Outline I

- Introduction to Programmable Logic Device
- CPLD and FPGA Hardware Architectures
- Foundation and XACTstep Software
- Foundation Schematic Flow
- Library Types
- Foundation Schematic Design
  - Lab 1, Lab 2,
- Foundation HDL Editor & State Editor
  - Lab 3, Lab 4
- Combinational Logic Design
- Synchronous Logic Design
Outline II

◆ Input / Output Design
◆ Memory Design
  • Lab 5, Lab 6
◆ Foundation Simulation
  • Lab 7
◆ Simulation Script
  • Lab 8
◆ Design Implementation
◆ M1 Implementation Options
  • Lab 9
◆ Timing Analyzer
  • Lab 10

Outline III

◆ Design Constraints
◆ Constraints Editor
  • Lab11, Lab 12
◆ Floorplanning
◆ Configuration
◆ Hardware Debugger and XChecker Cable
  • Lab 13

林岡正
03-5773693 ext. 146
max@cic.edu.tw
http://www.cic.edu.tw/research/xilinx
Introduction to FPGA

Introduction to PLD

PLD : Programmable Logic Device
SPLD : Small/Simple Programmable Logic Device
CPLD : Complex Programmable Logic Device
FPGA : Field Programmable Gate Array
Main Features

- Field-programmable
- Reprogrammable
- In-circuit design verification
- Rapid prototyping
- Fast time-to-market
- No IC-test & NRE cost
- H/W emulation instead of S/W simulation
- Good software
- ...

Programmability

- Why programmable? Why reprogrammable?
  - Logic is implemented by programming the “configuration memory”
  - Various configuration memory technologies
    - One-Time Programmable: anti-fuse, EPROM
    - Reprogrammable: EPROM, EEPROM, Flash & SRAM
Programmable Combinational Logic

Product Term-based Building Block
- 2-level logic
- High fan-in

Look-up Table-based Building Block
- 4 to 5 inputs, fine grain architecture
- ROM-like

Programmable Register

* Typical register controls: clock, enable, preset/clear, ...
Programmable Interconnect

Typical routing resources: switching elements, local/global lines, clock buffers...

Programmable I/O

Typical I/O controls: direction, I/O registers, 3-state, slew rate, ...
Field-Programmability

◆ **Why filed-programmable?**
  • You can verify your designs at any time by configuring the FPGA/CPLD devices on board via the download cable or hardware programmer

![Diagram showing download cable, FPGA/CPLD, and output display]

Software Environment

◆ **Various design entries and interfaces**
  • HDL: Verilog, VHDL, ABEL, AHDL...
  • Graphic: Viewlogic, OrCAD, Cadence, Aldec...

◆ **Primitives & macrofunctions provided**
  • Primitive gates, arithmetic modules, flip-flops, counters, I/O elements, ...

◆ **Constraint-driven compilation/implementation**
  • Logic fitting, partition, placement & routing (P&R)

◆ **Simulation netlist generation**
  • Functional simulation & timing simulation netlist extraction

◆ **Programmer/download program**
FPGA/CPLD Benefits

<table>
<thead>
<tr>
<th></th>
<th>Full-Custom ICs</th>
<th>Cell-Based ICs</th>
<th>Gate Arrays</th>
<th>High-Density PLDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Integration Density</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>High-Volume device cost</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Low-volume device cost</td>
<td>✔ ✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Time to Market</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Risk Reduction</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Future Modification</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Development Tool</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Educational Purpose</td>
<td>✔</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

This training course will focus on FPGA devices.
Xilinx Customer Support

◆ **Xilinx & distributor Field Application Engineers**
  - 台灣Xilinx(美商智霖) (02)2758-8373
  - 科成公司 (02)8780-1216, (03)531-8512
  - 三商安富利 (02)2516-7303

◆ **On-line help and documentation**
  - Includes databook and application notes

◆ **North American factory technical support hotline**
  - e-mail: hotline@xilinx.com

◆ **Internet site**
  - Homepage - http://www.xilinx.com
  - CIC - http://www.cic.edu.tw/research/Xilinx

◆ **Technical seminars**
Xilinx Device Families

CPLD and FPGA Hardware Architectures
### Xilinx Device Families II

<table>
<thead>
<tr>
<th>Device Family</th>
<th>Reconfigurable Element</th>
<th>Logic Cell Structure</th>
<th>Usable/Typical Gates</th>
<th>Family Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3000A/L,</td>
<td>SRAM</td>
<td>LUT</td>
<td>1,000 ~ 6,000</td>
<td>3020A/L, 3030A/L, 3042A/L, 3064A/L, 3090A/L</td>
</tr>
<tr>
<td>XC3100A/L</td>
<td>SRAM</td>
<td>LUT</td>
<td>1,000 ~ 7,500</td>
<td>3120A, 3130A, 3142A/L, 3164A, 3190A/L, 3195A</td>
</tr>
<tr>
<td>XC4000E</td>
<td>SRAM</td>
<td>LUT</td>
<td>3,000 ~ 25,000</td>
<td>4003E, 4005E, 4006E, 4008E, 4010E, 4013E, 4020E, 4025E</td>
</tr>
<tr>
<td>XC4000EX/XL/XLA</td>
<td>SRAM</td>
<td>LUT</td>
<td>5,000 ~ 85,000</td>
<td>4005XL, 4010XL, 4013XL/XLA, 4020XL/XLA, 4028EX/XL/XLA, 4036EX/XL/XLA, 4044XL/XLA, 4052XL/XLA, 4062XL/XLA, 4085XL/XLA</td>
</tr>
<tr>
<td>XC4000XV</td>
<td>SRAM</td>
<td>LUT</td>
<td>125,00 ~ 250,000</td>
<td>40125XV, 40150XV, 40200XV, 40250XV</td>
</tr>
<tr>
<td>Spartan/XL</td>
<td>SRAM</td>
<td>LUT</td>
<td>2,000 ~ 40,000</td>
<td>S05/XL, S10/XL, S20/XL, S30/XL, S40/XL</td>
</tr>
<tr>
<td>XC5200</td>
<td>SRAM</td>
<td>LUT</td>
<td>2,000 ~ 15,000</td>
<td>5202, 5204, 5206, 5210, 5215</td>
</tr>
<tr>
<td>XC6200</td>
<td>SRAM</td>
<td>SOG(1)</td>
<td>9,000 ~ 64,000</td>
<td>6209, 6216, 6236, 6264</td>
</tr>
<tr>
<td>Virtex</td>
<td>SRAM</td>
<td>LUT</td>
<td>50K ~ 1M</td>
<td>V50, V100, V150, V200, V300, V400, V600, V800, V1000</td>
</tr>
<tr>
<td>XC9500/XL</td>
<td>FLASH</td>
<td>SOP</td>
<td>800 ~ 12,800</td>
<td>9536/XL, 9572/XL, 95108, 95144/XL, 95180, 95216, 95288/XL, 95432, 95576</td>
</tr>
</tbody>
</table>

**Note:**

(1) XC6200 series is a special family of fine-grain, sea-of-gates FPGAs.

---

### XC9500 FastFLASH™ CPLDs

- 5 volt in-system programmable (ISP) CPLDs
- 5 ns pin-to-pin
- 36 to 288 macrocells (6400 gates)
- Industry’s best pin-locking architecture
- 10,000 program/erase cycles
- Complete IEEE 1149.1 JTAG capability

![XC9500 FastFLASH diagram](image)
**XC9500 - Architectural Features**

- 5-V, Uniform, all pins fast, PAL-like architecture
- FastCONNECT switch matrix provides 100% routing with 100% utilization
- Most flexible pin-locking architecture
  - Design iterations without board rework
- Flexible function block
  - 36 inputs with 18 outputs
  - Expandable to 90 product terms per macrocell
  - Product term and global three-state enables
  - Product term and global clocks
  - Product term and global set/reset signals
- 3.3V/5V I/O operation
- Complete IEEE 1149.1 JTAG interface

**XC9500XL - Enhanced Features**

- 3.3-V in-system programmable family
- Highest performance 4ns/200MHz
  - Leading-edge computing applications
- Smallest packages (CSP)
  - Reduced board area, increased flexibility
- Robust 5V, 3.3V, 2.5V interfacing
  - Worry-free mixed voltage operation
- 3rd generation FastFLASH process
  - Fast programming, rapid cost reduction
- Wide 54-input function blocks
Complete Interconnectivity with FastCONNECT™

Each function block is like a 36V18!
XC9500 Macrocell

From FastCONNECT 36

to/from other macrocells

P-Term Allocator

SUM-Term Logic

XOR

Register

DT D

P-term Clk

P-term R/S

P-term OE

Global Clocks

Global R/S

Global OEs

3

2 or 4

36

to/from other macrocells

XC9500/XL Product Family

<table>
<thead>
<tr>
<th></th>
<th>9536/XL</th>
<th>9572/XL</th>
<th>95108</th>
<th>95144/XL</th>
<th>95216</th>
<th>95288/XL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macrocells</td>
<td>36</td>
<td>72</td>
<td>108</td>
<td>144</td>
<td>216</td>
<td>288</td>
</tr>
<tr>
<td>Usable Gates</td>
<td>800</td>
<td>1600</td>
<td>2400</td>
<td>3200</td>
<td>4800</td>
<td>6400</td>
</tr>
<tr>
<td>tPD (ns)</td>
<td>5/4</td>
<td>7.5/5</td>
<td>7.5</td>
<td>7.5/5</td>
<td>10</td>
<td>10/6</td>
</tr>
<tr>
<td>Registers</td>
<td>36</td>
<td>72</td>
<td>108</td>
<td>144</td>
<td>216</td>
<td>288</td>
</tr>
<tr>
<td>Max I/O</td>
<td>34</td>
<td>72</td>
<td>108</td>
<td>133</td>
<td>166</td>
<td>192</td>
</tr>
<tr>
<td>Packages</td>
<td>VQ44</td>
<td>PC44</td>
<td>PC44</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
</tr>
<tr>
<td></td>
<td>PC44</td>
<td>PC84</td>
<td>PC84</td>
<td>TQ100</td>
<td>TQ100</td>
<td>TQ100</td>
</tr>
<tr>
<td></td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
</tr>
<tr>
<td></td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
</tr>
<tr>
<td></td>
<td>PQ160</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
</tr>
<tr>
<td></td>
<td>BG352</td>
<td>BG352</td>
<td>BG352</td>
<td>BG352</td>
<td>BG352</td>
<td>BG352</td>
</tr>
</tbody>
</table>
FPGA Architecture

Configurable Logic Block (CLB)

◆ Combinatorial logic via lookup table
  • Any function(s) of available inputs

◆ Output registered and/or combinatorial
  • Latches in XC5000/XC4000X
Look Up Tables

- Combinatorial Logic is stored in 16x1 SRAM Look Up Tables (LUTs) in a CLB
- Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Capacity is limited by number of inputs, not complexity
- Choose to use each function generator as 4 input logic (LUT) or as high speed sync. dual port RAM

