Two competing implementation approaches

**ASIC**
- Application Specific Integrated Circuit
- Designed all the way from behavioral description to physical layout
- Designs must be sent for expensive and time-consuming fabrication in semiconductor foundry

**FPGA**
- Field Programmable Gate Array
- No physical layout design; design ends with a bitstream used to configure a device
- Bought off the shelf and reconfigured by designers themselves
Which Way to Go?

**ASICs**
- High performance
- Low power
- Low cost in high volumes

**FPGAs**
- Off-the-shelf
- Low development cost
- Short time to market
- Reconfigurability

Low power

Low cost in high volumes
What is an FPGA?

- Block RAMs
- Basic Logic Blocks
- I/O Blocks
- Block RAMs
Modern FPGA

(#Logic resources, #Multipliers/DSP units, #RAM_blocks)

Graphics based on The Design Warrior’s Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)
# Xilinx FPGA Families

<table>
<thead>
<tr>
<th>Technology</th>
<th>Low-cost</th>
<th>High-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>220 nm</td>
<td></td>
<td>Virtex</td>
</tr>
<tr>
<td>180 nm</td>
<td>Spartan II,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Spartan IIE</td>
<td></td>
</tr>
<tr>
<td>120/150 nm</td>
<td></td>
<td>Virtex II, Virtex II Pro</td>
</tr>
<tr>
<td>90 nm</td>
<td>Spartan 3</td>
<td>Virtex 4</td>
</tr>
<tr>
<td>65 nm</td>
<td></td>
<td>Virtex 5</td>
</tr>
<tr>
<td>45 nm</td>
<td>Spartan 6</td>
<td></td>
</tr>
<tr>
<td>40 nm</td>
<td></td>
<td>Virtex 6</td>
</tr>
<tr>
<td>28 nm</td>
<td>Artix 7</td>
<td>Virtex 7</td>
</tr>
<tr>
<td>Technology</td>
<td>Low-cost</td>
<td>Mid-range</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>130 nm</td>
<td>Cyclone</td>
<td></td>
</tr>
<tr>
<td>90 nm</td>
<td>Cyclone II</td>
<td></td>
</tr>
<tr>
<td>65 nm</td>
<td>Cyclone III</td>
<td>Arria I</td>
</tr>
<tr>
<td>40 nm</td>
<td>Cyclone IV</td>
<td>Arria II</td>
</tr>
<tr>
<td>28 nm</td>
<td>Cyclone V</td>
<td>Arria V</td>
</tr>
</tbody>
</table>

**Altera FPGA Families**
Spartan-3 Family Attributes
## Spartan-3 FPGA Family Members

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>Equivalent Logic Cells</th>
<th>CLB Array (One CLB = Four Slices)</th>
<th>Distributed RAM Bits (K=1024)</th>
<th>Block RAM Bits (K=1024)</th>
<th>Dedicated Multipliers</th>
<th>DCMs</th>
<th>Maximum User I/O</th>
<th>Maximum Differential I/O Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S50</td>
<td>50K</td>
<td>1,728</td>
<td>16 x 12</td>
<td>12K</td>
<td>72K</td>
<td>4</td>
<td>2</td>
<td>124</td>
<td>56</td>
</tr>
<tr>
<td>XC3S200</td>
<td>200K</td>
<td>4,320</td>
<td>24 x 20</td>
<td>30K</td>
<td>216K</td>
<td>12</td>
<td>4</td>
<td>173</td>
<td>76</td>
</tr>
<tr>
<td>XC3S400</td>
<td>400K</td>
<td>8,064</td>
<td>32 x 28</td>
<td>56K</td>
<td>288K</td>
<td>16</td>
<td>4</td>
<td>264</td>
<td>116</td>
</tr>
<tr>
<td>XC3S1000</td>
<td>1M</td>
<td>17,280</td>
<td>48 x 40</td>
<td>120K</td>
<td>432K</td>
<td>24</td>
<td>4</td>
<td>391</td>
<td>175</td>
</tr>
<tr>
<td>XC3S1500</td>
<td>1.5M</td>
<td>29,952</td>
<td>64 x 52</td>
<td>208K</td>
<td>576K</td>
<td>32</td>
<td>4</td>
<td>487</td>
<td>221</td>
</tr>
<tr>
<td>XC3S2000</td>
<td>2M</td>
<td>46,080</td>
<td>80 x 64</td>
<td>320K</td>
<td>720K</td>
<td>40</td>
<td>4</td>
<td>565</td>
<td>270</td>
</tr>
<tr>
<td>XC3S4000</td>
<td>4M</td>
<td>62,208</td>
<td>96 x 72</td>
<td>432K</td>
<td>1,728K</td>
<td>96</td>
<td>4</td>
<td>712</td>
<td>312</td>
</tr>
<tr>
<td>XC3S5000</td>
<td>5M</td>
<td>74,880</td>
<td>104 x 80</td>
<td>520K</td>
<td>1,872K</td>
<td>104</td>
<td>4</td>
<td>784</td>
<td>344</td>
</tr>
</tbody>
</table>

