the FPGA. All the control is from PIC controller and the FPGA is working like a feed through circuit.

For further information on **PIC16F877**, kindly refer the datasheet provided along with the protoboard.

Source code for FPGA

Source code name = feed_PIC.vhd Pin lock file name = feed_PIC.ucf

Source code for PIC

Source code name = disp_pic.c Programming file name = disp_pic.hex

Note: For further information on using **compiler** and **PIC PRO** software, kindly refer the chapters **Using EDA** tools & **Configuration**.

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A1	AN0/RA0	36	34	A2	AN1/RA1	37	35
A3	AN2/RA2	40	41	A4	AN3/RA3	41	42
A5	AN4/RA4	46	45	A6	AN4/RA5	47	46
A7	RD0	55	49	A8	RD1	56	57
A9	RD2	60	60	A10	RD3	61	61
A11	RD4	65	67	A12	RD5	67	68
A13	RD6	70	71	A14	RD7	71	73
A15	MCLR/	75	81				
A7 A9 A11 A13 A15	RD0 RD2 RD4 RD6 MCLR/	55 60 65 70 75	49 60 67 71 81	A8 A10 A12 A14	RD1 RD3 RD5 RD7	56 61 67 71	57 61 68 73

PIC 16F877 Micro controller Pin details*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B1	RC0	38	36	B2	RC1	39	37
B 3	RC2	44	43	B 4	RC3	45	44
B 5	RC4	53	47	B6	RC5	54	48
B7	RC6	57	58	B8	RC7	58	59
B9	RB0/INT	63	62	B10	RB1	64	63
B11	RB2	68	69	B12	RD3	69	70
B13	RB4	73	74	B14	RB5	74	75
B15	RB6	86	83	B16	RB7	87	84

Jumper & Header Settings

Programming jumper

JP9, JP10 & JP11 are programming selection jumpers. User has to short 1-2 during programming and after programming the PIC controller, short 2-3.

JP1

JP1		,	IP2	J	IP3
Pin1	1Hz clock from RTC.	Pin1	Ra4	Pin1	PCRX
Pin 2	Ground	Pin 2	Rb0/int	Pin 2	PCTX
		Pin 3	Ground	Pin 3	Ground

RX2 (JP5)

Port RC7 is used for RX channel of RS-232, also this port pin is shard with base bus and by removing this jumper you can isolate the RS-232 from the base bus.

TX (JP6)

Port RC6 is used for TX channel of RS-232, also this port pin is shard with base bus and by removing this jumper you can isolate the RS-232 from the base bus.

SDA (JP7)

Port RC4 is used for serial data for RTC, also this port pin is shard with base bus and by removing this jum per you can isolate the RTC from the base bus.

SCLK (JP8)

Port RC3 is used for clock for RTC, also this port pin is shard with base bus and by removing this jumper you can isolate the RTC from the base bus.

JP9, JP10 & JP11

During programming the PIC controller, **short pins 2-3**, and after programming the controller, **short pins 1-2** for using with FPGA.

<u> </u>
AN0
AN1
AN2
AN3
AN4
AN5
AN6
AN7
Ground

Analog I/Ps (Header for inbuilt ADC I/Ps for PIC controller)



PIC-16F877 Microcontroller Board Ident



PIC 16F877 Controller Pin out

7. SRAM Memory Card

User gets a SRAM memory module along with the USDP. This has the total 2MB of data storage. The SRAM module contains four 512K x 8 memory chips which user can use according to his application requirement. All the I/Os of memories are brought on the edge connector, sharing with the PIC slot bus.

This creates choice for user for inserting a memory card or PIC card in any 3 of the slots. User should not insert both, the SRAM module and PIC module together on the board, as there may be conflict on the bus.

SRAM module contains on board address decoder for generating chip select for four memory chips, this makes the module more easier in use by just providing the address lines and control signals for data transfer.



