Programmable Logic Devices

Overview

In this lesson we will

✓ Begin with an overview of programmable logic devices - PLDs.
✓ Motivate the use of PLDs.
✓ Introduce some of the basic underlying concepts in PLDs.
✓ Introduce some of the basic PLD configurations and architectures.
✓ Examine several of the core programmable/reprogrammable technologies
✓ Examine several different PLD designs.
✓ Examine several important issues in the design process.

Introduction

PLD - Programmable Logic Device

Hardware focus typically on

Microprocessor, microcontroller, and microcomputer

World of embedded systems is continually changing
World of electronic systems in general is changing
Today find an increasing number and variety of other components

Being designed into the applications to

Support or contain the basic computing core(s)

These devices are grouped as programmable logic devices (PLD)

With such devices

We begin to move into the world of

Large and very large scale integrated devices and world of distributed systems

Today these devices play a supporting role

Tomorrow

Will provide a highly flexible hardware environment including

Very high speed logic
Multiple microprocessor cores
Dynamically reconfigurable systems

Key strengths of these devices lie in

Small geometries
Ability to take advantage of the regularity in
Transistor level organization
Combinational logic and in certain kinds of memory elements

With such regularity the chip designs and layouts can be highly optimized
Included in this category of devices generally find

- (Programmable) read only memories ((P)ROM)
- Programmable logic arrays (PLA)
- Field programmable gate arrays (FPGA)
- Application specific integrated circuits (ASIC)
- Full custom designed integrated circuits - different from ASICs

- ASICs offer the designer the support of libraries of components that might include
  Logic gates, counters, arithmetic parts, and storage elements
  These pre-existing modules enable the designer
  To complete a new design very quickly
  In addition to the supporting libraries
  Component vendor will offer both synthesis and place and route tools
  To aid in the development process
  Such tools becoming increasingly intelligent

- Full custom integrated circuit is going to be faster and more dense
  Than either the ASIC or PLD
  Development cycle can be substantially longer

Discussion will provide introduction to programmable logic devices
Detailed design information
Better found in the data sheets and application notes
From the various device manufacturers

We will begin with

- Brief discussion motivating the use of programmable logic devices
  In contemporary systems
- Then examine the underlying logical concepts that have led to
  Development and wide spread use of programmable logic devices (PLD)
- We will next move to the basic building blocks of PLDs
  Show how these can be configured into useful tools
- Will then look at the commonly used technologies
  For implementing programmable devices
  How they are able to store information
• Will then work from basic pieces to present
  Basic structure of the devices
  Variations on I/O configurations
  Fundamental architectures for the CPLD and the FPGA
  Then compare and contrast these architectures
• Next we will introduce and study two of the more commonly used components
  CPLD and the Gate Array
  General purpose device called a Programmable System on a Chip

We will begin our study with a brief introduction to programmable devices.

Why Use Programmable Logic Devices?
Programmable logic device is a generic term
Covers all subfamilies of programmable logic
Also refers to a particular architecture
Discussion context should clarify
  When the term is being used in a general or a specific sense
We utilize such devices in our designs for a number of reasons
  Generally they are
    Faster
    Consume less power
    Can support significantly more functionality
    In a much smaller package than SSI and MSI logics

Simplify Development
  During development
    There certainly will be far less work involved in
      Layout of system printed circuit boards
    Design can be specified by a text file
      Rather than on multiple pages of schematics
        Some who may consider this to be a disadvantage

Encourage Reuse
  PLD based design also facilitates the reuse core elements
    Thereby helping to get a new design to the market more quickly
  Approach like standard libraries in C or C++
Design can become piece of intellectual property
Design can be created then sold to variety of companies
Modify / enhance to accommodate specific needs
Incorporate into larger design
ARM processor is excellent example

Reduce Complexity and Potentially Power Consumption
Today, a programmable logic device can easily replace
Thousands of SSI and MSI gates
Almost as many storage devices
Portable music player or cellular telephone
Implemented with an equivalent number of SSI devices
Would be enormous
Reduction in size of our products alone
Would be reason enough for using these devices

Increased and New Capabilities
Today, PLDs can be programmed to incorporate several CPU cores.
In addition to the computing power of the main system processor
Can bring several support processors to bear
On special portions of the application
We learned earlier that in a multitasking system
Divide a problem into multiple cooperating tasks
That appear to be running simultaneously
With multiple core PLDs,
Can begin to easily and economically achieve true multiprocessing
Different from multitasking
Can inexpensively off-load special purpose tasks to peripheral processor(s)

Hardware vs. Software
Learned that contemporary / embedded systems
Mixture of hardware pieces and software pieces