I/O Block (IOB)

- Periphery of identical I/O blocks
  - Input, output, or bidirectional
  - Registered, latched, or combinatorial (XC3000/XC4000)
  - Three-state output
  - Programmable output slew rate
Xilinx FPGA Routing

- Fast Direct Interconnect - CLB to CLB
- General Purpose Interconnect - Uses switch matrix
- Long Lines
  - Segmented across chip
  - Global clocks, lowest skew
  - 2 Tri-states per CLB for busses
- Other routing types in CPLDs and XC6200

XC4000 Series FPGA Architecture
XC4000E Features

◆ XC4000 main features...
  - Select-RAM memory
  - Fully PCI compliant (speed grade -2 and faster)
  - Abundant flip-flops
  - Flexible function generators
  - Dedicated high-speed carry logic
  - Wide edge decoders on each edge
  - Internal 3-state bus capability
  - 8 global low-skew clock or signal distribution networks
  - System performance beyond 80MHz
  - IEEE 1149.1-compatible boundary scan logic support
  - 12-mA sink current per XC4000E output
  - Readback Capability

XC4000E Configurable Logic Blocks I
XC4000E Configurable Logic Blocks

II

◆ 2 Four-input function generators (Look Up Tables)
  • 16x1 RAM or Logic function

◆ 2 Registers
  • Each can be configured as Flip Flop
  • Independent clock polarity
  • Synchronous and asynchronous Set/Reset

XC4000E Function Generators

◆ Two 4-input function generators
  • Independent inputs (any two functions of four inputs)

◆ One 3-input function generator
  • Combines four-input functions to make any five-input function and some functions of up to nine inputs

◆ Using write enable input turns look-up tables into RAM
**XC4000E Dedicated Carry Logic**

![Carry Logic Diagram]

**XC4000E Select-RAM Memory**

**Old**
- Asynchronous
- Timing Critical
- Longer Design Times

**New**
- Synchronous
- Dual Port
- Simple Timing
- Programmable during device operation or at start-up configuration
- 3 ns read time

**Comparison**
- **Old**
  - Asynchronous
  - Timing Critical
  - Longer Design Times
  - 4 ns read time

- **New**
  - Synchronous
  - Dual Port
  - Simple Timing
  - Programmable during device operation or at start-up configuration
  - 3 ns read time

**Features**
- **Old**
  - Single Port
- **New**
  - Dual Port
  - Optional Dual Port

**Notes**
- New select-RAM memory is programmable during device operation or at start-up configuration.

**Key Points**
- XC4000E devices offer dedicated carry logic and select-RAM memory options.
- Asynchronous and synchronous operation is available.
- Design times are longer for asynchronous operation.
- Timing criticality and simpler timing are offered for synchronous operation.

**Applications**
- These devices are beneficial for applications requiring high performance and flexibility in memory and logic design.

---

*Images and diagrams illustrative of features and configurations.*
Single-Port RAM

- Synchronous write, Asynchronous read
- 16x2 or 32x1 max per CLB

Dual-Port RAM

- One common synchronous write port
- Two asynchronous read ports
- 16x1 max per CLB
IOB Features

- Registered, latched, or direct input
- Registered or direct output
  - Clock enable on both registers
- Programmable “fast” slew rate
  - Default “slow” for noise reduction
- Programmable TTL or CMOS thresholds
  - Input and output; default TTL
- Tri-state output buffer
  - Controlled locally or by global GTS

XC4000E I/O Block Diagram
XC4000E Interconnect Resources

- **Single-length lines**
  - Switch matrix
  - PIPs
- **Double-length lines**
- **Longlines**
  - Vertical longlines
  - Horizontal longlines
  - Global longlines: global reset & 8 global buffers
- **Dedicated paths**
  - Fast carry paths

XC4000E Routing Resources

XC4000E Family

<table>
<thead>
<tr>
<th>Logic Cells</th>
<th>4003E</th>
<th>4005E</th>
<th>4006E</th>
<th>4008E</th>
<th>4010E</th>
<th>4013E</th>
<th>4020E</th>
<th>4025E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Logic Gates</td>
<td>238</td>
<td>466</td>
<td>608</td>
<td>770</td>
<td>950</td>
<td>1,368</td>
<td>1,862</td>
<td>2,432</td>
</tr>
<tr>
<td>Typ Gate Range* (Logic + Select-RAM)</td>
<td>3K</td>
<td>5K</td>
<td>6K</td>
<td>8K</td>
<td>10K</td>
<td>13K</td>
<td>20K</td>
<td>25K</td>
</tr>
<tr>
<td>Max I/O</td>
<td>2-5K</td>
<td>3-9K</td>
<td>4-12K</td>
<td>6-15K</td>
<td>7-20K</td>
<td>10-30K</td>
<td>13-40K</td>
<td>15-45K</td>
</tr>
<tr>
<td>Packages:</td>
<td>80</td>
<td>112</td>
<td>128</td>
<td>144</td>
<td>160</td>
<td>192</td>
<td>224</td>
<td>256</td>
</tr>
<tr>
<td>PQ84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
<td>PC84</td>
</tr>
<tr>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
<td>PQ100</td>
</tr>
<tr>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
<td>TQ144</td>
</tr>
<tr>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
</tr>
<tr>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
</tr>
<tr>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
</tr>
<tr>
<td>PG120</td>
<td>PG156</td>
<td>PG156</td>
<td>PG191</td>
<td>PG191</td>
<td>PG223</td>
<td>PG223</td>
<td>PG223</td>
<td>PG223</td>
</tr>
<tr>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
<td>BG225</td>
</tr>
</tbody>
</table>

*20-25% of CLBs as RAM

100% Footprint Compatible
XC4000X Enhanced Features

◆ XC4000X enhanced features...
  - 800 ~ 85,000 system gates
  - Highest Performance
    - 3.3V XC4000XL/XLA
    - 2.5V XC4000XV
  - 0.35um, 5LM SRAM process for XC4000XL/XLA
  - 0.25um, 5LM SRAM process for XC4000XV
  - 12 or 24 mA sink current per XC4000X output
  - Twice the routing resources
  - New high speed quad interconnect resources
  - VersaRing I/O added for pin assignment flexibility
  - Latch capability in CLBs
  - Improved carry logic for faster adders, multipliers and DSP functions

XC4000X I/O Block Diagram
**XC4000X Interconnect Hierarchy**

**Single & Double**

- Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)
Quad

- Quad Lines (XC4000X only)

XC4000X VersaRing™
### XC4000X Family I

<table>
<thead>
<tr>
<th>Typ Logic Gates</th>
<th>4005XL</th>
<th>4010XL</th>
<th>4013XL/XLA</th>
<th>4020XL/XLA</th>
<th>4028EX/XL/XLA</th>
<th>4036EX/XL/XLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typ System Gates</td>
<td>5,000</td>
<td>10,000</td>
<td>13,000</td>
<td>20,000</td>
<td>28,000</td>
<td>36,000</td>
</tr>
<tr>
<td>(Logic + Select-RAM)</td>
<td>9,000</td>
<td>20,000</td>
<td>30,000</td>
<td>40,000</td>
<td>56,000</td>
<td>72,000</td>
</tr>
<tr>
<td>Avail RAM bits</td>
<td>6,272</td>
<td>12,800</td>
<td>18,432</td>
<td>25,088</td>
<td>32,768</td>
<td>41,472</td>
</tr>
<tr>
<td>Number CLBs</td>
<td>196</td>
<td>400</td>
<td>576</td>
<td>784</td>
<td>1,024</td>
<td>1,296</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>616</td>
<td>1,120</td>
<td>1,536</td>
<td>2,016</td>
<td>2,560</td>
<td>3,168</td>
</tr>
<tr>
<td>I/O</td>
<td>112</td>
<td>160</td>
<td>192</td>
<td>224</td>
<td>256</td>
<td>288</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5/3</td>
<td>5/3</td>
</tr>
<tr>
<td>Packages</td>
<td>PC84</td>
<td>PC84</td>
<td>PQ100</td>
<td>PQ100</td>
<td>HQ160</td>
<td>HQ160</td>
</tr>
<tr>
<td></td>
<td>TQ144</td>
<td>TQ144</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
</tr>
<tr>
<td></td>
<td>PQ160</td>
<td>PQ160</td>
<td>PQ160</td>
<td>HQ208</td>
<td>HQ208</td>
<td>HQ208</td>
</tr>
<tr>
<td></td>
<td>HQ208</td>
<td>HQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
<td>PQ208</td>
</tr>
<tr>
<td></td>
<td>BG256</td>
<td>BG256</td>
<td>BG256</td>
<td>BG256</td>
<td>BG256</td>
<td>BG256</td>
</tr>
<tr>
<td></td>
<td>TQ176</td>
<td>TQ176</td>
<td>BQ352</td>
<td>BQ352</td>
<td>BQ352</td>
<td>BQ352</td>
</tr>
<tr>
<td></td>
<td>VQ100</td>
<td>VQ100</td>
<td>PG299</td>
<td>PG299</td>
<td>PG411</td>
<td>PG411</td>
</tr>
</tbody>
</table>

100% Footprint Compatible

* 30% of CLBs as RAM

### XC4000X Family II

<table>
<thead>
<tr>
<th>Typ Logic Gates</th>
<th>4044XL/XLA</th>
<th>4052XL/XLA</th>
<th>4062XL/XLA</th>
<th>4085XL</th>
<th>40125XV</th>
<th>40250XV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typ System Gates</td>
<td>44,000</td>
<td>52,000</td>
<td>62,000</td>
<td>85,000</td>
<td>125,000</td>
<td>250,000</td>
</tr>
<tr>
<td>(Logic + Select-RAM)</td>
<td>90,000</td>
<td>110,000</td>
<td>130,000</td>
<td>175,000</td>
<td>250,000</td>
<td>500,000</td>
</tr>
<tr>
<td>Avail RAM bits</td>
<td>51,200</td>
<td>61,952</td>
<td>73,728</td>
<td>100,352</td>
<td>157,968</td>
<td>270,848</td>
</tr>
<tr>
<td>Number CLBs</td>
<td>1,600</td>
<td>1,936</td>
<td>2,304</td>
<td>3,136</td>
<td>4,624</td>
<td>8,464</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>3,840</td>
<td>4,576</td>
<td>5,376</td>
<td>7,168</td>
<td>10,336</td>
<td>18,400</td>
</tr>
<tr>
<td>I/O</td>
<td>320</td>
<td>352</td>
<td>384</td>
<td>448</td>
<td>448</td>
<td>448</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2.5</td>
<td>2.5</td>
</tr>
<tr>
<td>Packages</td>
<td>HQ208</td>
<td>HQ240</td>
<td>HQ240</td>
<td>HQ304</td>
<td>HQ304</td>
<td>HQ304</td>
</tr>
<tr>
<td></td>
<td>HQ240</td>
<td>HQ304</td>
<td>HQ304</td>
<td>BG352</td>
<td>BG352</td>
<td>BG432</td>
</tr>
<tr>
<td></td>
<td>BG352</td>
<td>BG352</td>
<td>BG432</td>
<td>BG432</td>
<td>BG432</td>
<td>BG432</td>
</tr>
<tr>
<td></td>
<td>BG560</td>
<td>BG560</td>
<td>BG560</td>
<td>BG560</td>
<td>BG560</td>
<td>BG560</td>
</tr>
<tr>
<td></td>
<td>PG411</td>
<td>PG411</td>
<td>PG475</td>
<td>PG559</td>
<td>PG599</td>
<td>PG599</td>
</tr>
</tbody>
</table>