SRAM Module Pin details*

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
A1	A16	36	34	A2	A17	37	35
A3	A18	40	41	A4	D0	41	42
A5	D1	46	45	A6	D2	47	46
A7	D3	55	49	A8	D4	56	57
A9	D5	60	60	A10	D6	61	61
A11	D7	65	67	A12	RD/	67	68
A13	WR/	70	71	A14	A19	71	73
A15	A20	75	81				

		Acex 1K	Spartan-II			Acex 1K	Spartan-II
B1	A0	38	36	B2	A1	39	37
B3	A2	44	43	B4	A3	45	44
B5	A4	53	47	B6	A5	54	48
B7	A6	57	58	B8	A7	58	59
B9	A8	63	62	B10	A9	64	63
B11	A10	68	69	B12	A11	69	70
B13	A12	73	74	B14	A13	74	75
B15	A14	86	83	B16	A15	87	84



SRAM Memory Module Board Ident



K6T4008C1C SRAM Memory Pin out

8. Power Electronics Module

Power Electronics Module is external module provided along with USDP for power electronics based applications.

The module is be divided in 6 sections,

- ? Relays
- ? Step down transformer.
- ? Isolated O/Ps.
- ? IGBT.
- ? High current rectifier.
- ? Stepper motor controller.

We would describe individual section usage; further designer can join the parts together depending on their application.

? Relay section

Two optically isolated relays have been provided onboard. User can turn relays ON by applying logic high on the given control I/P pins.

The relay pins are taken out on terminal headers from where user can interface his circuit for mechanical switch operation. For example user can connect light bulb, heater, buzzer, etc. for visual demonstration.

? Step down transformer

A divide by 100 transformer is been provided onboard, which user can use for line monitoring applications.

User can connect the AC line to its I/P header (JP4), which in turn get a divided by 100 O/P voltage on its secondary winding header (JP5). The secondary winding is center tapped, that means user will get the divide by 100 voltage on the outer pins of winding, the center tap can be used as a reference point.

User can connect this divided O/P voltage to ADC and use the digitized signal for the processing of AC line signal. For example designer can use for frequency measurement of AC line, voltage measurement, line monitoring, etc applications.

? Isolated O/Ps

Five optically isolated O/Ps are provided onboard. User can drive the I/P signal pins high to turn ON the optical isolator transistor. The emitter of optical transistors are been shorted together and name as IGND, user has to use all O/Ps with this common signal. Designer has to make digital ground (DGND) & IGND common if he is working with the same reference voltage levels.

The collector & IGND of optical transistors are brought on the terminal header (JP6) from where user can interface his logic.

? IGBT

A high current rating IGBT is provided onboard for motor control or high current based applications. The IGBT pins are brought on the terminal header (J2) from where user can interface his logic.

As the triggering logic varies from application to application, we have not given the circuit onboard, and user has to design and interface his own triggering logic to drive the IGBT ON/OFF.

Note: While designing the triggering logic of IGBT, use optical isolator in between the FPGA/uC and IGBT. Do not short the analog ground (AGND) & digital ground (DGND) directly, instead use a fuse in between both ground to protect from over flow of current in digital ground.

? High current rectifier

A High current rectifier is provided onboard, which can be used for rectifying AC lines upto 10 Amps of current. Designer can apply AC I/P to rectifier from its terminal header (JP1) and get the rectified DC O/P from terminal header (JP3).

Note: Take care while handling the rectifier terminal headers, as due to high voltage you may get shock, so please use rubber glove while handling the power module.

? Stepper motor controller

Power module has onboard transistor based stepper motor driver circuit. Designers can interface their uni-polar stepper motor on the provided terminal header (JP8). This circuit can drive motors of +12V/0.5 amps. User can apply phase control signals to the stepper motor I/P pins in proper sequence to run the stepper motor. The sequence of signal will determine the motor direction and stepping mode.

Note: Connect the motor winding the proper sequence, else the motor won't rotate and will keep on vibrating.

Source codes for Power Module (FPGA)

? Relay section	
Source code name	= relay.vhd
Pin lock file name	= relay.ucf

? Stepper motor	section
Source code nam	e = stepper.vhd
Pin lock file name	= stepper.ucf

Note: Take extensive care while handling the power module in high voltages, as there may be chance of shock in case of improper handling.

Check your design before connecting with the power module from your instructor.

ni logic will not take responsibility in case of any damages due to false design practice and improper handling.

Power Module Pin Detail

Power module is independent of USDP baseboard, and can me mapped to any of the I/Os of the adaptors available.

Here by are the header details of the power module.