Developing field of co-design addresses the challenge
Mapping a functional design onto the hardware and software components
Which make up the architecture of the system
With certain PLDs distinction between hardware and software
Becoming increasingly fuzzy
Find that programmable logic devices can be dynamically reconfigured
To accommodate or adapt to the problem being solved
Such capabilities give us much greater flexibility in developing our designs

Note application software not the same as modeling software
Modeling software gives ability to do
'What if' type as well as 'Cut and try' type design
Want to avoid the latter

The Only Way
As our systems grow smaller in physical size
Growing correspondingly larger in capabilities and complexity
To implement those capabilities
We turn to more sophisticated designs
Putting an increasingly larger amount of circuitry into the same package
Several years ago drawing logic diagram for our system
Built a breadboard of a prototype system

Problems
Turning on prototype for first time fraught with problems
Problem compounded by the physical realities of the real world
Parasitic Effects
Parasitic effects on a 12 to 18 inch square breadboard
Substantially different from those found on a
Production printed circuit board of the same circuit
Integrated version
Problems associated parasitic effects
Become worse as the geometries get smaller and speeds increase

Increasing Frequencies
Problem is exacerbated as the operating frequency of today’s designs
Continues to increase
Hardware based breadboard
Cannot accurately reflect the true behavior of the design
In the final product
PLD manufacturers
Taken many of these issues into consideration
With the architecture and designs of today’s devices
Provide tools to aide in the analysis and solution of these problems

Contemporary Design Approach
Today much of our design can be
Implemented using programmable logic devices
Are usually developed using
  Verilog, VHDL, or some other hardware design language (HDL)

Designing using an HDL significantly simplifies the development process
  Compared with a hardware based breadboard approach

Development path entails - be careful here
  ✓ Significant modeling of the design
  ✓ Repeated simulation and test (including real world affects) of that model

By the time that the design is programmed into the PLD
  Probability of it working properly the first time
  Has been greatly increased

With a breadboard implementation of a circuit prototype
  Correcting any design errors can entail several days of rewiring

With the PLD and an HDL implementation of the design - be careful
  Same modifications can be completed in substantially less time
  Programming in a high level language beats rewiring any day

**Basic Concepts**

**The Programmable Logic Device**

Underlying strength of the PLD

At the very basic level - ability to
  ✓ Represent combinational logic in a sum of products form - minterms
  ✓ Build circuits as combinations of these (reduced) minterms,
  ✓ Store (and utilize) the outputs of the combinational nets

Typically configured as two level AND-OR devices
  AND array followed by OR array
  Logic variables available in true and negated form

Product terms are simply the logical AND of the variables
Sum terms are built as the logical OR of the AND expressions
More complex / sophisticated designs utilize *Look Up Tables* - LUT

Core components of architecture of such a system not difficult to visualize
Made up of
  • Uniform network of interconnections
• Set of variables to produce the desired logical expressions

Consider the simple basic layout

Observe

We have the asserted and negated forms of each of the input variables
Each AND gate has available for input
    True and negated form of each variable
Each OR gate has the output of each AND gate available as an input
In combination we have an AND array followed by an OR array

For the configuration

With two input variables
call them A and B
We have four possible minterms:
$$\overline{A} \overline{B}, \overline{A}B, \overline{AB}, AB$$

With the specified three outputs
Can express up to three different sum of product logic expressions - minterms
Based upon which connections we choose to make (or break)

Some implementations
All of the connections are present (in others, they’re absent)
To implement a logic equation
Simply remove (or make) some of the connections.

**Example**

Using the above device, we can implement the logical equation,

$$F = \overline{A} \overline{B} + \overline{A}B$$

First AND

Make the A and B connections
    Indicated by small circles

Second AND

Make the A and B connections
    Indicated by small circles

First OR

Make connections to the first two legs and no connection to third leg
We have
AND array and an OR array as the final implementation

Utilizing such an architecture
Can realize any sum of products expression
Only restriction is the size of the device
Number of input and output pins
Number of product terms available

Output Configurations
To enhance both usefulness and flexibility
Programmable logic devices designed with a variety of output configurations.
Typically include
Standard unidirectional combinational output
Configuration may be extended to support
- Bidirectional input and output
  Which may also be programmable
- Tristate control
Outputs may also be invertible
Through a programmable XOR path

Latched or registered outputs support basic storage
Permit the design and the implementation of sequential logics

Typical I/O
Input driver
Supporting the true and negated states of a signal

Tristate output driver
The state of that output is fed back into the array
Bipolar input signal - why
Latched or registered tristate output

The output of the storage element
Fed back into the array from the Q output
Using a bipolar driver

Such approach
Balances loading on outputs of storage element

Polarity of the input to the storage element
Controlled through the XOR device on the D input

Polarity of the output signal
Controlled by selecting the state of the input signal

