Field Programmable Gate Arrays (FPGAs)

Cristinel Ababei
Dept. of Electrical and Computer Engr.
Marquette University

Overview

- FPGA Devices
  - ASIC vs. FPGA
  - FPGA architecture
- FPGA Design Flow
  - Synthesis
  - Place
  - Route
Traditional CMOS Circuits
(think of application specific integrated circuits, ASICs)

Once fabricated cannot be changed!

• Does not implement a specific circuit functionality!
• Can be (re)programmed or configured to implement any desired circuit!

Field Programmable Gate Array (FPGA)

Once fabricated:
• Does not implement a specific circuit functionality!
• Can be (re)programmed or configured to implement any desired circuit!
### ASIC vs. FPGA

<table>
<thead>
<tr>
<th>ASIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Specific Integrated Circuit</strong></td>
<td><strong>Field Programmable Gate Array</strong></td>
</tr>
<tr>
<td>• designed all the way from behavioral description to <strong>physical layout</strong></td>
<td>• no physical layout design; design ends with a <strong>bitstream</strong> used to configure a device</td>
</tr>
<tr>
<td>• designs must be sent for expensive and time consuming <strong>fabrication</strong> in semiconductor foundry</td>
<td>• bought <strong>off the shelf</strong> and reconfigured by designers themselves</td>
</tr>
</tbody>
</table>

---

### Which way to go?

<table>
<thead>
<tr>
<th>ASICs</th>
<th>FPGAs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High performance</strong></td>
<td><strong>Off-the-shelf</strong></td>
</tr>
<tr>
<td><strong>Low power</strong></td>
<td><strong>Low development cost</strong></td>
</tr>
<tr>
<td><strong>Low cost in high volumes</strong></td>
<td><strong>Short time to market</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Reconfigurability</strong></td>
</tr>
</tbody>
</table>
**Why FPGAs?**

- Custom ICs are very expensive to develop, and delay introduction of product to market (time to market) because of increased design time.
- Note: need to worry about two kinds of costs:
  - 1. cost of development, called non-recurring engineering (NRE)
  - 2. cost of manufacture
- A tradeoff usually exists between NRE cost and manufacturing costs

---

**Applications of FPGAs**

- Implementation of random logic
  - easier changes at system-level (one device is modified)
  - can eliminate need for full-custom chips
- Prototyping
  - ensemble of gate arrays used to emulate a circuit to be manufactured
  - get more/better/faster debugging done than possible with simulation
- Reconfigurable hardware
  - one hardware block used to implement more than one function
  - functions must be mutually-exclusive in time
  - can greatly reduce cost while enhancing flexibility
- Special-purpose computation engines
  - hardware dedicated to solving one problem (or class of problems)
  - accelerators attached to general-purpose computers
Applications of FPGAs

• Early on, used to serve as “glue logic” and for prototyping. Now? Everywhere!
  – Communications, software-defined radio, digital signal processing, ASIC prototyping, computer hardware emulation, medical imaging, computer vision, automotive, speech recognition, cryptography, bioinformatics, financial, bitcoin, ...
  – HW accelerators in datacenter servers (Intel purchased Altera for $16 billion).

Major FPGA Vendors

SRAM-based FPGAs
  ▪ Xilinx Inc.
  ▪ Altera Corp. ($16B Intel 2015)
  ▪ Atmel ($3.6B Microchip 2016)
  ▪ Lattice Semiconductor

Flash & antifuse FPGAs
  ▪ Actel Corp.
  ▪ Quick Logic Corp.

Share about 90% of the market
Xilinx FPGA Families

- Old families
  - XC3000, XC4000, XC5200
  - Old 0.5µm, 0.35µm and 0.25µm technology. Not recommended for modern designs.