JP1 (AC I/P)				
Pin No. Signal				
1	Line			
2	Neutral			

JP2 (DC O/P)			
Pin No.	Pin No. Signal		
1 VDC			
2 AGND (Analog ground)			

JP4 (AC Line, transformer)		
Pin No. Signal		
1	Primary1	
2 Primary2		

JP5 (Step O/P / 100)		
Pin No. Signal		
1	Secondary1	
2	Common	
3	Secondary2	

JP8 (Stepper Motor)			
Pin No. Signal			
1	W1		
2	W2		
3	W3		
4	W4		
5	DGND (digital ground)		

JP6 (Optically Isolated O/Ps)		
Pin No.	Signal	
1	OP1	
2	OP2	
3	OP3	
4	OP4	
5	OP5	
6	IGND (Isolated ground)	
7	DGND (Digital ground)	

JP7 (Relay1)		JP9 (Relay2)
Pin No.	Signal	Signal
1	Common	Common
2	NO	NC
3	NO	NC

J2 (IGBT)			
Pin no.	Signal	Description	
1	G	Gate	
2	С	Collector	
3	E	Emitter	

JP10 (FPGA Interface)				
Pin No.	Signal	Acex 1K	Spartan-II	Description
1	IP1	205	206	Isolated I/P 1
2	IP2	203	205	Isolated I/P 2
3	IP3	202	204	Isolated I/P 3
4	IP4	200	202	Isolated I/P 4
5	IP5	199	201	Isolated I/P 5
6	RELAY1	198	199	Relay control 1
7	RELAY2	197	195	Relay control 2
8	W2	196	194	Winding 2 control
9	W1	195	193	Winding 1 control
10	W4	193	192	Winding 4 control
11	W3	192	191	Winding 3 control
12-16	DGND (Digital ground)			

Note: Connect the power electronics module to FPGA with the provided cable only.



IGBT 1MB60D Circuit Schematic



Opto-coupler MCT2E Pin out



Power Module Board Ident

9. General Purpose PCB

General Purpose PCB can be used for glue logic design, analog circuit, or any other custom circuit interface with the USDP modules.

The LCD header provided on the PCB can be used to drive the LCD module provided along with USDP. The header (JP6) connects with the LCD module, and the LCD I/Os header (JP13) can be connected to the general I/Os provided on board. This gives flexibility to interface the LCD with any one of the USDP modules.

JP6 (LCD Header)

Pin No.	Signal	Pin No.	Signal
1	GND	2	+5V
3	NC	4	RS
5	NC	6	EN
7	D0	8	D1
9	D2	10	D3
11	D4	12	D5
13	D6	14	D7
15	NC	16	NC

JP1 (LCD I/Ps)

Pin No.	Signal	Pin No.	Signal
1	D0	2	RS
3	D2	4	EN
5	D4	6	D1
7	D6	8	D3
9	D7	10	D5


General Purpose PCB Board Ident

Chapter 11 Designing Application on USDP using provided modules

Any electronic system is developed for an application solving a problem or giving ease to product usage. A system can be defined as "A group of independent but inter related elements comprising a unified whole". Or "A processing platform, where all element work together for a goal, with the given instructions. "

An basic electronic system (see figure below) consists of a master which can be microprocessor or a microcontroller, a memory store to the instructions or data, analog circuitry to interface with the external world, digital logic to process the data at high speed and power electronics components to control high voltage/current peripherals.



Block diagram of electronic system

With the above basic components, designer can develop an electronic system for any application. More or less few components can be added or removed, but with the today's market scenario and product requirement, designers need the above system blocks to be in their system.

Now we need to see that what are the design steps through which a system passes before getting into the market or its deployment.

Design Flow / steps





The above flow contains the basic steps, which are majorly followed in industry. Looking at the above steps the most problematic stages are prototyping the system schematic and testing the hardware, which consumes the maximum amount of time.

As the prototyping is generally done on breadboards or the general purpose PCB, this takes intensive care on making it and many times there are errors in building the circuit. Also during testing, designers should be given flexibility to modify the schematic instantly and to carry various tests for the available schematic.

And it is hectic task to design with these traditional steps, also they don't offer f lexibility and designers don't get the privilege to market the product in short span of time.

Now here **USDP** gives the advantage over traditional methods of design and other protoboards available. As it is universal platform and provided nearly all the modules for system design, designers can just plug the required modules and design/verify their application in very short span of time.

USDP also serves as a very good platform to train students on concept of electronic system designing. As all the basic technology platforms are integrated on **USDP** (VLSI, Microcontroller, power electronics, communication, etc), students can work on projects where they can have real life practical experience for system design. This will make them more mature about practical concepts of electronics, and give them chance to learn real life project development.