In simple device architecture of the device may require that all storage devices
Use the same clock and the same reset or clear signal

In the simpler devices
Typically both the clock and the clear
Originate from outside the PLD device

More sophisticated devices may also support
Selection of internal or external clocking and reset sources
I/O pull-up resistors
Selectable enable
Ability to increase rise and fall times

Done to mitigate effects of large \( \frac{di}{dt}, \frac{dv}{dt} \)
More appropriately very small dt
To reduce switching noise

Basic Configurations
The core elements of programmable logics are found in four basic configurations
At one time these were stand alone devices
Today they are implementation methodologies in large devices
Analogous to SSI parts vs. VLSI

- (P)ROM - (Programmable) Read Only Memory
- PAL - Programmable Array Logic
- PLA - Programmable Logic Array
- PLS - Programmable Logic Sequencer
The (P)ROM

The (programmable) read only memory device (P)ROM

Most general and flexible
All combinations of the input bits are programmable.

To use a ROM or PROM to implement the following logical expression

\[ C\overline{D} + \overline{A}BC + A\overline{B}\overline{D} \]

(2) (11) (12)

Need a 16 x 1 bit memory
Four address lines will enable each memory bit to be uniquely addressed
For Example
Equation is implemented by storing
 Logical 1 in addresses 2, 11, and 12
 Logical 0 in all remaining.

The device will have a logical 1 as an output
Only when those addresses (or reduced minterms) appear on the device inputs
Each product term corresponds to 1 address

A (P)ROM based solution
Generally only appropriate when large number of product terms is required

Programmable Array Logic – PAL

In the PAL or Programmable Array Logic device
AND portion of the device is programmable
OR portion fixed
Such an architecture
Limits number of product terms in the sum
Product terms are not reusable
Must be duplicated if required
Implementation uses a fixed OR array
Output of each product term
Can be connected to only one OR gate
Typically the device supports
Bi-directional I/O pins
Tristate outputs with individual enables.
The structure of the device is shown in the accompanying diagram
Although drawn as only a single input to each device in the arrays
  Each supports multiple inputs
  Open circles illustrate the programmable links.

Programmable Logic Array – PLA
  In contrast to the PAL
    AND and OR portions of the device are programmable
    Consequently the product terms are reusable.

Programmable Logic Sequencer – PLS
  Programmable logic sequencer
    Also referred to as a registered device
  Simply a PLA plus flip-flop storage elements
  In most designs
    Outputs of the storage elements
      Fed back to the logic array
        Support for finite state machine design

For these devices
  Both the AND and OR arrays are programmable
  Product terms are reusable.
  Fragment of such a device is shown

PLA vs. PAL vs. (P)ROM
  PAL is the opposite of a PROM
  Can view the PROM as
    AND-OR array with a fixed AND array
      All \(2^n\) AND possibilities available

If a design needs a large number of AND combinations
  PROM is a good option
  Some example applications include
    Look up tables
    High speed mathematics
    Code translations
If the design requires only a few AND combinations
And not many of those are shared between outputs
PAL
With the fixed OR array is a good choice
If complete flexibility is necessary
PLA with its programmable AND and OR arrays
More appropriate.

**Programmable and Reprogrammable Technologies**
Programmable logic devices today fall into two major categories
Those that are programmable
Those that are reprogrammable
Those in the first category
Can be programmed one time
Those in the latter
Can be programmed multiple times.

**Programmable Technologies**
Devices that are one time programmable
Built to order by the manufacturer of the device or
Utilize a technology that will permit the designer
To enter the desired configuration one time
Devices built by a vendor mirror the technologies studied earlier
For implementing ROM type devices
Transistors are selectively included to produce the desired behavior
Devices that can be programmed once, in the field
Often based upon a technology called the *antifuse*.
In traditional electronics
*Fuse* utilized to protect against potentially damaging high currents
By opening a conducting path when necessary
*Antifuse* is designed to do the opposite
When a sufficiently high voltage is applied
Across an amorphous silicon link between two metal conductors
Metal-crystalline alloy is formed to complete the conducting path
The newly formed conducting path(s)
Enable the interconnection
Generic blocks of primitive logic components
To form more complex logic circuits
Once the low resistance path is formed, the process is irreversible
Reprogrammable Technologies

Today’s (re)programmable logic devices are generally SRAM based

Variations on programmable ROM technologies

SRAM Based Architectures

Built around volatile Read/Write memory cells
- That control the state of intra device connections

Device configuration and interconnection information
- Held in a companion nonvolatile storage medium
  - Such as a PROM or similar flash type of external boot device

One significant advantage of such an approach
- Configuration of the device can be dynamically changed at runtime
  - Thereby permitting the system to adapt to the problem at hand

Programmable ROM Based Architectures

Programmable ROM technologies fall into three broad categories EPROM, EEPROM, and FLASH.