100% Footprint Compatible

* 30% of CLBs as RAM
Spartan/XL Features

**Spartan/XL main features...**
- First ASIC replacement FPGA for high-volume production with on-chip RAM
- Advanced Ultradense 0.35um/0.5um process
- 2,000 ~ 40,000 gates
- Streamlined feature set based on XC4000 architecture
- High Performance - over 80 MHz system speeds
- Available in both 5- and 3-V versions
- Fully PCI compliant
- Low power segmented routing architecture
- Full readback capability
- Dedicated high-speed carry logic
- Internal 3-state bus capability
- 8 global low-skew clock or signal distribution networks
- IEEE 1149.1-compatible boundary scan logic support
- Versatile I/O and packaging products
Basic Architecture

- Array of CLBs surrounded by perimeter IOBs
- Special functions in corners

Spartan Configurable Logic Block (CLB)
Spartan Speed Grades

- Higher Spartan speed grade = higher performance
- Higher performance Spartan-5 speed grade available in ‘98

![Performance Graph]

- Spartan-4 faster than XC4000E-1
- Spartan-3 faster than XC4000E-2
- Spartan much faster than 5200

Spartan Family

<table>
<thead>
<tr>
<th>Part</th>
<th>Gates</th>
<th>CLB Matrix</th>
<th>CLBs</th>
<th>Flip-Flops</th>
<th>IOBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S05/XL</td>
<td>2,000 ~ 5,000</td>
<td>10x10</td>
<td>100</td>
<td>360</td>
<td>80</td>
</tr>
<tr>
<td>S10/XL</td>
<td>3,000 ~ 10,000</td>
<td>14x14</td>
<td>196</td>
<td>616</td>
<td>112</td>
</tr>
<tr>
<td>S20/XL</td>
<td>7,000 ~ 20,000</td>
<td>20x20</td>
<td>400</td>
<td>1,120</td>
<td>160</td>
</tr>
<tr>
<td>S30/XL</td>
<td>10,000 ~ 30,000</td>
<td>24x24</td>
<td>576</td>
<td>1,536</td>
<td>192</td>
</tr>
<tr>
<td>S40/XL</td>
<td>13,000 ~ 40,000</td>
<td>28x28</td>
<td>784</td>
<td>2,016</td>
<td>224</td>
</tr>
</tbody>
</table>
**Next: Virtex**

- **The Virtex™ family redefines the future of programmable logic**
  - break density and performance barriers
  - offer unprecedented system level integration.
- **Virtex series devices range from 50,000 to 1,000,000 system gates at clock speeds up to 200 MHz**
  - include many new features that address system level design challenges.

**Virtex Features**

- **Virtex Architecture Highlights**
  - Nine devices, from 50,000 to 1,000,000 system gates (1,728 to 27,648 Logic Cells)
  - Over 500 user I/O pins
  - Many package options, including leading edge 1.0mm FinePitch ball grid arrays and 0.8mm chip scale packages
  - Leading edge 2.5-Volt, 0.22 micron, five layer metal CMOS process
  - Fully 5-Volt tolerant I/Os
  - Timing-driven place and route tools allow compile times of 200,000 gates per hour (400 MHz Pentium II CPU)
**Virtex Features**

- **System-Critical Features**
  - Four fully digital Delay Locked Loops (DLLs) for system clock synchronization
    - 200+ MHz chip-to-chip communication
    - Less than 3ns clock-to-output time across all devices
    - Clock doubling and clock division
    - 0°, 90°, 180°, and 270° phase clocks
  - Select I/O™ technology supports multiple voltage and signal standards, simultaneously
    - Support for LVTTL, LVCMOS2, PCI33, PCI66, GTL/GTL+, SSTL, HSTL, AGP, and, CTT
  - Supports 3 levels of memory hierarchy
    - The Virtex SelectRAM+ memory hierarchy provides high bandwidth for memory block sizes in bytes (distributed memory), kilobytes (block memory), and megabytes (SSTL3 interfacing to external DRAM and SRAM).
  - Vector-based interconnect for fast, predictable, core-friendly routing across all densities
  - Fully 64 bit/66 MHz PCI and Compact PCI compliant

---

**Virtex Architecture Overview**

![Virtex Architecture Diagram](image-url)
Virtex Block SelectRAM+ Amounts

<table>
<thead>
<tr>
<th>Virtex Device</th>
<th># of Blocks</th>
<th>Total Block SelectRAM+ Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCV50</td>
<td>8</td>
<td>32,788</td>
</tr>
<tr>
<td>XCV100</td>
<td>10</td>
<td>40,960</td>
</tr>
<tr>
<td>XCV150</td>
<td>12</td>
<td>49,152</td>
</tr>
<tr>
<td>XCV200</td>
<td>14</td>
<td>57,344</td>
</tr>
<tr>
<td>XCV300</td>
<td>16</td>
<td>65,536</td>
</tr>
<tr>
<td>XCV400</td>
<td>20</td>
<td>81,920</td>
</tr>
<tr>
<td>XCV600</td>
<td>24</td>
<td>98,304</td>
</tr>
<tr>
<td>XCV800</td>
<td>28</td>
<td>114,688</td>
</tr>
<tr>
<td>XCV1000</td>
<td>32</td>
<td>131,072</td>
</tr>
</tbody>
</table>

Virtex Input/Output Block (IOB)
Supported Select I/O Standards

<table>
<thead>
<tr>
<th>I/O Standard</th>
<th>Input Reference Voltage (V_REF)</th>
<th>Output Source Voltage (V_COO)</th>
<th>Board Termination Voltage (V_TT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVTTL 2-24 mA</td>
<td>N/A</td>
<td>3.3</td>
<td>N/A</td>
</tr>
<tr>
<td>LVCMOS2</td>
<td>N/A</td>
<td>2.5</td>
<td>N/A</td>
</tr>
<tr>
<td>PCI</td>
<td>N/A</td>
<td>3.3</td>
<td>N/A</td>
</tr>
<tr>
<td>GTL</td>
<td>0.8</td>
<td>N/A</td>
<td>1.2</td>
</tr>
<tr>
<td>GTL+</td>
<td>1.0</td>
<td>N/A</td>
<td>1.5</td>
</tr>
<tr>
<td>HSTL Class I</td>
<td>0.75</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>HSTL Class III</td>
<td>0.75</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>HSTL Class IV</td>
<td>0.75</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>SSTL3 Class I and II</td>
<td>1.5</td>
<td>3.3</td>
<td>1.5</td>
</tr>
<tr>
<td>SSTL2 Class I and II</td>
<td>1.125</td>
<td>2.5</td>
<td>1.125</td>
</tr>
<tr>
<td>CTT</td>
<td>1.5</td>
<td>3.3</td>
<td>1.5</td>
</tr>
<tr>
<td>AGP</td>
<td>1.32</td>
<td>3.3</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Global Clock Distribution Network

Global Clock Rows

Global Clock Column

Global Clock Spine

Global Clock Rows

GCLKPAD3

GCLKBUF3

GCLKPAD2

GCLKBUF2

Global Clock Spine

GCLKBUF1

GCLKPAD1

GCLKBUF0

GCLKPAD0
Virtex Family

<table>
<thead>
<tr>
<th>Device</th>
<th>System Gates</th>
<th>CLB Array</th>
<th>Logic Cells</th>
<th>Maximum Available I/O</th>
<th>BlockRAM Bits</th>
<th>Max Select RAM Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCV50</td>
<td>57,906</td>
<td>16x24</td>
<td>1,728</td>
<td>180</td>
<td>32,768</td>
<td>24,576</td>
</tr>
<tr>
<td>XCV100</td>
<td>108,904</td>
<td>20x30</td>
<td>2,700</td>
<td>180</td>
<td>40,960</td>
<td>38,400</td>
</tr>
<tr>
<td>XCV150</td>
<td>164,674</td>
<td>24x36</td>
<td>3,888</td>
<td>260</td>
<td>49,152</td>
<td>55,296</td>
</tr>
<tr>
<td>XCV200</td>
<td>236,686</td>
<td>28x42</td>
<td>5,292</td>
<td>284</td>
<td>57,344</td>
<td>75,264</td>
</tr>
<tr>
<td>XCV300</td>
<td>322,970</td>
<td>32x48</td>
<td>6,912</td>
<td>315</td>
<td>65,536</td>
<td>98,304</td>
</tr>
<tr>
<td>XCV400</td>
<td>468,252</td>
<td>40x60</td>
<td>10,800</td>
<td>404</td>
<td>81,920</td>
<td>153,600</td>
</tr>
<tr>
<td>XCV600</td>
<td>661,111</td>
<td>48x72</td>
<td>15,552</td>
<td>500</td>
<td>98,394</td>
<td>221,184</td>
</tr>
<tr>
<td>XCV800</td>
<td>888,439</td>
<td>56x84</td>
<td>21,168</td>
<td>514</td>
<td>114,688</td>
<td>301,056</td>
</tr>
<tr>
<td>XCV1000</td>
<td>1,124,022</td>
<td>64x96</td>
<td>27,648</td>
<td>514</td>
<td>131,072</td>
<td>393,216</td>
</tr>
</tbody>
</table>

Other FPGA Resources

- Tri-state buffers for busses (BUFT's)
- Global clock & high speed buffers (BUFG's)
- Wide Decoders (DECODEx)
- Internal Oscillator (OSC4)
- Global Reset to all Flip-Flops, Latches (STARTUP)
- CLB special resources
  - Fast Carry logic built into CLBs
  - Synchronous Dual Port RAM
  - Boundary Scan
Naming Conventions

- Xilinx Component naming convention: Part name - speed- package.

Example:

XC4028XL-3-BG256

- Package
- Speed Grade
- Sub-family (3V = Xl or L, no XL or L = 5V)
- Maximum number of gates (thousands)
- Family (XC4000, XC9500, XCS…)
- S = Spartan

The speed grade is a relative measure of internal delay. Smaller numbers mean faster parts for all families EXCEPT Spartan. For Spartan and all future devices, larger numbers mean faster parts.

CPLD or FPGA?