Also the modules are independent to each other and do not depend on the base board, thus any further modules which **ni logic** will introduce can be plugged on existing system and up gradation of **USDP** can be easily done.

Only the designer has to take care that the all the modules are connected with FPGA independently, thus user has to included FPGA in their designs, but further after verifying logic, they can exclude it while going for final product design.

Chapter 12 An Application Implementation on USDP

In this chapter we take an example for developing a real life example on USDP.

Application: Access control system

Today many applications are developed for security and access controlling. The basic applications of access control can be developed and prototyped on USDP. The basic model consists of keypad interface for password entering, solenoid for door open & close which can be replaced with relays here, user display for welcome notes, menus and messages displays.

Modules to be used

89C51 module, Xilinx FPGA module, LCD module, general purpose PCB, Power module and keypad.



Application description:

We keep the example very simple; user reaches to gate and reads the message, "Enter the Password", then he has to enter the 3-digit password (0 - 9) to open the gate (relay will indicate the gate opening), after entering the password he has to enter the "F1" key, which in turn will take the password. If the password is right then the relay will turn ON, and LCD will display "Password OK". If the password is wrong then the relay will remain OFF, and a LED will be glown indicating false password and LCD will display "Password Not Matched".

This is a simple access control system, further user can modify the given source codes to make it more complex in terms of number of trials, buzzer O/Ps, and change of password, etc.

Password will be "143".



Beginning with application:

User has to get together the modules specified, and use them to run the application on USDP. Here is the procedure to be followed to begin with application:

- ? Insert 89c51 Module on USDP
- ? Program it with the given programming file **access_control.hex**.
- ? Turn off the USDP power supply.
- ? Insert general purpose PCB in USDP & make connections with LCD module.
- ? Interface the USDP with power module and keypad.
- ? Now insert the Xilinx FPGA module in USDP, and program it with the given source code access_control.vhd.
- ? Reset the system and check the application.

Designers can use the sample codes provided along with USDP. The list of codes to be used for this application is listed below. User can refer these codes for their other applications and modify accordingly to their requirements.

Source code for FPGA

Source code name	= access_control.vhd
Pin lock file name	= access_control.ucf

Source code for 89c51

Source code name	= access_control.c
Header file name	= LCD_routine.c
Programming file name	= access_control.hex

Note: Save the LCD_routine.c file in your project folder while compiling the access_control.c file.

Chapter 13 Sample Codes

Here is the list of sample codes provided along with the USDP.

ADC/DAC Module

Code description:

This code works as feed through circuit between the ADC and DAC. The data coming from ADC is stored in latch and given to DAC for reconstruction of signal.

Working with code:

User has to insert ADC/DAC module, connect the analog signal on the ADC1 channel, and CRO probe to DAC1 channel (user can modify the code for channel selection). Program the Xilinx/Altera FPGA module for the given VHDL source code and pin constraint file. After programming the FPGA, check the waveforms on the CRO, which would be nearly the same analog signal, the difference would be due to sampling interval and amplitude difference due to resolution of ADC and DAC.

Source code for FPGA Source code name = ADC_DAC_feed.vhd Pin lock file name = ADC_DAC_feed.ucf

89c51 Module

Code description:

The 89c51 source code takes data from PORT0 and transmits it to serially to PC whenever there is change on the port data.

Working with code:

Program the 89c51 with the given HEX file, there after program the FPGA with the provided feed through circuit between switch and 89c51. After programming FPGA, change the position of LD7 to LD0 to observe the changes on the serial port of PC.

To the read he data from serial port of PC, users can write a program in 'C', or can use the COM port reader software provided along with USDP.

Source code for FPGASource code namePin lock file name= feed_89c51.ucf

Source code for 89c51 Source code name = serial_transmit.c Programming file name = serial_transmit.hex

PIC Module

Code description:

This source code has a message-displaying program. This displays **"HI lam USDP"** on the 7-segment displays. Users can use this code for their reference or modify according to their requirement.

Working with code:

All the PIC I/Os are shared with the LEDs also, so remove all the jumpers of switches so that there is no conflict after programming the controller. Now, Program the PIC microcontroller using the provided programmer software. Use the HEX file provided with the USDP.

There after program the FPGA with the provided feed through circuit VHDL code. This VHDL code acts like the feed through between the reset switch and the 7-segment displays. After programming the FPGA, reset the controller and check the message display on the 7-segment displays.