- High-performance families
  - Virtex (220 nm)
  - Virtex-E, Virtex-EM (180 nm)
  - Virtex-II, Virtex-II PRO (130 nm)
  - Virtex-4 (90 nm)
  - Virtex-5 (65 nm)
  - Virtex-6

- Low Cost Family
  - Spartan/XL – derived from XC4000
  - Spartan-II – derived from Virtex
  - Spartan-IIE – derived from Virtex-E
  - Spartan-3 (90 nm)
  - Spartan-3E (90 nm) – logic optimized
  - Spartan-3A (90 nm) – I/O optimized
  - Spartan-3AN (90 nm) – non-volatile
  - Spartan-3A DSP (90 nm) – DSP optimized
  - Spartan-6

Zynq-7000

- Based on the Xilinx All programmable SoC architecture; 28nm technology node
- ARM dual-core Cortex-A9 MPCore processors
- Fixed processing system that can operate independently from the programmable logic
- Processor boots on reset like any processor-based device or ASSP
- Processor acts as “system master” and controls the configuration of the programmable logic enabling full or partial reconfiguration of the programmable logic during operation
- Standard development flows providing a familiar programming environment for software developers
- Additional documentation and resources:
# Zynq-7000 Device Family

<table>
<thead>
<tr>
<th>Processor Core</th>
<th>Z-7010</th>
<th>Z-7015</th>
<th>Z-7020</th>
<th>Z-7030</th>
<th>Z-7045</th>
<th>Z-7100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Core</td>
<td>Dual ARM® Cortex™-A9 MPCore™ with CoreSight™</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor Extensions</td>
<td>NEON™ &amp; Single / Double Precision Floating Point for each processor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1 Cache</td>
<td>512 KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Cache</td>
<td>256 KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Interfaces</td>
<td>DDR3, DDR3L, DDR2, LPDDR2, 2x Quad-SPI, NAND, NOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peripherals</td>
<td>2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic Cells</td>
<td>28K Logic Cells</td>
<td>74K Logic Cells</td>
<td>85K Logic Cells</td>
<td>125K Logic Cells</td>
<td>350K Logic Cells</td>
<td>444K Logic Cells</td>
</tr>
<tr>
<td>BlockRAM (Mb)</td>
<td>240 KB</td>
<td>380 KB</td>
<td>560 KB</td>
<td>1,060 KB</td>
<td>2,180 KB</td>
<td>3,020 KB</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>80</td>
<td>160</td>
<td>220</td>
<td>400</td>
<td>900</td>
<td>2,020</td>
</tr>
<tr>
<td>Transceiver Count</td>
<td>4 (6.25 Gb/s)</td>
<td>up to 8 (12.5 Gb/s)</td>
<td>up to 16 (12.5 Gb/s)</td>
<td>up to 16 (10.3125 Gb/s)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Zynq-7000 Diagram

[Diagram showing Zynq-7000 processing system]
ZebBoard

Intel Altera FPGA Families

- High & Medium Density FPGAs
  - Stratix™ II, Stratix, APEX™ II, APEX 20K, & FLEX® 10K
- Low-Cost FPGAs
  - Cyclone™ & ACEX® 1K
- FPGAs with Clock Data Recovery
  - Stratix GX & Mercury™
- CPLDs
  - MAX® 7000 & MAX 3000
- Embedded Processor Solutions
  - Nios™, Excalibur™
- Configuration Devices
  - EPC
Altera: Cyclone V

- Extends the Cyclone FPGA series
- Wide spectrum of general logic applications
- Up to 300,000 logic elements (LEs)
- Additional documentation and resources:

Cyclone V Key Architectural Features
Cyclone V Devices

Cyclone V Device Variants and Packages

Table 3: Device Variants for the Cyclone V Device Family

<table>
<thead>
<tr>
<th>Variant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclone V E</td>
<td>Optimized for the lowest system cost and power requirement for a wide spectrum of general logic and DSP applications</td>
</tr>
<tr>
<td>Cyclone V GX</td>
<td>Optimized for the lowest cost and power requirement for 614 Mbps to 3.125 Gbps transceiver applications</td>
</tr>
<tr>
<td>Cyclone V GT</td>
<td>The FPGA industry's lowest cost and lowest power requirement for 6.144 Gbps transceiver applications</td>
</tr>
<tr>
<td>Cyclone V SE</td>
<td>SoC with integrated ARM-based HPS</td>
</tr>
<tr>
<td>Cyclone V SX</td>
<td>SoC with integrated ARM-based HPS and 3.125 Gbps transceivers</td>
</tr>
<tr>
<td>Cyclone V ST</td>
<td>SoC with integrated ARM-based HPS and 6.144 Gbps transceivers</td>
</tr>
</tbody>
</table>