All such devices built upon what are known as floating gate technologies
- FAMOS – Floating Gate Avalanche Injection MOS
  - Developed by D. Frohman-Bentchkowsky (1971-72)

To introduce the concept
- Begin with the memory fragment below
  - Gives a high-level view of a portion of a programmable AND array

Each column in the AND array
- Implements a NOR type AND
  - That is an AND of low true signals

Individual devices turn on with high on gate
- Implies input must be LOW else device turns on

Initially all transistors connected
- Remove transistor for logical 1
  - Include for logical 0
All transistors on a column must be OFF
For the output to be logical one

Each N type transistor comprising the AND has two gates
One of the gates is floating
Unconnected
Surrounded by high impedance insulating material

Initial / Unprogrammed State
Initially the floating gate has no charge on it
Thus has no effect on circuit operation
All transistors are effectively connected
Can be turned OFF or ON

When a positive voltage is applied
To the non-floating gate
Transistor turns ON
Output of the associated AND is logical 0

Programmed State
To program the device
High voltage applied to each location where link is not wanted
Program 1 to remove transistor 0 to include
Shown in the first diagram

Through avalanche injection
Negative charge collects on the floating gate

When the programming voltage is removed
A number of electrons are trapped on the floating gate
Because of surrounding insulating material
Shown in the second graphic
Trapped electrons cause an increase the threshold voltage
To approximately 7.0 V
Which thereby prevents the transistor from turning ON
When a normal logical 1 is applied to the non-floating gate
Illustrated in the final drawing

Transistor is effectively disconnected from the circuit
Tests have shown that the charge can be retained for up to 10 years

Erasing the Device
In the early years
Ultraviolet light was used to erase the device

Under the ultraviolet light
Floating gate becomes slightly conductive
Charges trapped during programming
Given enough energy to leak away

Such devices are known as EPROMs – erasable PROMs.
Today’s devices are significantly improved from the early designs
Floating gate is surrounded by an ultra thin insulating layer

Rather than using an avalanche injection scheme
Charge is placed onto the floating gate using tunneling physics
Devices are also electrically erasable
Device can be erased
Apply a voltage of opposite polarity to the charging voltage
On the non-floating gate
Thus, we can use the same equipment as was used to program device

Cross sectional diagram of two contemporary types of programmable device
Given below
Diagram on the left
Called electrically erasable device EEPROM
Uses Floating Gate Tunneling Oxide technology
Uses Erase Through the Oxide technology
Diagram on the right
Called FLASH technology
Uses Floating Gate Tunneling Oxide technology

**Architectures**
Independent of whether the programmable logic device appears as PLD, CPLD, or FPGA

Architecture of the device comprises 3 key components
- Blocks of combinational logic and/or storage elements
- Input/Output logic
- Interconnections between these blocks of functionality

Power and capabilities of the device determined by
- Size (or grain) and flexibility of the logic blocks
- Speed and generality of the interconnection net – key point

PLD and CPLD devices
- Tend to be coarser grained
- Building blocks are generally referred to as *macrocells*
  
  Can be viewed as the programmable logic equivalent of MSI logic blocks

FPGA devices
- Usually finer grained
- With a very rich interconnection topology
  
  Building blocks tend to be the SSI logic equivalent
  
  Often built as *look-up tables* (referred to as LUTs)

Today devices range in capability
From
- Simple encoding or decoding networks
To
- Ability to implement a full CPU core
  
  With accompanying peripheral devices
PLDs
Simplest devices known as PLDs
Programmable logic devices
Distinguished from the general use of the term
To describe the complete class of programmable devices
The basic devices intended to replace collections of combinational logic circuitry
Implemented using SSI and MSI components

Devices are implemented
As the AND array – OR array pair discussed earlier

Lattice Semiconductor
Provided an early implementation of such a device
Called a generic array logic or GAL16V8™ device.

GAL™ supports
16 inputs and 8 outputs
Configured as
- 10 input only
- 2 output only
- 6 that are bi-directional
Each output can accept up to 7 product terms
Each product term can have up to 32 inputs
Circuit fragment below illustrates
Input and the two different output configurations
Programmable exclusive OR gate
Provides the ability to invert the sense of the output signal
With the addition of storage devices
Capability of the GAL16V8™ was extended as the GAL22V10™
Supports the design and implementation of basic sequential digital circuits

The device incorporates 10 D type flip-flops into a structure called a **macrocell**
With common clock and asynchronous reset
Individual synchronous preset.

Each macrocell could be configured as registered or non-registered

Output product term support is organized in pairs
Two support 8 product terms
Two support 10 etc.
Up to a maximum of 16

The circuit fragment below illustrates typical
Unregistered and registered output macrocells.