Note: As the memory module also shares the same bus with PIC, so remove the memory module while using the PIC module.

Source code for FPGA	
Source code name Pin lock file name	<pre>= feed_PIC.vhd = feed_PIC.ucf</pre>

Source code for PIC Source code name = disp_pic.c Programming file name = disp_pic.hex

Power Electronics Module

Stepper Motor Controller Section

Code description:

This source code can control uni polar stepper motors. The code can control the direction, speed and stepping of motor. This can be done with the help of switches provided on board.

Working with code:

Insert he FPGA module on USDP slot, connect the power module interface cable with FPGA through the parallel port connector provided.

Connect the stepper motor winding on the header (JP8), while connecting connect the winding the same phase as mentioned on the motor.

Program the FPGA with the given source code, and control the motor signals from the switches.

Source code for FPGA	
Stepper motor section	
Source code name	= stepper.vhd
Pin lock file name	= stepper.ucf

Relay section

Code description:

This source code can control the relays provided on the power module. User can turn ON the relay by providing logic High on the relay I/P pins.

Working with code:

Insert he FPGA module on USDP slot, connect the power module interface cable with FPGA through general purpose PCB or the parallel port connector provided. In this case user has to take care of pin assignment. Connect your application signals on the relay headers (JP7 & JP9), while connecting take care for Normally Open (NO), Normally Close (NC) and Common (COM) ports of relay.

Program the FPGA with the given source code, and control the relays from the switches.

Source code name	= relay.vhd
Pin lock file name	= relay.ucf

Keypad Controller

Code description:

This code is the 4x4 keypad controller. Users can press the keys and see the key value on the 7-segment displays. Further users can modify this code for their applications.

Working with code:

Insert he FPGA module on USDP slot, connect the keypad with the cable provided. Program the FPGA with the given source code. Press the keys of keypad and check the displayed value on the 7-segment displays.

Source code name = keypad.vhd Pin lock file name = keypad.ucf

LCD Module

Code description:

This code is generalized code written in 'C' for message displaying on LCD module. The code written is for 89c51RD2 controller. At bottom of source code, there is a **dispstr** function, which displays the written message in its body on LCD. Users can modify this **text** line to change the message to be displayed.

Working with code:

Insert the 89c51 module in USDP; connect the LCD module to 89c51 through the general purpose PCB. Program the 89c51 controller with the given HEX code, and check the message displayed.

Source code name = LCD_USDP.c Programming file name = LCD_USDP.hex

Access control application

Code description:

Sample codes are provided along with USDP for access control application. Samples are provided for FPGA and 89c51 controller. The FPGA consists the control logic, and controller has the LCD logic.

Working with code:

Insert the 89c51 module in USDP; connect the LCD module to 89c51 through the general purpose PCB. Connect the keypad to its header; make connections with the power electronics module through general purpose PCB.

Program the 89c51 controller with the programmer provided; turn OFF the power supply, insert the FPGA module & program it with the given source code and pin lock file.

Source code for FPGA	
Source code name	= access_control.vhd
Pin lock file name	= access_control.ucf
Source code for 89c51	
Source code name	= access_control.c
Header file name	= LCD_routine.c
Programming file name	= access_control.hex

Chapter 14 Glossary of Terms

ASIC (Application Specific Integrated Circuit)

A custom integrated circuit designed specifically for one end product or a closely related family of end products. Analog Digital and Mixed Signal

Analog, Digital and Mixed Signal

A circuit used to count the number of events is generally digital. Sometimes chip are called mixed signal chips which means that they contain both analog and digital circuits

Analog-to-Digital Converter (ADC)

An electronic circuit that converts a continuously varying signal (temperature, pressure, voltage, etc.) into digital zeroes and ones that can be processed by a microprocessor or microcontroller. Converts an analog signal sample to a digital representation suitable for digital processing and switching.

Asynchronous

Asynchronous system (computer, circuit, device) is one in which events are not executed in a regular time relationships. They are timing independent. Each event or operation is performed upon receipt of a signal generated by the completion of a previous event or operation, or upon availability of the system resources required by the event or operation.

С

Common programming language used in science, engineering and DSP. Also comes in the more advanced C++.