Table 10: Maximum Resource Counts for Cyclone V SE Devices

<table>
<thead>
<tr>
<th>Resource</th>
<th>A2</th>
<th>A4</th>
<th>A5</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Elements (LE) (K)</td>
<td>25</td>
<td>40</td>
<td>85</td>
<td>110</td>
</tr>
<tr>
<td>ALM</td>
<td>9,434</td>
<td>15,694</td>
<td>32,075</td>
<td>41,509</td>
</tr>
<tr>
<td>Register</td>
<td>37,736</td>
<td>60,376</td>
<td>128,300</td>
<td>166,036</td>
</tr>
<tr>
<td>Memory (KB)</td>
<td>M10K</td>
<td>2,000</td>
<td>3,970</td>
<td>5,570</td>
</tr>
<tr>
<td></td>
<td>MLAB</td>
<td>138</td>
<td>231</td>
<td>480</td>
</tr>
<tr>
<td>Variable-precision DSP Block</td>
<td>36</td>
<td>84</td>
<td>87</td>
<td>112</td>
</tr>
<tr>
<td>18 x 18 Multiplier</td>
<td>72</td>
<td>168</td>
<td>174</td>
<td>224</td>
</tr>
<tr>
<td>FPGA PLL</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>HPS PLL</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>FPGA GPIO</td>
<td>145</td>
<td>145</td>
<td>288</td>
<td>288</td>
</tr>
<tr>
<td>HPS I/O</td>
<td>181</td>
<td>181</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td>LVDS</td>
<td>32</td>
<td>32</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>37</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>FPGA Hard Memory Controller</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HPS Hard Memory Controller</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ARM Cortex-A9 MPCore Processor</td>
<td>Single-or dual-core</td>
<td>Single-or dual-core</td>
<td>Single-or dual-core</td>
<td>Single-or dual-core</td>
</tr>
</tbody>
</table>
Logic Element (LE)

- The smallest unit of logic located in a LAB of all Altera devices supported by the Quartus software.
- Logic element (LE) is also generally known as a logic cell.
- In supported device (Arria series, Cyclone series, and Stratix series) family devices, a logic element consists of:
  - a four-input LUT
  - a programmable register
  - a carry chain

Adaptive Logic Module (ALM)

- Basic building block of supported device (Arria series, Cyclone V, Stratix IV, and Stratix V) families
- Contains among others:
  - two or four register logic cells
  - two combinational logic cells
  - two dedicated full adders
  - a carry chain
  - a register chain
- [http://quartushelp.altera.com/15.0/master.htm#mergedProjects/quartus/gl_quartus_welcome.htm](http://quartushelp.altera.com/15.0/master.htm#mergedProjects/quartus/gl_quartus_welcome.htm)
DE1-SoC Board

- $175 USD (academic)
- FPGA Device
  - Cyclone V SoC 5CSEMA5F31C6 Device
  - Dual-core ARM Cortex-A9 (HPS)
  - 85K Programmable Logic Elements
  - 4,450 Kbits embedded memory
  - 6 Fractional PLLs
  - 2 Hard Memory Controllers
- Built-in USB Blaster for FPGA programming

Overview

- FPGA Devices
  - ASIC vs. FPGA
  - FPGA architecture
- FPGA Design Flow
  - Synthesis
  - Place
  - Route
FPGA Architecture – General

FPGA Architecture – Detail
1) Configurable Logic Block (CLB)

- **4-input look-up table (LUT)**
  - Implements combinational logic functions (essentially store truth table of the function)
  - How do we implement LUT’s?

- **Register**
  - Optionally stores output of LUT

> Think of LUT as of memory that stores truth table of any Boolean function of 4 inputs!
> The four inputs represent the "address" from where to read from this memory!

---

**How could you build a generic Boolean logic circuit? Memories as LUTs**

- 1-bit memory to hold boolean value
- Address is vector of boolean input values
- Contents encode a boolean function
- Read out logical value (col) for associated row
LUT as general logic gate

- An n-LUT as a direct implementation of a function truth-table.
- Each latch location holds the value of the function corresponding to one input combination.

Example: 2-LUT

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Can be used to implement any function of 2 inputs.
How many of these are there?
How many functions of n inputs?