CPLD
- Non-volatile
- JTAG Testing
- Wide fan-in
- Fast counters, state machines
- Combinational Logic
- Small student projects, lower level courses

FPGA
- SRAM reconfiguration
- Excellent for computer architecture, DSP, registered designs
- ASIC like design flow
- Great for first year to graduate work
- More common in schools
- PROM required for non-volatile operation
Density & Cost Roadmap

Cost

5,000 85,000
Logic Gates

0.4K 3K 7.5K 20.1K
Logic Cells

XC4000E

XC4000EX

XC4036EX

XC4000XL

XC4085XL

XC4000XY

XC40250XV

Next Generation FPGAs: Virtex

1M Gates

XC4085XL

XC40250XV

1M Gates

Appendix

XC3000/XC5000/XC6000/Hardwire
XC3000 Features

◆ XC3000 main features...
  • Complete line of 4 related FPGA product families
    – XC3000A, XC3000L, XC3100A, XC3100L
  • 1,000 ~ 7,500 gates
  • High fan-out signal distribution, low-skew clock nets
  • Internal 3-state bus capabilities
  • System clock speeds over 85MHz

◆ XC3100A enhanced features...
  • Ultra-high-speed FPGA family
    – 1.55 ~ 4.1ns logic delays
  • 8 mA output sink current and 8mA source current
  • High-end additional family member XC3195A
  • PCI compliant (-2, -1, -09 speed grade)
XC3000 CLB II

- 1 configurable function generator
- CLB inputs
  - 5 general-purpose (A-E)
  - Internal feedback from flip-flops
- 2 CLB outputs
  - F, G, or registered

XC3000 Configurable Function Generator

- 7 shared sources
  - 5 general-purpose inputs (A-E)
  - 2 storage element feedbacks (QX, QY)
- Lookup table configurations
  - Any function of 5 inputs
  - 2 functions of 4 inputs
**XC3000 I/O Block Diagram**

- **T/ŒE**
- **O**
- **OK (Output Clock)**
- **I**
- **Q**
- **IK (Input Clock)**

- **Slew Rate Control**
- **Passive Pull-Up**
- **Output Buffer**
- **Input Buffer TTL or CMOS**
- **Vcc**
- **Pad**

**XC3000 Interconnect Resources**

- **General-purpose interconnect**
  - Switching matrix
  - PIPs (Programmable Interconnect Point)

- **Direct interconnect**
  - X -> right B / left C
  - Y -> above D / below A
  - Die edge direct interconnect

- **Longlines**
  - 3 vertical longlines per column
  - 2 horizontal longlines per row
  - 2 additional outer longlines
  - 2 Global nets: global buffer, alternate buffer & global reset
**XC3000 Family**

<table>
<thead>
<tr>
<th>Part</th>
<th>Typical Gates</th>
<th>CLB Matrix</th>
<th>CLBs</th>
<th>Flip-Flops</th>
<th>IOBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>3120A</td>
<td>1300-1800</td>
<td>8x8</td>
<td>64</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>3130A</td>
<td>2000-2700</td>
<td>10x10</td>
<td>100</td>
<td>360</td>
<td>80</td>
</tr>
<tr>
<td>3142A</td>
<td>2500-3700</td>
<td>12x12</td>
<td>144</td>
<td>480</td>
<td>96</td>
</tr>
<tr>
<td>3164A</td>
<td>4000-5500</td>
<td>14x16</td>
<td>224</td>
<td>688</td>
<td>120</td>
</tr>
<tr>
<td>3190A</td>
<td>5000-7000</td>
<td>16x20</td>
<td>320</td>
<td>928</td>
<td>144</td>
</tr>
<tr>
<td>3195A</td>
<td>6500-8500</td>
<td>22x22</td>
<td>484</td>
<td>1320</td>
<td>176</td>
</tr>
</tbody>
</table>

**XC5200 Features**

**XC5200 main features...**
- 0.5um 3 ML SRAM
- 3,000 ~ 23,000 gates
- Price competitive with Gate Array
- System Performance beyond 50 MHz
- 6 levels of interconnect hierarchy
- VersaRing I/O Interface for pin-locking
- Dedicated carry logic for high-speed arithmetic functions
- Cascade chain for wide input functions
- Built-in IEEE 1149.1 JTAG boundary scan test circuit on all I/O pins
- Internal 3-state bussing capability
- 4 dedicated low-skew clock or signal distribution nets
XC5200 CLB (1 of 4 Logic Cells)

XC5200 Function Generators

- Four four-input functions
- Two five-input functions
  - 5-input AND uses only one lookup table
- Cascaded sixteen-input function
**XC5200 I/O Block Diagram**

![I/O Block Diagram](image)

**XC5200 Interconnect Resources**

- **Hierarchical routing resources**
  - Logic cell feedthrough path
  - VersaBlock routing
    - Local interconnect matrix
    - Direct connects
  - GRM: general routing matrix
    - Single-length lines
    - Double-length lines
    - Longlines
    - Global lines
  - VersaRing I/O interface
**XC5200 VersaBlock & VersaRing**

**VersaBlock**
- Abundant local routing plus versatile logic
- 5 independent inputs & 3 outputs to each LC
- Each LC contains a direct feedthrough path

**VersaRing I/O interface**
- Abundant connections from each IOB to the nearest VersaBlock, in addition to Longline connections surrounding the device
- Increase pin-locking flexibility

---

**XC5200 Family**

<table>
<thead>
<tr>
<th>Part</th>
<th>Gates</th>
<th>CLB Matrix</th>
<th>CLBs</th>
<th>Flip-Flops</th>
<th>IOBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>5202</td>
<td>3000</td>
<td>8x8</td>
<td>64</td>
<td>256</td>
<td>84</td>
</tr>
<tr>
<td>5204</td>
<td>6000</td>
<td>10x12</td>
<td>120</td>
<td>480</td>
<td>124</td>
</tr>
<tr>
<td>5206</td>
<td>10000</td>
<td>14x14</td>
<td>196</td>
<td>784</td>
<td>148</td>
</tr>
<tr>
<td>5210</td>
<td>16000</td>
<td>18x18</td>
<td>324</td>
<td>1296</td>
<td>196</td>
</tr>
<tr>
<td>5215</td>
<td>23000</td>
<td>22x22</td>
<td>484</td>
<td>1936</td>
<td>244</td>
</tr>
</tbody>
</table>
**XC6200 Architecture**

- 16x16 Tile
- FastMAP™ Interface
- Address
- Data
- Control
- User I/Os

*Number of tiles varies between devices in family*

---

**HardWire™**

- Unique no-risk 100% compatible mask-programmed cost reduction of Xilinx FPGA
- Cost-effective for volume applications
  - Savings of 40% to 70%
- Architecture-equivalent mask-programmed version of any FPGA
  - Requires virtually no customer engineering resources, test vectors, or simulation
  - ALL FPGA features (e.g., Configuration, Power-On Reset, JTAG, etc.) are fully supported
### Family Selection Guidelines I

<table>
<thead>
<tr>
<th>Feature</th>
<th>E</th>
<th>EX</th>
<th>XL</th>
<th>5200</th>
<th>6200</th>
<th>9500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Pin-To-Pin</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Fastest State Machine</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Fastest Arithmetic Counters</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Bi-directional Busses</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DSP (Multiply/Accumulate)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>RAM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

### Family Selection Guidelines II

<table>
<thead>
<tr>
<th>Feature</th>
<th>E</th>
<th>EX</th>
<th>XL</th>
<th>5200</th>
<th>6200</th>
<th>9500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tolerant of pin-locking</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Footprint Compatible Families</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>In-System Programmable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Boundary Scan</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Fast/Partial Configuration</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-Volatile/single chip (no PROM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Low Standby Current</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
### Family Selection Guidelines III

<table>
<thead>
<tr>
<th>Feature</th>
<th>E</th>
<th>EX</th>
<th>XL</th>
<th>5200</th>
<th>6200</th>
<th>9500</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTL &amp; CMOS 5v Output</td>
<td>T, C</td>
<td>T, C</td>
<td>T</td>
<td>T, C</td>
<td>T, C</td>
<td>T, C</td>
</tr>
<tr>
<td>3.3 V Operation</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hi-Rel, Mil, Mil-Temp</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Foundation and XACTstep™ Software

XACTstep™ M1 Software

ALLIANCE Series

Software Backplane

Foundation Series

Libraries and Interfaces for Leading EDA Vendors

Core Implementation Software - Map, Place, Route, Bitstream generation, and analysis

Complete, Ready-to-Use
Includes Schematic, Simulation, and ABEL/VHDL/Verilog Synthesis
Foundation Series Overview

◆ Integrated Aldec front end and Xilinx implementation tools
  • Aldec Project Manager can invoke Xilinx Design Manager and other tools

◆ Aldec tools:
  • Schematic capture
  • Gate-level simulation
  • State Editor
  • HDL Editor
  • VHDL/Verilog/ABEL Language Assistant

◆ Includes Viewlogic schematic import feature
◆ Includes on-line documentation and tutorials
◆ Synopsys FPGA Express (VHDL, Verilog)
Foundation Project Manager

Schematic Entry
VHDL/Verilog/ABEL Text Entry

- From schematic menu (or via HDL Editor), select Hierarchy -> New Symbol Wizard... to create symbol.
- Select HDL Editor & Language Assistant to learn by example, then define block.
- Synthesize to EDIF.

State Machine Graphical Editor

Graphical editor synthesizes into VHDL/Verilog/ABEL code
Foundation Simulator

Implementation - M1 Design Manager

- Manages design data
- Access reports
- Supports CPLDs, FPGAs

Flow Engine
Timing Analyzer
Floorplanner
PROM File Formatter
Hardware Debugger
EPIC Design Editor
Flow Engine

- View status of tools
- Control tool options
- Implement design to the bitstream

Floorplanner
Online Documentation Viewer

◆ Xilinx Books - Dynatext
  • "開始" -> "程式集" -> "Xilinx Foundation Series" -> "Online Documentation Viewer"
  • dtext.exe

Foundation Express Installation

◆ Software Protection
  • CD Key
  • FLEXLM Floating License

◆ Installation
  • http://www.cic.edu.tw/research/xilinx/install/fndinst15.html

◆ Setup Env. Variable
  set XILINX = E:\FNDTN
  set PATH = E:\FNDTN\BIN\NT;%PATH%
  set LM_LICENSE_FILE = 2200@nt-server (假設 License Server 為 nt-server)

◆ 如何安裝 FLEXlm License Server？
  • http://www.cic.edu.tw/research/xilinx/Faq/faq09.html

◆ 不同網域如何 check License Server？
  • http://www.cic.edu.tw/research/xilinx/Faq/faq12.html
Foundation Schematic Flow

Schematic Flow Project Processing I

- Create Project
- Select Schematic Flow
- Select Target
- Create Top-Level Schematic
- Add Macros
  - Schematic, FSM, LogiBLOX, HDL
- Add Hierarchy?
- Functional Simulation
  - (analyze Logic)
- Reports

Design Entry
Project Manager Overview

- **Foundation organizes related files into a distinct logical unit called a project**
- **Manages and supervises all Foundation Series tools involved in the design process**
- **Integrates all Foundation tools into a unified environment.**
- **This environment includes such tools as**
  - Schematic Editor
  - HDL Editor
  - State Diagram Editor
  - Fast gate-level Logic Simulator
  - External third-party programs
- **Project Manager is designed to work with one project at a time.**
Project Manager Performs Functions

- Automatically loads all design resources when a project is open.
- Checks if all project resources are available and up-to-date.
- Shows the design process flow.
- Provides buttons for launching applications involved in the design process.
- Provides interface to external third-party programs.
- Places all error and status messages in the message window.
- Provides automated data transfer between tools involved in processing your designs.
- Provides design status information.