Concurrency

The ability of an electronic circuit to do several (or at least two) different things at the same time. Contrast with computer programs, which usually execute only one instruction at a time unless the program is running on a processor with multiple, concurrent execution units.

Combinational Logic

Combinational logic is purely functional logic, which does not maintain any internal state. Thus it will provide the same result for the same input no matter what sequence the input is sent.

CPLD (Complex Programmable Logic Device)

A programmable IC which is more complex than the original Programmable Logic Devices such as AMD's (originally MMI's) PALs but somewhat less complex than Field Programmable Logic Arrays.

Digital Signal Processor (DSP)

A processor system specialized for the computation of signal processing algorithms. It usually consists of many programmable processor elements interconnected via networks to each other and to memory, sensors, displays and other external devices. It is often distinguished from general purpose-or data processors in that is must operate in real-time; it often has a much higher data input rate. and it usually must perform a higher percentage of mathematical, often floating-point, operations.

Digital-to-Analog Converter (DAC)

A circuit that translates a signal from a numeric, digital representation used by microprocessors and microcontrollers into an analog signal. Converts a digital word to an analog value.

EDIF (Electronic Design Interchange Format)

A standard representation format for describing electronic circuits, used to allow the interchange of circuit design information between EDA tools.

FPGA (Field Programmable Gate Array)

An integrated circuit containing a large number of logic cells or gates that can be programmably configured after the IC has been manufactured. Some FPGAs use fuses for this programming and others store the configuration in an on chip EEPROM or RAM memory. Fuse programmed parts cannot be reprogrammed so they can only be configured once. EEPROM based FPGAs can be erased and reprogrammed so they can be configured many times. RAM based FPGAs can be reconfigured quickly, even while the circuit is in operation.

Finite Impulse Response (FIR)

An impulse response that has a finite number of nonzero values. Often used to indicate that a filter is carried out by using convolution, rather than recursion.

HDL (Hardware Description Language)

A synthetic computer based language used for the formal description of electronic circuits. An HDL can describe a circuit's operation, its design, and a set of tests to verify circuit operation through simulation. The two most popular digital HDLs are VHDL and Verilog. An analog HDL called AHDL is under development by many vendors. HDLs make it easier to develop very large designs through formal software engineering methods that define ways to divide a large team project into smaller modules that can be implemented by individual team members.

Hardware/Software Codesign

The simultaneous development of product hardware and software. This design approach is more difficult than a serial design which first develops the hardware and then the software that will run on the hardware but the benefit is a reduced time to market. To develop software before hardware is ready, software developers often create a behavioral model of the hardware which can run the software and thus prove its function.

Interrupt

An input to a processor that signals the occurrence of an outside event; the processor's response to an interrupt is to save the current machine state and execute a predefined subprogram.

The subprogram restores the machine state on exit and the processor continues in the original program

Joint Test Action Group (JTAG)

The Joint Test Action Group. This group created the foundation for the IEEE work.

Set of specifications that enable board and chip level functional verification of a board during production. Committee that established the Test Access Port (TAP) and boundary-scan architecture defined in IEEE Standard 1149.1-1990.

Liquid-Crystal Display (LCD)

The screen technology commonly used in notebook and smaller computers

Logic

The sequence of functions performed by hardware or software. Hardware logic is made up of circuits that perform an operation. Software logic is the sequence of instructions in a program.

Moore's Law

An empirical law developed and later revised by Intel's Gordon Moore which predicts that the IC industry is capable of doubling the number of transistors on a silicon chip every 18 months (originally every year) resulting in declining IC prices and increasing performance. Most design cycles in the electronics industry including embedded system development firmly rely on Moore's law.

Net List

A computer file (sometimes a printed listing) containing a list of the signals in an electronic design and all of the circuit elements (transistors, resistors, capacitors, ICs, etc.) connected to that signal in the design.

PLCC (Plastic Leaded Chip Carrier)

A low cost IC package (usually square). PLCCs have interconnection leads on either two (usually only for memory chips) or all four sides (for logic and ASIC chips).

PLD (Programmable Logic Device)

The generic term for all programmable logic ICs including PLAs (programmable logic arrays), PALs, CPLDs (complex PLDs), and FPGAs (field programmable gate arrays).

Pipelining

Splitting the CPU into a number of stages, which allows multiple instructions to be executed concurrently

PROM (Programmable Read Only Memory)

An integrated circuit that store programs and data in many embedded systems. PROM stores and retains information even when the power is off but it can only be programmed or initialized once.