LUT as general logic gate

- Look-Up Tables are primary elements for logic implementation
- Each LUT can implement any function of 4 inputs
5-Input functions implemented using two LUTs

Recall: Multiplexer/Demultiplexer

- **Multiplexer**: route one of many inputs to a single output
- **Demultiplexer**: route single input to one of many outputs
Multiplexers/Selectors: to implement logic

- 2:1 mux: \( Z = A' I_0 + A I_1 \)
- 4:1 mux: \( Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3 \)
- 8:1 mux: \( Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7 \)

Multiplexers as LUTs

- \(2^n:1\) multiplexer implements any function of \(n\) variables
  - With the variables used as control inputs and
  - Data inputs tied to 0 or 1
  - In essence, a look-up table

Example:
- \( F(A,B,C) = m_0 + m_2 + m_6 + m_7 \)
  - \( = A'B'C' + A'BC' + ABC' + ABC \)
  - \( = A'B'(C') + A'B(C') + AB'(0) + AB(1) \)
Cascading Multiplexers

- Large multiplexers implemented by cascading smaller ones

![Cascading Multiplexers Diagram]

control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

control signal A chooses which of the upper or lower mux's output to gate to Z

4-LUT Implementation

- n-bit LUT is implemented as a $2^n \times 1$ memory:
  - Inputs choose one of $2^n$ memory locations.
  - Memory locations (latches) are normally loaded with values from user's configuration bit stream.
  - Inputs to mux control are the CLB inputs.

- Result is a general purpose "logic gate"
  - n-LUT can implement any function of n inputs!
  - Example:

![4-LUT Implementation Diagram]
Example: Xilinx Virtex-E Floorplan

Configurable Logic Blocks
- 4-input function gens
- buffers
- flipflop

Input/Output Blocks
- combinational, latch, and flipflop output
- sampled inputs

Block RAM
- 4096 bits each
- every 12 CLB columns

Virtex-E Configurable Logic Block (CLB)

CLB = 4 logic cells (LC) in two slices
LC: 4-input function generator, carry logic, storage element
80 x 120 CLB array on 2000E

16x1 synchronous RAM
FF or latch
Details of Virtex-E Slice – implements any two 4-input functions

4-input function

3-input function; registered

2) Basic I/O Block (IOB) Structure

Three-State Control

Output Path

Input Path
IOB Functionality

- IOB provides interface between the package pins and CLBs
- Each IOB can work as uni- or bi-directional I/O
- Outputs can be forced into High Impedance
- Inputs and outputs can be registered
  - advised for high-performance I/O
- Inputs can be delayed

Example: Virtex-E IOB detail
3-a) Routing Resources: Interconnects

- Logic blocks embedded in a ‘sea’ of connection resources
- CLB = logic block
  IOB = I/O buffer
  PSM = programmable switch matrix (switch block)
- Interconnections critical
  - Transmission gates on paths
    - Flexibility
    - Connect any LB to any other
    - **but**
      - **×** Much slower than connections within a logic block
      - **×** Much slower than long lines on an ASIC

3-b) Routing Resources: Switch and Connection Boxes

- **Connection Box:** connects channel wires to the I/O pins of CLBs.
- **Switch Box:** allow wires to switch between vertical and horizontal wires.
3-c) Routing Resources: Switch Blocks

Connection Blocks

Connection to Output of CLB

Connection to Input of CLB
Example: SRAM-type FPGA Interconnection

- Millions of SRAM cells holding LUTs and Interconnect Routing info
- Volatile Memory. Loses configuration when board power is turned off
- Keep Bit Pattern describing the SRAM cells in non-Volatile Memory
- Configuration takes ~ secs

Configuring an FPGA

- Millions of SRAM cells holding LUTs and Interconnect Routing info
- Volatile Memory. Loses configuration when board power is turned off
- Keep Bit Pattern describing the SRAM cells in non-Volatile Memory
- Configuration takes ~ secs
Overview

- FPGA Devices
  - ASIC vs. FPGA
  - FPGA architecture
- FPGA Design Flow
  - Synthesis
  - Place
  - Route

Typical Digital IC Design Flow Vs. FPGA Design Flow
FPGA Generic Design Flow or Methodology

- **Design Entry:**
  - Create your design files using:
    - schematic editor or
    - hardware description language (VHDL, Verilog)

- **Design implementation on FPGA:**
  - *Partition, place, and route* to create bit-stream file

- **Design verification:**
  - Use Simulator to check function.
  - Load onto FPGA device (cable connects PC to development board)
  - Check operation at full speed in real environment