New Schematic Project

- Open the project Manager
  - Start > Programs > Xilinx Foundation Series > Xilinx Foundation Project Manager
- Create a New Project
  - On the getting started dialog box: click the Create a New Project radio button
  - Select the Schematic Flow
A project includes:
- Project documents (Schematics, HDL source files, and state diagram files)
- Project libraries
- Output and intermediate files (netlists, bitstreams, report and log files)
- Configuration files

Files Tab
- For new projects, the Project Manager automatically creates the following files:
  - A configuration file called the Project Description File (.pdf)
  - A user constraints file (.ucf)
  - Three types of library files (project library, Simprims library, and device library)
- A Foundation project always has one or more “top-level” design files
  - In Schematic Flow, all top-level files must be schematics, state diagrams, or ABEL files
  - In HDL Flow, you designate the top-level entity(VHDL) or module(Verilog) at the time of synthesis.
Hierarchy Browser II

<table>
<thead>
<tr>
<th>Extension</th>
<th>File Type</th>
<th>Created By</th>
</tr>
</thead>
<tbody>
<tr>
<td>.pdf</td>
<td>Project description file</td>
<td>Project Manager</td>
</tr>
<tr>
<td>.sch</td>
<td>Schematic source file</td>
<td>Schematic Capture</td>
</tr>
<tr>
<td>.v</td>
<td>Verilog source file</td>
<td>HDL Editor</td>
</tr>
<tr>
<td>.vhd</td>
<td>VHDL source file</td>
<td>HDL Editor</td>
</tr>
<tr>
<td>.abl</td>
<td>XABEL source file</td>
<td>HDL Editor</td>
</tr>
<tr>
<td>.asf</td>
<td>FSM(ABEL) source file</td>
<td>FSM Editor</td>
</tr>
<tr>
<td>.ucf</td>
<td>User constraints file</td>
<td>Project Manager</td>
</tr>
<tr>
<td>.tve</td>
<td>Test vector file</td>
<td>Logic Simulator</td>
</tr>
</tbody>
</table>

◆ Versions Tab
- Display the revisions and versions of the chip implementations of the design
- For a newly created project, this tab is empty

Project Flowchart Area

◆ Flow Tab - Project Flowchart
- Display the project flowchart
- When you start programs from the project flowchart, the Project Manager automatically controls the transfer of input and output data between applications

◆ Contents Tab
- Display info related to the object currently selected (file, library, etc.) from the hierarchy tree on the Files tab

◆ Reports Tab
- Access and display reports that have been generated in the design process

◆ Synthesis Tab (Schematic Flow Only)
- Update or synthesize VHDL, Verilog, ABEL, and State Machine macros
Documenting Your Design

To attach text files or other files to the Project, perform the following steps.

2. In the Add Document dialog box, select the documents from the Files list box.
3. Click OK.
Project Archiving

◆ The entire project directory, plus files needed to restore Foundation Series and system settings, can be zipped into a single file.

- When you select *File > Archive Project* from the Project Manager, the Archive Project Wizard - Setup window appears.
- In Archive Project Wizard - Setup window, you can specify the location for the archive .zip file, add comments, provide a password, or modify the compression factor.
- Foundation Project Manager contains a Restore Project option to automatically unzip archived projects. (*File > Restore Project*).
Library Types

◆ Library Types
  • System Libraries
  • User Libraries

◆ Project Libraries
  • System libraries - These libraries are automatically added to the project by rule. ex: xc4000e(Unified Library), simprims(Simulation Library)
  • Project working library - It is the default storage for user macros generated throughout the development of the project. This library is created automatically when you are setting up a new project.
System Libraries Overview

◆ Libraries contain descriptions of each component with pin names, functionality, timing, etc.

◆ There are two libraries:
  - The Unified Library contains "ready made" components with non-variable function and size
  - The LogiBLOX Library contains templates which can be customized for function and size

◆ Both libraries allow easy design migration across Xilinx devices and families

Unified Library

◆ The catalog of design elements is known as the "Unified Libraries."

◆ Elements in these libraries are common to all Xilinx device architectures.

◆ Xilinx maintains software libraries with thousands of functional design elements (primitives and macros) for different device architectures.

◆ Schematic design is based on “Unified Library”
  - XC3000A, XC3000L, XC3100A, XC3100L, XC4000E, XC4000L, XC4000EX, XC4000XL, XC4000XV, XC400XL, XC5200, XC9500, XC9500XL, Spartan, SpartanXL, and Virtex
  - A project must have only one “Unified Library”
LogiBLOX Overview

◆ LogiBLOX is a graphical interactive design tool
  • You can use to create high-level modules such as counters, shift registers, and multiplexers.
  • LogiBLOX includes both a library of generic modules and a set of tools for customizing them.

◆ LogiBLOX graphical user interface (GUI) can create and process high-level LogiBLOX modules
  • They can fit into your schematic-based design or HDL synthesis-based design.

◆ LogiBLOX can be used in schematic editors and third-party synthesis tools
  • Schematic: Aldec, Viewlogic, Mentor Graphics, and Cadence,
  • Synthesis: Synopsys FPGA Compiler, FPGA Express, and Exemplar Logic.

Starting LogiBLOX

◆ From the Project Manager
  • Tools > Design Entry > LogiBLOX module generator

◆ From Schematic Capture
  • Options > LogiBLOX

◆ From HDL Editor
  • Synthesis > LogiBLOX
LogiBLOX templates and GUI

- LogiBLOX is composed of two parts:
  - LogiBLOX Library containing templates of **VARIABLE SIZE**
    - Templates are expanded or customized (Counters, Adders, Registers, RAM, ROM)
    - Templates have many implementations (e.g. Binary, Johnson, LFSR counters)
  - LogiBLOX GUI and Synthesizer to create
    - A design file for implementation
    - Symbol for schematic capture tool
    - HDL code for instantiation in your design
    - Functional simulation model

LogiBLOX GUI
Generic LogiBLOX Functions

◆ One generic model per function type (ex: counter) - Attributes can be specified
  • ex: bus width, load, clock enable, etc.

◆ Arithmetic:
  • Counter, Adder/Subtractor, Accumulator, Comparator

◆ Storage:
  • Shift Register, Data Register, Memory (ROM, RAM, SYNC_RAM, DP_RAM)

◆ Logic:
  • Multiplexer, Decoder, Tristate Buffers, Simple gates

◆ I/O:
  • Input/Output (schematic only), Pad (schematic only)

◆ Others:
  • Clock Divider, Constant

Example: LogiBLOX Counter Template

◆ The Counter template has a number of options:
  • Example of COUNTER styles:
    – Binary, Johnson, LFSR, One-HOT
  • Up, down, or bi-directional
  • Fixed pre-load and/or parallel load
  • Asynchronous or Synchronous starting value
  • Clock enable
  • Counter range
Using LogiBLOX in Schematics

◆ Invoke the Module Selector from within your design entry tool
◆ Configure your project directory using the LogiBLOX Setup window
◆ Select a base module type (ex: Counter, Memory, or Shift-register)
◆ Customize the module: select bus size and function (ex: Johnson Counter)
◆ Select OK to create a schematic symbol and a simulation model for the selected module
◆ Use the LogiBLOX module in the design the same as any other component.
  • You can simulate in functional or timing mode

Learn More LogiBLOX

◆ Import an existing LogiBLOX module from another directory or project into the current project library
  • Options > Import LogiBLOX from the Schematic Capture window
  • And, choose the .MOD file of the module you want to import
◆ Documentation Viewer
  • LogiBLOX Reference/User Guide
Choosing Project Libraries

To add or remove project libraries for the current project:

2. To add a library to the project, select it in the Attached Libraries box and click Add>>.
3. To remove a library from the project, select it in the Project Libraries box and click <<Remove.
4. Click Close to complete the operation.
Foundation Schematic Design

Schematic Editor Overview

- Support for multiple sheet, flat and hierarchical schematics.
- Integration with Logic Simulator
- Integration with non-schematic design entry tools (HDL Editor and State Diagram Editor), providing for the use of non-schematic macros.
- Import of Viewlogic schematics.
- Schematic netlist exported to XNF, EDIF and VHDL formats.
Schematic Editor Window

Select and Drag
Hierarchy Push/Pop
Symbol toolbox
Draw wires
Draw bus taps
Add Net or Bus name
I/O Terminal
Power Symbol
Graphics toolbox

Design Structure

◆ Large schematics usually cannot be fitted into a single schematic sheet.
◆ A large schematic must be divided into smaller portions convenient to edit and simulate.
  • Single sheet designs
  • Multi-sheet designs
  • Hierarchical designs
Hierarchical Schematic

**Advantages of hierarchical designs:**
- The symbols in a hierarchical schematic library can represent large functional blocks.
- Top-down or bottom-up methodology assists in team development by defining design sections for each designer.
- You can use multiple instances of the same macro.
- Macros can be used in multiple projects.

**Disadvantages of hierarchical designs:**
- Netlist names can become very long.
- Updating macros often requires changing their symbols.

Adding and Removing Top-Level Schematic Documents I

**To be a part of a project, a top-level schematic file must be attached to the project.**

**To create a new project schematic document**
- In Schematic Editor choose *New Sheet* from the *File* menu, or click the New Schematic button in the main toolbar.

**To create a new non-project schematic sheet**
- Choose *Scratchpad* from the *File* menu.

**To add an existing top-level schematic to a project, use Project Manager:**
- 1. Choose *Add* from the *Document* menu. The Add Document dialog box will open.
- 2. Find and select the desired schematic file. Then click *OK* to complete the operation.
Adding and Removing Top-Level Schematic Documents II

To remove a top-level schematic file from a project, use Project Manager:

1. Select the file in the hierarchy browser.
2. Choose Remove from the Document menu, or press the Del key. Project Manager will prompt you to confirm the operation.

Opening Schematic Documents

In Schematic Editor you can open the following schematic documents:

- top-level sheets
- non-project sheets
- macro sheets

To open a schematic document:

1. Choose the Open command from the File menu. The Open Sheet dialog box will open.
2. Select the desired document.
3. Click OK to complete the operation.
**Bottom-Up Methodology I**

- **Make sure that the necessary libraries have been assigned to the project.**
  - You can view the currently attached project libraries in the Files tab of the Project Manager.
  - To add additional libraries, select *File > Project Libraries* from the Project Manager.