The macrocell building block concept
Forms the basis upon which the larger and more sophisticated CPLDs are built
The AND-OR array
Provides a fully interconnected mesh within the macrocell
Full inter connectivity means that any macrocell can communicate
Directly with any other macrocell

![Diagram](image)

Earlier simple implementations of entire device
Now form building blocks of more complex
Implementing the Design

Logic designs are developed
Textually
Using any of a number of hardware design languages (HDL)
Verilog, VHDL, or ABEL
Through schematic capture

Once in electronic form
Design can be synthesized
Using a tool that is similar to, yet different from, a software compiler
Into a form that is compatible with the tool used to program the device

CPLD
The Complex Programmable Logic Device CPLD
Provides significantly greater capability than the basic PLD
Functionality of these devices ranges
From
What is found in the more powerful MSI devices
To
That in the low ends of the LSI units and moving to VLSI

General Architecture
General architecture comprises
• Collection of logic blocks
• Interconnection net
• Input / Output

Each logic block
Utilizes building blocks and techniques discussed earlier
Contains
An AND / OR array
Number of macrocells
Diagram below
Illustrates a device with six logic blocks
Each contains eight macrocells.

Interconnection Net
Interconnection net
Provides means to route information from one logic block to another
Will often contain
Global system clock
Signals supporting global set and reset controls
For any sequential logic or memory elements

Signal routing between logic blocks within the interconnection net
Supported on several different levels
In some cases
Full inter connectivity is not supported
Some logic blocks may not be able to exchange information
directly or indirectly
With every other logic block
In other cases
Inter connectivity is qualified
Qualification includes
Direct connection to a limited subset of the logic blocks or speed
Connection to another block via routing
Through one or two other blocks along the path
Such routing restrictions may preclude
Fully utilizing all logic blocks or macrocells available in device.
Logic Blocks
Each logic block will contain a number of macrocells
Depending upon the vendor and the design of the part
   Within each logic block there will be a local interconnection mesh
   When the source and destination of a transfer are within the same block
   Rate of exchange can be significantly faster than between logic blocks

Differences between compiler software and synthesis hardware tools
   Significantly important
   Differences are decreasing

Each macrocell will contain
   • Rich and flexible set of logic functionality
     Supporting both combinational and storage capability
   • Storage generally implemented as D or T type flip-flop
     Can be selectively bypassed
   • Input to the flip-flop can be selected from a number of different sources
     Internal to the macrocell or from without
   • Flip-flop clock may be
     Global clock or
     Built from product terms in the logic array

Macrocell Inputs and Outputs
   Can include variety of selectable capabilities
   Output signal may originate from
   ✓ The storage element
   ✓ Combinational term from the AND/OR array

In what may seem unusual in today’s world of increasing logic speeds
   Some devices support the capability
   To slow signal rise and fall times
   See earlier discussion
   A little reflection explains the oddity
   Significant source of noise arises from
   Rapidly changing voltages or currents
   Coupled with the pandemic parasitic inductors and capacitors

   • Tristate capability on I/O lines
     Enables external information to be brought into
     Logic equations implemented in the macrocells
Because the device may be or typically are interfacing with a bus
Each I/O line may also include a weak pull up
That can easily be overdriven by an input signal
To define state of bus in undriven or tristate mode

FPGAs
Field Programmable Logic Array, FPGA,
Provides capabilities on par with the CPLD
As the technologies continue to evolve
Distinctions between the two devices becoming increasingly blurred
We noted CPLD implements a coarse grained architecture
• Built around logic blocks and macrocells
FPGA utilizes a finer grained structure
• Based upon
  Smaller configurable logic blocks
  Rich interconnection scheme

Each logic block in an FPGA is far simpler that in the CPLD
Typically comprised
• Some basic combinational logic implemented via a look-up table - LUT
• A storage device
• Large amount of internal I/O

Large number of flip-flops and rich interconnect capability
Make device much more flexible for embedded systems than CPLD

General Architecture
FPGA is architected either as
• SRAM device to hold the configuration or
• Electrically programmable antifuses.

SRAM Device
SRAM based design gives tremendous flexibility
At power ON configuration is downloaded to the device
As a serial bit stream from an external PROM
Because it is a soft load
Configuration of the device can be dynamically changed at runtime
In response to the type of problem being solved

Device organized as array of interconnected configurable logic blocks
Devices on the periphery of the array
Connect to I/O blocks
Interconnection is supported by a series of channels
Over which signals are routed using switch matrices
Portion of a typical switch matrix is implemented as shown

Representative architecture given as

In the diagram
Signal on any row can be routed to any column
By closing the switch at the desired intersection
Signal on any column can be routed to any row
In a similar way
Often called a *crosspoint switch*

**Antifuse Device**
Configuration of an FPGA utilizing an antifuse based architecture
Irreversibly set when the user of the device programs it.