---

**VHDL description (Your Source Files)**

- **Synthesis**
- **Implementation**
- **Configuration**
- **Functional simulation**
- **Post-synthesis simulation**
- **Timing simulation**
- **On chip testing**
Logic Synthesis

VHDL description

architecture MLU_DATAFLOW of MLU is
begin
    A1<=A when (NEG_A='0') else not A;
    B1<=B when (NEG_B='0') else not B;
    Y<=Y1 when (NEG_Y='0') else not Y1;
    MUX_0<=A1 and B1;
    MUX_1<=A1 or B1;
    MUX_2<=A1 xor B1;
    MUX_3<=A1 xnor B1;
    with (L1 & L0) select
        Y1<=MUX_0 when "00",
        MUX_1 when "01",
        MUX_2 when "10",
        MUX_3 when others;
end MLU_DATAFLOW;

Circuit netlist

Implementation

- After synthesis the entire implementation process is performed by FPGA vendor tools
Translation

Synthesis
Circuit netlist

Timing Constraints
Electronic Design Interchange Format
EDIF
NCF
UCF

Constraint Editor
Native Constraint File
User Constraint File

Translation
NGD
Native Generic Database file

Pin Assignment

FPGA
top_level_design

CLOCK
CONTROL(0)
CONTROL(1)
CONTROL(2)
RESET
SEGMENTS(0)
SEGMENTS(1)
SEGMENTS(2)
SEGMENTS(3)
SEGMENTS(4)
SEGMENTS(5)
SEGMENTS(6)

H3
K2
K3
G5
H1
K4
G1
G2
H5
H4
H6
B10
B11
Circuit Netlist

Mapping
Placement

FPGA

CLB SLICES

Example placement (VPR tool)
Example placement (ISE tool)

Routing

FPGA

Programmable Connections
Example routing (VPR tool)

Example routing (VPR tool) – zoom-in
Once a design is implemented, you must create a file that the FPGA can understand

- This file is called a **bitstream**: a BIT file (.bit extension)

- The BIT file can be downloaded directly to the FPGA, or can be converted into a PROM file which stores the programming information
Map report

Design Summary

Number of errors: 0
Number of warnings: 0
Logic Utilization:
  Number of Slice Flip Flops: 30 out of 26,624 1%
  Number of 4 input LUTs: 38 out of 26,624 1%
Logic Distribution:
  Number of occupied Slices: 33 out of 13,312 1%
  Number of Slices containing only related logic: 33 out of 33 100%
  Number of Slices containing unrelated logic: 0 out of 33 0%
*See NOTES below for an explanation of the effects of unrelated logic

Total Number 4 input LUTs: 62 out of 26,624 1%
Number used as logic: 38
Number used as a route-thru: 24
Number of bonded IOBs: 10 out of 221 4%
  IOB Flip Flops: 7
Number of GCLKs: 1 out of 8 12%

Place & route report

Asterisk (*) preceding a constraint indicates it was not met.
  This may be due to a setup or hold violation.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Requested</th>
<th>Actual</th>
<th>Logic</th>
<th>Absolute</th>
<th>Number of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* TS_CLOCK = PERIOD TIMGRP &quot;CLOCK&quot; 5 ns</td>
<td>5.000ns</td>
<td>5.140ns</td>
<td>4</td>
<td>-0.140ns</td>
<td>5</td>
</tr>
<tr>
<td>HIGH 50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS_gen1Hz_Clock1Hz = PERIOD TIMGRP &quot;gen1 5.000ns 4.137ns 2 0.863ns 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;gen1Hz_Clock1Hz&quot; 5 ns HIGH 50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Post layout timing report

Clock to Setup on destination clock CLOCK

<table>
<thead>
<tr>
<th>Source Clock</th>
<th>Dest: Rise</th>
<th>Dest: Rise</th>
<th>Dest: Fall</th>
<th>Dest: Fall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+---------</td>
<td>+---------</td>
<td>+---------</td>
<td>+---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.140</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Timing summary:
-----------------
Timing errors: 9  Score: 543
Constraints cover 574 paths, 0 nets, and 187 connections

Design statistics:
Minimum period: 5.140ns (Maximum frequency: 194.553MHz)

Summary

- FPGAs are more and more prevalent!
- They are here to stay!
- They offer a flexible platform for increasingly complex systems
- Design automation tools (i.e., CAD tools) take care of the entire design process from VHDL → configuration bitstream file