- **Enter your schematic design in Schematic Capture**
  - Each macro is a self-enclosed entity. Any connection to the top-level sheet can only be performed through hierarchy connectors.

- **The hierarchy connectors must be specified explicitly as Input, Output, or Bidirectional.**
  - This specification is important because the design entry tools automatically generate a symbol.
  - The location of the pins on the symbol depends upon their schematic I/O definition (only inputs are on the left-side of the symbol outline). If needed, edit this symbol in the Symbol Editor.

**Bottom-Up Methodology II**

[Diagram of a schematic design with labeled inputs and outputs, showing connections between various components.]
Bottom-Up Methodology III

◆ Create the macro symbol
  • Select Hierarchy > Create Macro Symbol from Current Sheet from the Schematic Capture window. The new symbol is automatically placed in the current project’s working library.

Top-Down Methodology I

◆ To implement a top-down design, you first create a symbol for the hierarchical macro and then create the underlying schematic.
  1. To create an empty symbol, select the Hierarchy > New Symbol Wizard from Schematic Capture.
  2. Click Next.
  3. In the Design Wizard - Contents dialog box, choose Schematic in the Contents section. Enter the Symbol Name and then select Next.
  4. In the Design Wizard - Ports dialog box, select New.
  5. Enter all ports including bus pins and power supply pins, if any. Select Next.
  6. In the Design Wizard - Attributes, enter a reference for the new symbol. Select Next.
  7. Click Finish in the Design Wizard - Contents window.
  8. Place the symbol on a schematic sheet and make the required connections.
  9. Push down into the symbol by clicking the Hierarchy Push/Pop function and double clicking on the macro symbol.
Top-Down Methodology II

10. An empty schematic sheet appears with the selected symbols' input pins located to the left and the output pins located to the right.

11. Enter the design and then select *File > Save*.
Starting a New Schematic

- By clicking the Schematic Editor button in the Design Entry area of the Flow tab
- By choosing Schematic Editor from the Design Entry submenu of the Tools menu.
  - From Project Manager, Tools > Design Entry > Schematic
- By double-clicking in the hierarchy tree the name of the schematic document you wish to open.

Placing Symbols

- Switch to the Symbols mode and open the SC Symbols toolbox
  - Choose Symbols from the Mode menu or click the Symbol toolbox button in the Schematic toolbar.
- Placing Symbols
  - 1. Select the desired symbol from the list in the toolbox.
  - 2. Move the mouse pointer over the desired location on the schematic.
  - 3. If you need to rotate or mirror the symbol before placement, do one of the following:
    - Press Ctrl + R to rotate the symbol by 90 degrees right (clock-wise).
    - Press Ctrl + L to rotate the symbol by 90 degrees left.
    - Press Ctrl + M to mirror the symbol.
  - 4. Click the mouse button to place the symbol.
Deleting, Moving and Replacing Symbols

◆ Deleting Symbols
  • To delete a symbol, select it with the mouse button and press the Del key or Edit > Delete

◆ Moving Symbols
  • 1. Move the mouse pointer over the symbol you wish to move.
  • 2. Press the mouse button, and holding it move the symbol to a new location.
  • 3. Release the mouse button to place the symbol at the new location.

◆ Replacing symbols
  • 1. Select the symbol.
  • 2. Options > Replace Symbol
  • 3. Select the button : all or selected
  • 4. Fill the symbol name in "with symbol"

Placing Wire Terminals

◆ Placing Wire Terminals
  • 1. Click the I/O Terminal button in the Schematic toolbar.
  • 2. Type its name in the Terminal Name box.
  • 3. Select the type of the terminal from the Terminal Type list box (INPUT, OUTPUT, BIDIR)
  • 3. Click the OK button to close the dialog. The mouse pointer will adopt a new shape.
  • 4. Move the mouse pointer over the desired location and click the mouse button to place the terminal symbol.
  • 5. If you want to place several terminal symbol whose name consist of a fixed prefix and consecutive numbers, click the Repeat button instead of OK.
Drawing Wires I

◆ **Switch to the Draw Wires mode**
  - Choose Draw Wires from the Mode menu or click the Draw Wires button in the Schematic toolbar.
  - In this mode the mouse looks as

◆ **Draw a wire**
  - 1. To Start a wire, Click a pin or terminal or existing wire.
  - 2. To anchor a corner, click the mouse button.
  - 3. To cancel the connection by pressing the Esc key, thus switching back to the Drag and Select mode.
  - 4. To complete the connection, move the mouse pointer over another pin, terminal or wire, and then click the mouse button.

Drawing Wires II

◆ **Naming Wires**
  - 1. Switch to the Select and Drag mode
  - 2. Double click the wire you want to name, the Net Name window will open.
  - 3. Type the desired name in the Net Name box, and then click OK.

◆ **Renaming Wires**
  - 1. Double-click an appropriate wire. The Net Name dialog box will open.
  - 2. Edit the name in the Net Name box, and then click OK. All labels attached to the net will change to the new name.
Drawing Wires III

◆ Renaming Wires with Terminals
  - 1. Double-click an appropriate wire. The I/O Terminal dialog box will open.
  - 2. Edit the name in the Terminal Name box, and the click OK.

◆ Moving Names
  - 1. Switch to the Select and Drag mode.
  - 2. Select it and drag to the desired location.

◆ Moving Wires
  - 1. Switch to the Select and Drag mode.
  - 2. Move the mouse pointer over the wire segment you wish to move.
  - 3. Press the mouse button and holding it, move the wire segment to a new location.
  - 4. Release the mouse button to place the wire segment at the new location.

Using Buses I

◆ Naming Conventions
  - A generic bus name consists of an identifier followed by index bounds enclosed in square brackets:

  $$BUS\_NAME[X:Y]$$

  - X and Y are integer numbers greater or equal 0. There can be both X>Y and X<Y. If
    the both index bounds can be equal (X=Y), the bus consists of a single discrete signal.
  - Data[3:0] = (Data3, Data2, Data1, Data0)
  - Data[0:3] = (Data0, Data1, Data2, Data3)
  - Data[2:2] = Data2

◆ Bus Pins

![](image)
Using Buses II

- **Bus Taps**
- **Bus Terminals**
  - Input
  - Output
  - BiDirectional
  - Unspecified

**Connectivity Rules**

- DT7=A7 EA15=B7 XOUT8=S7
- DT6=A6 EA14=B6 XOUT9=S6
- DT5=A5 EA13=B5 XOUT10=S5
- DT4=A4 EA12=B4 XOUT11=S4
- DT3=A3 EA11=B3
- DT2=A2 EA10=B2
- DT1=A1 EA9=B1
- DT0=A0 EA8=B0

Using Buses III

- **Unnamed Buses**
  - AOUT3=BIN7=CIN15
  - AOUT2=BIN6=CIN14
  - AOUT1=BIN5=CIN13
  - AOUT0=BIN4=CIN12
  - BIN3=CIN11
  - BIN2=CIN10
  - BIN1=CIN9
  - BIN0=CIN8
  - OUX3=D7
  - OUX2=D6
  - OUX1=D5
  - OUX0=D4
Using Buses IV

◆ Editing Bus Connections
   - Double-clicking a bus pin opens the Bus Pin Connections dialog box, which allows you to enter the exact sequence of signals connected to the bus pin.

   ![Bus Pin Connections dialog box]

   - XA7=A3
   - XA6=A2
   - XA5=A1
   - XA4=A0
   - XA3=QA
   - XA2=QB
   - XA1=QC
   - XA0=A7

Using Buses V

◆ Drawing Buses
   - Switch to the Draw Buses mode, choose Draw Buses from the Mode menu or click the Draw Buses button in the Schematic toolbar.
   - The mouse pointer in the Draw Buses mode looks as ⬇️
   - To connect a bus pin or bus terminal with another bus pin or bus terminal:
     - 1. Click the first bus pin or bus terminal.
     - 2. To complete the connection, move the mouse pointer over another bus pin or bus terminal, and then click the mouse button.
     - 3. Double click the bus segment, and then fill the bus name and range.
Using Buses VI

◆ Adding Bus Terminals
  
  - To add a bus terminal to an existing bus segment:
    1. In the Drag and Drop mode, double-click the bus segment close to its end where you wish to add a terminal. The Add Bus Terminal /Label dialog box will open.
    2. Specify the name of the bus segment, index bounds and the type of the terminal (I/O marker).
    3. Click OK to complete the operation.
  
  - To start drawing a bus segment, and end it with a bus terminal:
    1. Click with the right mouse button to detach the temporal bus line from the mouse pointer.
    2. Click the I/O Terminal button in the Schematic toolbar. The Add Bus Terminal /Label dialog box will open.
    3. Specify the name of the bus segment, index bounds and the type of the terminal (I/O marker).
    4. Click OK to complete the operation.

Using Buses VII

◆ Automatic Drawing Bus Taps
  
  - This mechanism ensures that connections between the bus and symbol pins are automatically drawn and labeled with the names of consecutive bus members.
  
  - To automatically draw connections between a bus and discrete pins of a symbol:
    1. Click the Draw Bus Tap button in the Schematic toolbar, or choose the same command from the Mode menu.
    2. Click the bus name to which you want to draw connections.
    3. Click the symbol pin.
      - A wire will be automatically drawn between the pin and bus.
      - After this operation, the index in the name of the current bus member will be incremented by 1.
    4. Click remaining pins in the desired sequence.
    5. Press the Esc key to return to the Select and Drag mode.
Checking Connections

◆ SC Query /Find window that allows you to unambiguously check connections on schematics
  - Choose Query from the Edit menu or click the Query Window button.

◆ Querying Connections
  - Find out what symbols and pins are connected by a given wire or bus net
    - select the net with the mouse.
  - Find out what nets are connected to a given component symbol
    - select the symbol with the mouse.

◆ Searching for Schematics Items
  - Specify the type of the schematic item you want to find
  - Specify what schematic sheets should be searched.
  - Type the item name in the Search Box, and then press the Enter key to launch the search.

Zooming Schematics

◆ Zoom in or out a schematic
Navigation through Project Hierarchy
(in Schematic Capture)

◆ **Switch to Hierarchy mode**
  - Choose Hierarchy Push from the Hierarchy menu or click the Hierarchy Push /Pop button in the Schematic toolbar.
  - The mouse pointer adopts a different shape, shown as $\text{H}$.

◆ **To leave the Hierarchy mode**
  - Choose again Hierarchy Push from the Hierarchy menu, click the Hierarchy Push /Pop button in the Schematic toolbar or simply press the Esc key.

◆ **To open a macro’s schematic, use $\text{H}$ to double-click the macro symbol.**

◆ **To leave the macro schematic and go back to the upper hierarchy level**
  - Choose Hierarchy Pop from the Hierarchy menu, or double click over empty space of the macro schematic.