**Notes:**
1. Logic Cell = 4-input Look-Up Table (LUT) plus a ‘D’ flip-flop. "Equivalent Logic Cells" equals "Total CLBs" x 8 Logic Cells/CLB x 1.125 effectiveness.
# FPGA Nomenclature

**Pb-Free Packaging**

For additional information on Pb-free packaging, see [XAPP427](#): "Implementation and Solder Reflow Guidelines for Pb-Free Packages".

![Diagram](image)

<table>
<thead>
<tr>
<th>Device</th>
<th>Speed Grade</th>
<th>Package Type / Number of Pins</th>
<th>Temperature Range (T_J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S50</td>
<td>-4</td>
<td>VQ(G)100: 100-pin Very Thin Quad Flat Pack (VTQFP)</td>
<td>Commercial (0°C to 85°C)</td>
</tr>
<tr>
<td>XC3S200</td>
<td>-5</td>
<td>CP(G)132: 132-pin Chip-Size Package (CSP)</td>
<td></td>
</tr>
<tr>
<td>XC3S400</td>
<td></td>
<td>TOQ(G)144: 144-pin Thin Quad Flat Pack (TQFP)</td>
<td></td>
</tr>
<tr>
<td>XC3S1000</td>
<td></td>
<td>PQ(G)208: 208-pin Plastic Quad Flat Pack (PQFP)</td>
<td>Industrial (-40°C to 100°C)</td>
</tr>
<tr>
<td>XC3S51500</td>
<td></td>
<td>FT(G)256: 256-ball Fine-Pitch Thin Ball Grid Array (FTBGA)</td>
<td></td>
</tr>
<tr>
<td>XC3S2000</td>
<td></td>
<td>FG(G)320: 320-ball Fine-Pitch Ball Grid Array (FBGA)</td>
<td></td>
</tr>
<tr>
<td>XC3S4000</td>
<td></td>
<td>FG(G)456: 456-ball Fine-Pitch Ball Grid Array (FBGA)</td>
<td></td>
</tr>
<tr>
<td>XC3S50000</td>
<td></td>
<td>FG(G)76: 676-ball Fine-Pitch Ball Grid Array (FBGA)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FG(G)900: 900-ball Fine-Pitch Ball Grid Array (FBGA)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FG(G)1156: 1156-ball Fine-Pitch Ball Grid Array (FBGA)</td>
<td></td>
</tr>
</tbody>
</table>
FPGA Nomenclature Example

**XC3S1500-4FG320**

- **Spartan 3 family**
- 1500 k equivalent logic gates (1.5 M)
- speed grade -4 = standard performance
- 320 pins
- package type
FPGA Design Flow
Design and implement a simple unit permitting to speed up encryption with RC5-similar cipher with fixed key set on 8031 microcontroller. Unlike in the experiment 5, this time your unit has to be able to perform an encryption algorithm by itself, executing 32 rounds....

Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RC5_core
is
    port(
        clock, reset, encr_decr: in std_logic;
        data_input: in std_logic_vector(31 downto 0);
        data_output: out std_logic_vector(31 downto 0);
        out_full: in std_logic;
        key_input: in std_logic_vector(31 downto 0);
        key_read: out std_logic;
    );
end AES_core;

Specification / Pseudocode

On-paper hardware design
(Block diagram & ASM chart)

VHDL description (Your Source Files)

Functional simulation

Synthesis

Post-synthesis simulation
FPGA Design process (2)