Figure shows portion of antifuse based architecture
Device comprises
Columns or rows of configurable logic blocks
Separated by an interconnection channel
Connections are made
From
A logic block signal
To
A path in the channel
By programming the antifuse at the desired connection point
To the low resistance state.
The very small size of the antifuses
Permits the device to support a substantial number of interconnections.

Example PLD Devices
Let’s now look at several commercially available programmable logic devices
That are representative examples of the concepts we’ve been discussing.
While not pushing the limits of the state of the art in programmable logic devices
They are very good choices for many mid level embedded applications

We’ll examine – while somewhat dated concepts still valid
Xilinx XC9500XL™ CPLD
Altera Flex 10K™ FPGA
Cypress PSOC™ (Programmable System on a Chip) mixed signal array

Xilinx XC 9500XL™ Family CPLD
The Xilinx XC9500XL™ family comprises four devices
Increasing complexity and capability
Considered to be medium density
Devices support
800 to 6400 usable gates
36 to 288 registers
Architecture is based upon
Fully interconnected Function and Input / Output Blocks
Interconnection is accomplished via a switch matrix
Observe primary key pieces
• Input
• Output
• Function blocks
• Interconnect
High level functional block diagram for the device is given as

Function Blocks

Each Function Block has

- 54 true and complemented inputs
- Up to 18 outputs
- Comprises up to 18 independent macrocells and a logic array

  The logic array can be programmed to produce up to 90 product terms
  Each of the 90 can be allocated to each macrocell
  Minterm type expressions

- Receives
  Global clocks
  Set and reset signals
  Either two or four global output enable signals

- Outputs and their associated output enable signals
  Sent to the I/O Block (shown above as two separate blocks)

- The 18 data output signals
  Also sent to the interconnect switch for routing to other Function Blocks

Macrocells

Each macrocell can be individually configured to implement either
  Combinational or registered logic functions

Configuration of the macrocell is controlled by

  Five local product terms derived from
  AND array
  Product terms from other macrocells.
Register function implements either a D or T type
With asynchronous set and reset operations.

The five local product terms provide the following local controls:
  - Flip-Flop Set
  - Flip-Flop Reset
  - Flip-Flop Clock
  - Flip-Flop Data Polarity
  - Output Enable

Represents typical macrocell

Switch Matrix

*Switch Matrix* is used to connect signals to the *Function Block* inputs
As noted all *Function Block* outputs also drive the switch matrix

I/O Block

*I/O Block* provides the means by which
Internal logic connects to the device I/O pins
As figure below illustrates block comprises
  - Set of input and output buffers
  - The output enable selection.

Output enable control can originate from any one of four different sources:
- Product term from the macrocell
- Any of the global OE signals
- Always enabled
- Always disabled
The I/O subsystem also supports
Programmable slew rate control for noise management
Programmable ground pins

Additional ground pins also contribute to noise management
By lowering the resistance of the ground path for the chip

![Diagram of I/O subsystem]

Altera Flex 10K™ FPGA
The Altera Flex 10K™ is
Logic array plus
Embedded memory
Based upon a high density CMOS SRAM process

Device family supports up to
Maximum of over 300,000 system gates
41K bits of RAM
Read and Write cycle times of up to 84 MHz and 63 MHz respectively

As noted earlier
Problems of clock management are becoming
Increasingly
Significant
Challenging
With the increasing
Internal and external data rates
Ever decreasing device geometries (and associated parasitic effects)
Of particular importance in the Flex 10K
Ability to manage clock delay and / or skew
  • Within and across the device and
  • Ability to multiply an input frequency up
    To provide finer grained temporal resolution within the device
Important feature

The Flex 10K supports both of these capabilities
Through built in
  • Low skew clock distribution trees and
  • Phase lock technologies
Used to
  Manage clock synchronization across the device
    Important
    Provide device wide management of signals in time

Family supports slew rate control on output signals
  Help to control system noise
    Similar to earlier devices that we have discussed,

The I/O capability of the family (at close to 500 I/O pins)
  On par with many contemporary high performance microprocessors
The high-level architecture for the device is given in diagram below
Architecture
Each device contains two major blocks or subsystems
*Embedded array* used to build memory and specialized logic functions
*Logic array* to implement more general logic

Embedded Array
Embedded array comprises of series of *embedded array blocks* (EABs)
Shown in figure below

When the EABs are used to implement *memory*
Each provides up to 2 K bits that can be utilized to create
ROM, RAM, dual-ported RAM, or FIFO functions