Creating Schematic Netlist

◆ **A netlist is a description of connections between schematic components.**

◆ **A netlist is generated from project schematics and saved to a binary file with the ALB extension.**

◆ **The netlist is generated automatically when you start functional simulation.**

◆ **To create a project netlist, Options > Create Netlist**

◆ **To creates the ALB binary netlist from current macro schematic sheet, Options > Create Netlist from Current Sheet**
Exporting Netlist

◆ Schematic Editor supports conversion of the binary ALB netlist into a selected text format

◆ To export the project netlist into a selected text format:
  1. Choose Export Netlist from the Options menu. The Export Netlist dialog box will open.
  2. From the Netlist Format list box, select the desired format. You can choose XNF, EDIF200 or VHDL.
  3. If you want to use a specific extension for the exported netlist file, use the Options button.
  4. Choose the source netlist ALB file. By default, the project netlist is automatically selected.
  5. Click OK to start exporting.

Create Macro Symbols from Netlists

◆ Schematic Editor allows you to import an external netlist (edif, xnf) and save it to be a symbol. It behaves like a schematic primitive.

◆ To import an external netlist:
  1. Choose the Create Macro Symbol from Netlist command from the Hierarchy menu. The dialog box will open.
  2. From the Netlist Format list box, select the desired input netlist format. You can choose either XNF or EDIF.
  3. Choose the source netlist file.
  4. Click OK to start importing.
Generating Schematics from Netlists

- You can generate a schematic from an existing netlist of type ALB or EDIF.
- To generate a schematic from a netlist:
  1. Choose Generate Schematic from Netlist from the File menu. The Generating Schematic dialog box will open.
  2. Select the desired netlist type from the List files of type list box, then the desired netlist file.
  3. Click the Options button to display the Page Setup dialog box which allows you to select the desired page size and orientation.

Importing Viewlogic Schematics

- Schematic Editor supports import of flat and hierarchical schematics from Viewlogic designs.
- To import a Viewlogic schematic:
  1. Choose File > Import Viewlogic Schematic. Then Select Viewlogic Schematics window will open.
  2. Select the desired schematic file(s) and click the Options button. The Options dialog box will open.
  3. Select the desired options and click OK.
  4. Click OK to start the schematic import.
Integrity Test

◆ The integrity test performs a comprehensive analysis of the project netlist. It does not create a netlist but detects all design errors and inconsistencies.

- Bus taps without labels. Such taps behave like hanging wires.
- Duplicated hidden net names.
  - Hidden net names are generated automatically when you are drawing wires.
  - If two wires have the same hidden names, they make up a single net. Such situation is usually unintended and the user is unable to detect it by himself.
- Hanging wires ended with bubbles at both ends.
- Hanging wire touching a symbol pin.
- Two names with different bound indexes assigned to the same bus.
- Discrete wires connected to a bus (via bus taps) that are not members of this bus.
- Macro names identical to the project name.

◆ To run the integrity test, choose Integrity Test from the Options menu.

Symbol Editor Overview I

◆ Symbol Editor is an application designed for creating and editing symbols. It allows you to edit shape, pins, pin descriptions and numbers for the component symbol.

◆ You can start Symbol Editor in one of the following ways:

- From Project Manager:
  - Tools > Design Entry > Symbol Editor
- From Schematic Editor:
  - Select the component symbol you wish to edit, and then choose Symbol Editor from the Options menu or press Ctrl+E.
- From Library Manager:
  - On the Objects tab, double-click the name of the component whose symbol you wish to edit.
Symbol Editor Window II

To specify symbol attributes select the desired symbol in the Select mode and click on the Properties icon.

Double click the symbol.
Foundation HDL Editor
&
State Editor

Features of Foundation HDL Entry

- Support three languages: ABEL, VHDL, Verilog
- Provide very useful instrument, Language Assistant
- Syntax correctness of the code created within the HDL Editor
- Has interface to synthesis tools which generate XNF netlist from HDL source code

How to run HDL Entry

- Tool > Design Entry > HDL Editor
- Click the HDL Editor button in the project flowchart
Create a New File

- Click ‘use HDL Design Wizard’ or ‘create empty’
- Use ‘Design Wizard’
- Use Language Assistant to Edit

HDL Editor Window

- Library
- Entity
- Architecture
The Language Assistant is a tool helping in VHDL, Verilog and ABEL source code entry. It provides a number of templates with prepared pieces of code. There are two groups of templates:

- language templates with basic language constructs,
- synthesis templates with synthesis-oriented implementation of basic functional blocks, such as multiplexers, flip-flops, counters, etc.

Synthesize HDL Flow

- Configuration
  - Synthesis -> Configuration
- Check Syntax
  - Synthesis -> Check Syntax
- Choose Options
  - Synthesis -> Options
- Synthesize & Create Macro
  - Project -> Create Macro
Synthesize ABEL

◆ Synthesis Options
  • Chip - enables insertion of IBUF and OBUF pads; used for HDL type projects.
  • Macro - disables insertion of IBUF and OBUF pads; used for schematic type projects.

ABEL Details

◆ To know about ABEL
  • Refer to DATA I/O’s XILINX-ABEL manual
  • Foundation HDL Entry - Language Assistant
  • Xilinx Foundation Series On-Line Help System

◆ To make writing ABEL code easy
  • Use Foundation HDL Entry Editor to edit your ABEL files
Synthesize VHDL/Verilog I

**General**

- **Compile** - Controls automatic insertion of IBUF and OBUF pads.
  - Chip - Enables insertion of IBUF and OBUF pads. This option should be selected if you synthesize a top-level HDL document.
  - **Macro** - Disables insertion of IBUF and OBUF pads. This option should be selected if you synthesize an HDL macro.
- **Optimize for** - Controls whether optimization is to be done for speed or smaller area.
  - Speed - Specifies that optimization for speed is to be done.
  - Area - Specifies that optimization for smaller area is to be done.

Synthesize VHDL/Verilog II

- **Effort** - Controls the mapping effort for the design, affecting timing optimization most.
  - High - Takes longer to compile but should produce better designs as the mapping process proceeds until all strategies are tried.
  - Low - Takes the least time to compile. Use Low if you run a test to check logic. Not recommended if the design must meet area or timing requirements.

**Advanced**

- The Advanced tab of the FPGA Synthesis Options dialog allows you to select the module (Verilog specific) or entity/architecture pair to be the top-level of the synthesized HDL macro.
- Module (FPGA Express synthesis option for Verilog source code)
  - Selects the top-level Verilog module.
Synthesize VHDL/Verilog III

- **Entity** (FPGA Express synthesis option for VHDL source code)
  - Selects the top-level entity.
- **Architecture** (FPGA Express synthesis option for VHDL source code)
  - Selects the architecture of the top-level entity.

**Library Alias**

- The Library Alias tab allows you to define the library and add a number of HDL files to it.
- The files should exist in the project's hierarchy tree for the option to have effect, which means they are synthesized to the defined library.
- The library alias feature is global to the project.

Synthesize VHDL/Verilog IV

**FSM**

- **Default Encoding**
  - **One Hot** - Selects the state machines states 'one hot' encoding
  - **Binary** - Selects the state machines states binary encoding
- **Interpretation of VHDL 'when others'**
  - **Fastest and smallest (only defined states)** - Selects the interpretation of the VHDL 'when others' as only defined states of the state machine. This is the fastest interpretation.
  - **Safest (all possible, including illegal, states)** - Selects the interpretation of the VHDL 'when others' as all possible states, also illegal ones. This is the safest interpretation.
State Editor Overview

- State Editor is a tool designed for graphical editing of state machine diagrams.
- You can freely mix HDL code with state diagrams.
- State diagram cannot instantiate lower-hierarchy level design entities.
- State Editor allows you to partition the logic of a complex sequential block into several concurrent machines.

State Diagram Elements

[Diagram showing state machine elements: diagram actions, latch, variable/signal, state, condition, transition, action, machine name, reset, etc.]
State Editor Window

Open State Editor

- Open the State Editor
  - Tools -> Design Entry -> State Editor... in the Projects Manager
  - Click the FSM Editor button in the project flowchart
Using HDL Design Wizard

This wizard will help you to create your new design quickly and easily. You will be able to specify basic features of your project and to enter ports.

To begin creating the design, click "Next."

In your design, an HDL language will be used. Now you can choose your preferred language:
- ABEL
- VHDL
- Verilog

Choose the name of the file in which your design will be saved.

To create a new port, click "New." To change attributes of a port, select it on the list. Then you can change its name, range, and direction. To set other attributes, click Advanced.

To delete a port, select it on the list and click "Delete."

The diagram can include one or more machines. Please select the number of machines you want:
- One
- Two
- Three
- Four
- Many

Page Setup
Edit a Diagram I

◆ To add a port to a diagram
  1. Choose Input Port, Output Port or Bi-directional Port command from the FSM menu.
  2. Place the port on the current diagram.

◆ To add a signal to a diagram
  1. Choose the Signal command from the FSM menu.
  2. Place the signal icon in the desired place on a diagram.

Edit a Diagram II

◆ To add a state to a machine
  1. Choose the State command from the FSM menu.
  2. Place the state bubble on the current diagram (within machine's frame).
Edit a Diagram III

◆ To add a reset to the diagram
  1. Choose the Reset command from the FSM menu.
  2. Place the reset icon in the desired place within machine frames.

◆ To draw a transition between states
  Transitions connect states and describe the sequence of states. Transitions are also used for connections with the reset and entry/exit (for hierarchical states).
  1. Choose the Transition command from the FSM menu.
  2. Click over the state where the transition begins.
  3. Click over the state where the transition ends.
  4. Click over a diagram to end the operation.

Edit a Diagram IV

◆ To add a condition to a transition
  Condition is a Boolean HDL expression associated with a transition. If it is true, the machine goes from one state to another.
  1. Choose the Condition command from the FSM menu.
  2. Click on the transition you wish to add condition for.
  3. Enter the condition's text.
  4. End conditions editing by clicking the left mouse button.
Edit a Diagram V

◆ To add an action to a state

- Action is a set of HDL statements, which assigns new values to ports or internal signals/variables.
- 1. Choose the Action command from the FSM menu. It displays a submenu.
- 2. Select either Entry, State or Exit, depending on the type of action you want to add. The mouse pointer changes into an action line, with a black dot at the beginning.
- 3. Click with the black dot end over the desired state.
- 4. Enter the action’s text.
- 5. End actions editing by clicking the left mouse button.

Entry Action is executed when a machine is entering the state with which the action is associated.
State Action is executed when a machine remains in the state with which the action is associated.
Exit Action is executed when a machine is leaving the state with which the action is associated.
Transition Action is executed when a machine is going through the transition with which the action is associated.
Diagram Action is executed concurrently with the machines.