Implementation

Configuration

Timing simulation

On chip testing
Tools used in FPGA Design Flow

- Xilinx XST
- Synplify Premier
- Xilinx ISE

Functionally verified VHDL code

Design
- VHDL code

Synthesis
- Netlist

Implementation
- Bitstream
Synthesis
Synthesis Tools

Xilinx XST

Synplify Premier

... and others
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

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 (RTL view)
Mapping
Xilinx XST Inputs/Outputs
Xilinx XST Inputs

- RTL VHDL and/or Verilog files
- Core files
  These files can be in either NGC or EDIF format. XST does not modify cores. It uses them to inform area and timing optimization surrounding the cores.
- Constraints – XCF
  Xilinx constraints file in which you can specify synthesis, timing, and specific implementation constraints that can be propagated to the NGC file.
Xilinx XST Outputs

- **NGC**
  Netlist file with constraint information

- **NGR**
  This is a schematic representation of the pre-optimized design shown at the Register Transfer Level (RTL). This representation is in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, and is generated after the HDL synthesis phase of the synthesis process.

- **LOG**
  This report contains the results from the synthesis run, including area and timing estimation.
RTL view in Synplify Premier

General logic structures can be recognized in RTL view
Crossprobing between RTL view and code

- Each port, net or block can be chosen by mouse click from the browser or directly from the RTL View.
- By double-clicking on the element its source code can be seen:

```
-- generate divided clock
IF enable_xmit_clk = '1' OR enable_rcv_clk = '1' THEN
  IF clk_cnt >= unsigned(div_msib_lsb) - 1 THEN
    clk_cnt <= |others => '0'|;
  ELSE
    clk_cnt <= unsigned(clk_cnt) + 1;
  END IF;
ELSE
  clk_cnt <= (0=>'1', others=>'0');
END IF;
END PROCESS clk_div;
```

- Reverse crossprobing is also possible: if section of code is marked, appropriate element of RTL View is marked too:
Technology View in Synplify Pro

Technology view is a mapped RTL view. It can be seen by pressing button or by double-click on “.srm” file.

As in case of “RTL View”, buttons can be used here.

Two additional buttons are enabled:
- show critical path
- open timing analyst

Pay attention: technology view is usually large and presented on number of sheets.
Viewing critical path

- Critical path can be viewed by pressing on
- Delay values are written near each component of the path
Timing Analyst

- Timing analyst opened by pressing on 

- Timing analyst gives a possibility to analyze different paths in the design

- Timing analyst can be opened only from Technology View
Implementation
Implementation

• After synthesis the entire implementation process is performed by FPGA vendor tools
Implementation

Writing VHDL SDF file 'time_sim.sdf' ...
INFO:NetListWriters:635 - The generated VHDL netlist contains Xilinx SIMPRIM simulation primitives and has to be used with SIMPRIM library for correct compilation and simulation.
INFO:NetListWriters - Xilinx recommends running separate simulations to check for setup by specifying the MAX field in the SDF file and for hold by specifying the MIN field in the SDF file. Please refer to Simulator documentation for more details on specifying MIN and MAX field in the SDF.
INFO:NetListWriters:665 - For more information on how to pass the SDF switches to the simulator, see your Simulator tool documentation.

Number of warnings: 0
Number of info messages: 3
Total memory usage is 186884 kilobytes

Created netgen log file 'time_sim.nlf'.
Implementation ver1->rev1: 0 error(s), 7 warning(s)
Implementation ended with warning(s).
Translation