When used to create *logic functions*
Each EAB can implement from 100 to 600 gates
That can be utilized to construct such expected logic functions as
Arithmetic circuits
Micro controllers
Sequential circuits
Digital signal processing functions
EABs can be used
  Independently or in concert to implement more complex logical functions

Embedded array block is implemented as a block of RAM
  Has registered input and output ports
    Buffered I/O via flip-flops
  In the registered configuration
    Device can easily implement the underlying state machines
      Used to implement error management
      In data telecommunication schemes
    Computation and control blocks
      Necessary for performing FFT computations

To implement logic functions
  Each EAB is programmed with a read only pattern during configuration
    Creates a large look up table or LUT
  As we discussed in an earlier
    Combinatorial functions can easily be implemented by
      Simply reading the memory at the appropriate address
        Where the results have been stored

Major advantage of such an approach
  Design much faster than propagating signals
    Through a corresponding logic block

LUT approach enables complex logic functions to be implemented
  With only a single level of delay
    Eliminates routing delays inherent in AND / OR array approach
  Using the look-up table approach
    Can easily implement a high speed 4 x 4 multiply block

The EABs can be utilized as a synchronous RAM (SDRAM)
  Synchronous RAM block in an EAB is self-timed
    Based upon a global clock
    Write signals are thence derived from that clock
  Multiple EABs can be combined to produce larger RAM blocks
  All the EABs can be cascaded to form single RAM block
Logic Array

Logic array is made up of a series of logic array blocks (LABs) illustrated in the diagram.

Each LAB contains
- Eight logic elements (LE)
- A local interconnect

Each logic element contains
- Four input look up table (LUT)
- Programmable flip-flop
- Specialized signal paths that support carry and cascade functionality

Logic elements can be used to create MSI scale logic blocks.
Examples might include
- Smaller counters, decoding logic, or simple finite state machines

Multiple LABs can be combined
To create larger and more complex logical functions
Each LAB
- Is the equivalent of approximately 96 usable logic gates
- Provides four local control signals for configuring LAB functionality
- Supports programmable inversion
- Two of the control signals can be used as clocks
- Two can be used as clear / preset control for the local flip-flop
- Clock source can originate either from
  - Dedicated clock input pin or
  - Global clock.

Logic Element
- The logic element is the smallest unit of logic in the device
- Each contains a four input look-up table
  - Thus can compute any function of four variables
- The local flip-flop can be configured as
  - D, T, JK, or SR type device
  - Each supports clock, clear, and preset
  - Clock source can originate either from
    - Dedicated clock input pin
    - Global clock
  - Preset / Clear can come from
    - Global signal
    - I/O signal
    - Locally generated signal.

To support high-speed logical operations
- Involving multiple adjacent logic elements
Device incorporates
- Two dedicated data paths
- Carry chain
- Cascade chain
- These bypass the local interconnect net
  - With its attendant delays
Such capability is important

Each logic element provides two independently controlled outputs
- One drives the local interconnect net
- Other drives either the row or column FastTrack Interconnect
  - See diagram – important feature
Input / Output and Internal Interconnection
Interconnection to and from the input and output device pins
Achieved via a series of row and column channels
Called FastTrack Interconnect
These run the length and width of the device

At the end of each row and column is an I/O element – IOE
Portion of the IOE is illustrated below
Observe that the I/O element contains
Bi-directional buffer and flip-flop
These can be used to provide
General purpose input output
A bi-directional register

When configured into the registered mode and utilizing a dedicated clock
Input setup time, $\tau_{su}$, can be as low as 3.7 ns
$\tau_{hold}$, the hold time is 0 ns
In the registered output mode
Propagation delays $\tau_{HL}$ or $\tau_{LH}$
From the clock are specified as 5.3 ns
The *FastTrack Interconnect* architecture

Provides connections between logic elements (LE) and device I/O pins

Signal path is implemented as

Series of horizontal and vertical routing channels

Each row is served by dedicated row interconnect

That can drive I/O pins and feed other LABs

Each column interconnect

Routes signals between rows

Can drive the I/O pins

See earlier graphic

The I/O Element (IOE)

Implements a bi-directional I/O buffer and register

Each IOE selects

Clock, clear, clock enable, and output enable controls

From an internal network called a *peripheral control bus*

Bus uses high speed drivers

To minimize signal skew across devices

Peripheral control signals can be allocated as

- Output enable signals
- Clock enable signals
- Clock signals
- Clear signals

**Cypress CY8C21x23™ PSOC**

In today’s embedded systems comprise

Core microprocessor

Mix of analog and digital peripheral devices

Such as A/D or D/A converters

Timers,

Communications channels

The CPLD and FPGA devices are strictly digital

They have grown to become essential components

Their role is becoming increasingly important

Today, they are still ancillary to the core design

Cypress Semiconductor is taking step towards more general purpose solution

With their Programmable System on a Chip (PSOC) family
PSOC is built around a CPU core
    Adds a
    A digital
    An analog
    Resources subsystem