RTL (Register Transfer Level or Register Transfer Logic)

A register level description of a digital electronic circuit. Registers store intermediate information between clock cycles in a digital circuit, so an RTL description describes what intermediate information is stored, where it is stored within t he design, and how that information moves through the design as it operates.

Simulation

Modeling of an electronic circuit (or any other physical system) using computer based algorithms and programming. Simulations can model designs at many levels of abstraction (system, gate, transistor, etc.). Simulation allows engineers to test designs without actually building them and thus can help speed the development of complex electronic systems. However, the simulations are only as good as the mathematical models used to describe the systems; inaccurate models lead to inaccurate simulations. Therefore, accurate component models are essential for accurate simulations.

Synchronous

A digital circuit where all of the operations occur in lock step to a master clock signal A mode of transmission in which the sending and receiving terminal equipment are operating continually at the same rate and are maintained in a desired phase relationship by an appropriate means An operation or operations that are controlled or synchronized by a clocking signal.

Synthesis (also Logic Synthesis)

A computer process that transforms a circuit description from one level of abstraction to a lower level, usually towards some physical implementation. Synthesis is to hardware design what compilation is to software development. In fact, logic synthesis was originally called hardware compilation.

System-on-a-Chip (SoC)

Combining several chips with different functions onto one, single chip.

ТСК

Test Clock, a TAP pin used to supply clocks to the TAP Controller.

TDI

Test Data In, a TAP pin used to shift the test data in to the TAP Controller.

TDO

Test Data Out, a TAP pin used to shift the test data out from the TAP Controller.

TMS

Test Mode Select, a TAP pin that provides the stimulus to change the state of the TAP Controller.

USDP

Universal Development Platform, a integrated platform where designers can put different technology modules all together for their system design.

User Constraints File (UCF)

A user created ASCII file for storing timing constraints and location constraints for a design implementation.

Verilog

A hardware description language developed by Gateway Design Automation (now part of Cadence) in the 1980s which became very popular with ASIC and IC designers.

VHDL (VHSIC Hardware Description Language)

A hardware description language developed in the 1980s by IBM, Texas Instruments, and Intermetrics under US government contract for the Department of Defense's VHSIC (Very High Speed Integrated Circuit) program. VHDL enjoys a growing popularity with ASIC designers as VHDL development tools mature.

Chapter 15 Troubleshooting

Errors while programming FPGA

There may be errors while programming FPGA, this may be due to many reasons, kindly check the following steps to recover the error.

- ? FPGA modules should be properly inserted.
- ? Check the jumper settings of FPGA module.
- ? Check the programming mode selection settings.
- ? See that the programming cable is properly inserted.
- ? The PLD & slot selector cards are in proper position.
- ? Ground the clock (GCK0) I/P during programming; use it after programming the FPGA.
- ? In case of Xilinx, open the iMPACT programmer with selected programming mode only.
- ? The parallel port of PC should be in ECP/EPP mode; else there would be connection errors.

Errors while programming 89c51

There may be errors while programming 89c51, kindly check the following steps to recover the error.

- ? Module should be properly inserted.
- ? The RS-232 programming cable should be properly inserted.
- ? Check the Flash Magic programmer settings (refer chapter configuration).

Errors while programming PIC controller

There may be errors while programming PIC16F877, kindly check the following steps to recover the error.

- ? Module should be properly inserted.
- ? The programming cable should be properly inserted.
- ? Check the ProgPIC programmer settings (refer chapter configuration).
- ? Check the jumper settings of PIC module.
- ? Plug in the +18V adaptor on the plug socket of module.

Controller modules not working properly

- ? After programming of 89c51 & PIC controllers, you have reset them from FPGA or from keys; else they won't function properly sometimes.
- ? Check the I/O connections properly.
- ? Check the reset logic.
- ? Check the source code thoroughly, there may be infinite loop or some other problem in code.

ADC/DAC Module not working properly

- ? Insert the module properly.
- ? Check the analog signal connections.
- ? Check the channel is properly selected.
- ? Check the ADC & DAC logic.

Disclaimer

ni logic pvt. Ltd., Pune takes the liability to replace the module/product for any design fault from our side.

ni logic pvt. Ltd., Pune does not take any responsibility of failures or damages caused to product due to incorrect design practices, misuse, improper handling and not following the datasheet specifications of devices.