Synthesis

Circuit netlist

Timing Constraints

Electronic Design Interchange Format

Native Constraint File

EDIF

NCF

UCF

Constraint Editor or Text Editor

User Constraint File

Translation

Native Generic Database file

NGD
<table>
<thead>
<tr>
<th>Description</th>
<th>FPGA Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seven Segment 0 'a'</td>
<td>H2</td>
</tr>
<tr>
<td>Seven Segment 0 'b'</td>
<td>H3</td>
</tr>
<tr>
<td>Seven Segment 0 'c'</td>
<td>H6</td>
</tr>
<tr>
<td>Seven Segment 0 'd'</td>
<td>H5</td>
</tr>
<tr>
<td>Seven Segment 0 'e'</td>
<td>G5</td>
</tr>
<tr>
<td>Seven Segment 0 'f'</td>
<td>G4</td>
</tr>
<tr>
<td>Seven Segment 0 'g'</td>
<td>H1</td>
</tr>
<tr>
<td>Seven Segment 0 'dp'</td>
<td>C2</td>
</tr>
<tr>
<td>Seven Segment 1 'a'</td>
<td>J1</td>
</tr>
<tr>
<td>Seven Segment 1 'b'</td>
<td>J2</td>
</tr>
<tr>
<td>Seven Segment 1 'c'</td>
<td>K2</td>
</tr>
<tr>
<td>Seven Segment 1 'd'</td>
<td>C3</td>
</tr>
<tr>
<td>Seven Segment 1 'e'</td>
<td>C1</td>
</tr>
<tr>
<td>Seven Segment 1 'f'</td>
<td>H4</td>
</tr>
<tr>
<td>Seven Segment 1 'g'</td>
<td>B1</td>
</tr>
<tr>
<td>Seven Segment 1 'dp'</td>
<td>J4</td>
</tr>
</tbody>
</table>

The segments of the display are labelled "a-g" and "dp" in the table above and the figure below.
Example of an UCF File

NET "CLOCK" LOC = "P10";
NET "reset" LOC = "B10";
NET "S_SEG0<6>" LOC = "H1";
NET "S_SEG0<5>" LOC = "G4";
NET "S_SEG0<4>" LOC = "G5";
NET "S_SEG0<3>" LOC = "H5";
NET "S_SEG0<2>" LOC = "H6";
NET "S_SEG0<1>" LOC = "H3";
NET "S_SEG0<0>" LOC = "H2";
Placing

FPGA

CLB SLICES
Routing

Programmable Connections

FPGA
Configuration

• Once a design is implemented, you must create a file that the FPGA can understand
  • This file is called a bit stream: 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
Two main stages of the FPGA Design Flow

Synthesis

Technology independent

RTL Synthesis
- Code analysis
- Derivation of main logic constructions
- Technology independent optimization
- Creation of “RTL View”

Map
- Mapping of extracted logic structures to device primitives
- Technology dependent optimization
- Application of “synthesis constraints”
- Netlist generation
- Creation of “Technology View”

Implementation

Technology dependent

Place & Route
- Placement of generated netlist onto the device
- Choosing best interconnect structure for the placed design
- Application of “physical constraints”

Configure
- Bitstream generation
- Burning device
Report files
Map report header

Xilinx Mapping Report File for Design 'Lab3Demo'

Design Information
------------------
Command Line : c:\Xilinx\bin\nt\map.exe -p 3S1500FG320-4 -o map.ncd -pr b -k 4 -cm area -c 100 Lab3Demo.ngd Lab3Demo.pcf
Target Device : xc3s1500
Target Package : fg320
Target Speed : -4
Mapper Version : spartan3 -- $Revision: 1.34 $
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%
Related and Unrelated Logic

Related logic is defined as being logic that shares connectivity – e.g. two LUTs are "related" if they share common inputs. When assembling slices, Map gives priority to combine logic that is related. Doing so results in the best timing performance.

Unrelated logic shares no connectivity. Map will only begin packing unrelated logic into a slice once 99% of the slices are occupied through related logic packing.

Note that once logic distribution reaches the 99% level through related logic packing, this does not mean the device is completely utilized. Unrelated logic packing will then begin, continuing until all usable LUTs and FFs are occupied. Depending on your timing budget, increased levels of unrelated logic packing may adversely affect the overall timing performance of your design.
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 errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>* TS_CLOCK = PERIOD TIMEGRP &quot;CLOCK&quot; 5 ns HIGH 50%</td>
<td>5.000ns</td>
<td>5.140ns</td>
<td>4</td>
<td>-0.140ns</td>
<td>5</td>
</tr>
<tr>
<td>TS_gen1Hz_Clock1Hz = PERIOD TIMEGRP &quot;gen1&quot; 5 ns HIGH 50%</td>
<td>5.000ns</td>
<td>4.137ns</td>
<td>2</td>
<td>0.863ns</td>
<td>0</td>
</tr>
</tbody>
</table>

Post layout timing report

Clock to Setup on destination clock CLOCK

<table>
<thead>
<tr>
<th>Src:Rise</th>
<th>Src:Fall</th>
<th>Src:Rise</th>
<th>Src:Fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Clock</td>
<td>Dest:Rise</td>
<td>Dest:Rise</td>
<td>Dest:Fall</td>
</tr>
<tr>
<td>CLOCK</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)