The analog and digital blocks, I/O, and the system bus
    All configurable

The CPU Core
The CPU core is build around an 8 bit Harvard architecture
    Separate instruction and data memory
    CPU performs at the four MIPS level
    Million instructions per second
    Driven from a clock source of up to 24 MHz

Supporting the CPU core are
    • Flash ROM for program store
    • An SRAM for data store

Completing the core is
    • An interrupt controller
    • A sleep and watchdog timer
    • Clock source module

The components within the CPU core
    Connected to each other and to the system resources via a system bus

The Digital Subsystem
The Digital Subsystem is built around an array of four digital blocks
Each block is an eight bit configurable resource
    Can be combined with other blocks into more complex peripheral devices

Typical examples include:
    Timing Peripherals
        Counters
        Timers
    PWMs
        Used for motor control among other things
    Telecommunications Devices
        UART,
        SPI Master and Slave
    Shift Register Based Devices
        CRC Generator / Checker
        PN Sequence Generation
Components within the *Digital Subsystem*
Connected to each other and to any of the I/O ports
Through a *Digital Interconnect Bus*

The Analog Subsystem

*Analog Subsystem* is built around four analog blocks
Blocks can be configured into a variety of different devices such as,
  Analog to Digital Conversion
    Single – 8 bit Conversion
    Dual – 8 bit Conversion
  Comparison Devices
    Single-ended Comparison with absolute Reference
    Single-ended Comparison with DAC Reference

Using a configuration scheme similar to that in the *Digital Subsystem*
Components are connected to each other and any of the I/O ports
Through an *Analog Interconnect Bus*.

The System Resources
In addition to the capabilities provided by the Analog and Digital Subsystems
The device provides further capabilities grouped as *System Resources*
These are accessed via the system bus

Among the capabilities supported by the block are
  Digital
    Clock Dividers supporting three different frequencies
    I^2^C Master, Multi-master, and Slave
  Analog
    Low Voltage Detection Interrupts and Power on Reset support
    Fixed Voltage Reference
    Switch Mode Pump

The Design Process
The design process for PLD based systems
Differs little from what we have already discussed
The development cycle is governed by the same cautions and precautions

As we’ve learned earlier
The embedded system development cycle begins with
Identifying the requirements

For PLD based designs, we are, more often than not, the customer
Thus, we know the requirements
These requirements still need to be documented
From the requirements
  Write a formal design specification
    Quantifying all of the signals and all of the constraints

  Our requirements and design specifications for the PLD
    Naturally derive from those of the larger enclosing system

A PLD based design will be developed
  Using one of the many hardware design languages
    Verilog or VHDL

Incorporating a programmable device into a system
  Does not eliminate the need for a detailed timing analysis
    Portions of the larger system that will be implemented in the PLD

Each device and device type is going to be different
  Thus it’s important
    Work with the manufacturer’s data sheets
    Thoroughly understand the behavior of the device
    Any critical timing paths

Some basic, yet important, timing specifications to be aware of include
  - $\tau_{PD}$ - Propagation delay from input to output
  - $\tau_{CO}$ - Propagation delay from clock (edge) to output
  - $\tau_{CF}$ - Propagation delay from clock (edge) to internal flip-flop feedback
  - $\tau_{SU}$ - Setup time from inputs to clock(edge)
  - $\tau_{H}$ - Hold time from inputs to clock(edge)

Clock management

Signal routing and management

Partitioning of the logic into appropriate blocks
  To maximize performance (including inter and intra block communication)
  Minimize power

Are essential parts of the design, modeling, and simulation of the design

Reducing module coupling and increasing cohesion
  Definitely applies in the PLD context
  Any device will ultimately be exchanging information
    With other external companion devices
Properly assigning input and output pins

Associating these assignments with
     Corresponding logic functions and
     Their internal placement
Can have a significant effect on device performance

While focused on the design and development of a PLD based subsystem,
     Remember that at the end of the day it is a part of a larger system
     It must interact with and is subject to the same constraints
     As the containing system

Working with the vendor’s tools
     Design is ultimately synthesized into an electronic interchange format
     Used to either
     Program the CPLD
     Build an for an FPGA
          SRAM set
          Antifuse pattern

Summary

In this lesson we
     ✓ Began with an overview of programmable logic devices - PLDs.
     ✓ Motivated the use of PLDs.
     ✓ Introduced some of the basic underlying concepts in PLDs.
     ✓ Introduced some of the basic PLD configurations and architectures.
     ✓ Examined several of the core programmable/reprogrammable technologies
     ✓ Examined several different PLD designs.
     ✓ Examined several important issues in the design process.