Edit a Diagram VI

◆ To choose the clock of a machine

- 1. Choose the Machines command from the FSM menu.
- 2. Select the machine from the displayed submenu. It displays the Machine Properties dialog box.
- 3. Select the desired clock from the list of clock ports.
- 4. Click OK.
Machines

![Diagram of machines with states and transitions]

Synthesize & Create Macro

- **Synthesize (the same with HDE Editor)**
  - Synthesis -> Configuration ...
  - Check Codes
    - Synthesis -> HDL Code Generation
  - Generate a Chip
    - Synthesis -> Options ... > Chip
    - Synthesis -> Synthesize
  - Create a macro
    - Synthesis -> Options ... > Macro
    - Project -> Create macro
16-bit Adder Examples

Many choices for implementing an adder
- Speed vs. density trade-off controlled by user and PLD features

<table>
<thead>
<tr>
<th>Family</th>
<th>Type</th>
<th>CLBs</th>
<th>Levels</th>
<th>AppLINX</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3000A</td>
<td>Bit-Serial</td>
<td>16</td>
<td>16</td>
<td>XAPP 022</td>
</tr>
<tr>
<td>XC3000A</td>
<td>Parallel</td>
<td>24</td>
<td>8</td>
<td>XAPP 022</td>
</tr>
<tr>
<td>XC3000A</td>
<td>Lookahead</td>
<td>30</td>
<td>6</td>
<td>XAPP 022</td>
</tr>
<tr>
<td>XC3000A</td>
<td>Conditional</td>
<td>41</td>
<td>3</td>
<td>XAPP 022</td>
</tr>
<tr>
<td>XC4000E-3</td>
<td>Carry</td>
<td>8</td>
<td>10.1ns</td>
<td>XAPP 018</td>
</tr>
<tr>
<td>XC5200-5</td>
<td>Carry</td>
<td>8</td>
<td>20ns</td>
<td>5200 DataSheet</td>
</tr>
</tbody>
</table>
Arithmetic Functions

- **Arithmetic Macros are optimized for density and speed**
  - Example: Each CLB can form a two-bit full-adder

- **Most Arithmetic macros have been pre-placed**
  - Require Vertical Orientation to maintain speed and density
  - Known as RPM or “Relationally Placed Macro”
  - Examples:
    - ADDx adders
    - ADSUx adder/subtractors
    - CCx counters
    - COMPXCx magnitude comparators

```
A<3>  B<3>  Z<3>
A<2>  B<2>  Z<2>
A<1>  B<1>  Z<1>
A<0>  B<0>  Z<0>
```

Three-State Buffers

- **Each CLB is associated with two Three-State buffers (BUFT)**
  - BUFTs are used independently of LUTs and Flip-Flops

- **Three-State library components:**
  - Three-state buffers: BUFT, BUFT4, BUFT8, BUFT16
  - Wired AND (Open Drain): WAND1, WAND4, WAND8, WAND16
    - XC4000 only
    - Must use a PULLUP primitive
  - 2 Input OR Gate with Wired-AND Open-Drain Buffer Output: WOR2AND
    - XC4000X only
    - Must use a PULLUP primitive

- **Delay varies per family**
  - 2.9 ns in the XC4003E (-1)
  - 5.6 ns in the XC4020E (-1)
Use BUFT for Buses

- Use to multiplex signals onto long routing lines to use as buses

BUFTs for Multiplexers

- BUFT can be used to build large MUXes
  - Large MUXes composed of LUTs need multiple levels of logic
  - Large MUXes composed of BUFTs have only one level of logic
    - CLB resources are not used
  - Use of BUFTs constrains placement
- Multiplexer macros use lookup tables
  - Example: M4_1E
- Create BUFT macros from Three-State buffer components
  - BUFT, BUFT4, BUFT8, BUFT16
Wide Decoders

◆ The Wide Decoder is a dedicated wired-AND
  • Useful for address decoding
◆ IOBs or CLBs can drive the Wide Decoder
  • Located along the periphery of the die
  • All IOB drivers must be on same edge as the decoder
  • Four decoder lines per edge
◆ Use DECODE macro
  • XC4000 only
  • DECODE4/8/16/24
  • Must use a PULLUP primitive

CLB Mapping Control in Schematic

◆ Allows user to force mapping of logic from schematic into a single CLB
◆ XC3000
  • CLBMap can specify entire CLB
◆ XC4000/Spartan/XC5000
  • FMap specifies a function generator in a CLB
  • HMap specifies an XC4000/Spartan H function generator in a CLB
Advanced Hardware Design Techniques

Synchronous Logic
(Flip-Flops and Latches)

Library Offerings

◆ Types of Register Functions
  • Shift Registers
    – Left/Right, Arithmetic, Logical, Circular
  • Clock Dividers
    – Output Duty Cycle
  • Counters
    – LFSR, Binary, One_Hot, Carry Logic
  • Accumulators
Naming Conventions

Flip-Flop
- D-Type (D), JK-Type (JK), Toggle-Type (T)
- Asynchronous Preset (P), Asynchronous Clear (C)
- Synchronous Set (S), Synchronous Reset (R)
- Clock Enable
- Inverted Clock

Transparent D Latch
- Asynchronous Preset (P), Asynchronous Clear (C)
- Gate Enable
- Inverted Gate

Counters

◆ Libraries support a wide variety of fast and efficient counters
  - Counters offer trade-offs between *speed*, *density*, and *complexity*
  - Example: LogiBlox counter styles
    - Binary: predictable outputs, uses carry logic
    - Johnson: fastest practical counter, but uses more flip-flops; glitch free decoding
    - LFSR: fast & dense, but pseudo-random outputs
    - One-Hot: useful for generating series of enables
    - Carry Chain: High speed and density
  - The LogiBlox synthesizer will automatically pick the best implementation based on your design, or you can force an implementation with the STYLE parameter (schematic).
16 Bit Counter Examples

◆ The following are implemented in XC4000XL-3

<table>
<thead>
<tr>
<th>Macro</th>
<th>CLBs</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB16CLE/D</td>
<td>18 - 20</td>
<td>23 - 24 ns</td>
</tr>
<tr>
<td>CC16CLED</td>
<td>19</td>
<td>19 ns</td>
</tr>
<tr>
<td>CC16CLE</td>
<td>9</td>
<td>16 ns</td>
</tr>
<tr>
<td>LogiBLOX: LFSR</td>
<td>9</td>
<td>7 ns</td>
</tr>
</tbody>
</table>

◆ Simpler functions are faster and smaller
◆ Carry Logic Counters are generally faster (depends on size)
◆ App Notes 001 and 014 describe fast PRE-scaled counters

Global Clock Buffers

◆ Clock Buffers are low-skew, high drive buffers
  • Also known as Global Buffers
  • Drive low-skew, high-speed long line resources
  • Drive all Flip-Flops and Latches in FPGA
  • Can also be used for high-fanout signals

◆ Additional clocks and high fanout signals can be routed on long lines
  • Otherwise routed on general interconnect
    – Slower and higher skew
Using a Clock Generated Off-Chip

◆ Connect IPAD directly to clock buffer primitive
  - Required for BUFG
◆ Place & route uses special fast input pin
◆ Provides higher speed and uses fewer routing resources

Use Global Clock Buffers

◆ Use clock buffers for highest fanout clocks
  - Drive low-skew, high-speed long line resources
  - Use BUFG primitive to be family-independent
◆ Limit number of clocks to ease placement issues
  - XC3000: 2
  - XC4000: 8
  - Spartan: 8
  - Virtex: 28
  - XC5000: 4
**XC3000 Clock Buffers**

- 2 global clocks per device
  - 1 primary (GCLK), 1 secondary (ACLK)
- Connects to clock pins only
- Can be driven by a special IOB or internal logic
- Use GCLK first

![Diagram of XC3000 Clock Buffers](image)

**XC4000E/Spartan Global Buffers**

- Eight global buffers per FPGA
  - Four primary (BUFGP), Four secondary (BUFGS)
- Primary buffers must be driven by a semi-dedicated IOB
- Secondary buffers can be driven by a semi-dedicated IOB or internal logic and have more routing flexibility
  - Use BUFGS if extra 1-2ns of delay is acceptable
- Use generic BUFG primitive in your design
  - Allows software to choose best type of buffer
  - Allows easy migration across families

![Diagram of XC4000E/Spartan Global Buffers](image)
XC4000X Global Buffers

◆ Global Low-Skew Buffers (BUFGLS) (8)
  • Standard clock buffers
  • To be used for most internal clocking, whenever a large portion of the device must be driven

◆ Global Early Buffers (BUFGE) (8)
  • A faster clock access
  • CLB access is limited to one-fourth of the device

◆ BUFGLS and BUFGE share a single pad
  • The same IPAD symbol can drive one buffer of each type, in parallel

◆ Use generic BUFG primitive in your design

Global Low-Skew Buffers

◆ Any BUFGLS (GCK1 - GCK8) can drive any or all clock inputs on the device
Global Early Buffers

- Left and right BUFGEs can drive any or all clock inputs in the same quadrant or edge (GCLK1 is shown. GCK2, GCK5, and GCK6 are similar.)

- Top and bottom BUFGEs can drive any or all clock inputs in the same quadrant. (GCLK8 is shown. GCK3, GCK4, and GCK7 are similar.)

XC5000 Global Buffers

- 4 global buffers per device
  - All BUFG
- Main function is for clocks
  - Limited routing if used for non-clock pins
- Can be driven by a special IOB or internal logic
Generating Clock On-Chip - XC3000

**XC3000**
- Internal amplifier for external crystal
- Use OSC or GXTL primitive and enable in Configuration Template

![XC3000 Diagram]

Generating Clock On-Chip - XC4000/Spartan

**XC4000/Spartan**
- Internal configuration clock available after configuration
- Use OSC4 primitive
- Nominal values (approximately):
  - 8 MHz, (500 kHz, 16 kHz, 490 Hz, 15 Hz)

![XC4000/Spartan Diagram]
Generating Clock On-Chip - XC5000

**XC5000**
- OSC5 (16MHz)
  - OSC5 is used where an internal oscillator is required
  - OSC5 primitive with OSC1 and OSC2 outputs
  - DIVIDE1\_BY = 4 / 16 / 64 / 256
  - DIVIDE2\_BY = 2 / 8 / 32 / 128 / 1024 / 4096 / 16384 / 65536
- CK\_DIV
  - CK\_DIV is applicable when a user clock input is specified.

Global Reset

**All flip-flops are initialized during power up via Global Set/Reset network**

**You can access Global Set/Reset network by instantiating the STARTUP primitive**
- Assert GSR for global set or reset
- GSR is automatically connected to all CLB flip-flops using dedicated routing resources
- Saves general use routing resources for your design
- DO NOT CONNECT GSR to set/reset inputs on Flip-Flops

**Any signal can source the global set/reset, but the source must be defined in the design**

**Use Global Reset as much as possible**
- Limit the number of flip-flops with an asynchronous reset
  - Extra routing resources are used
Avoid Gated-Clock or Asynchronous Reset

- Move gating to non-clock pin to prevent glitch from affecting logic
- Or separate input signal changes by at least a CLB delay (tILO) to minimize the likelihood of a glitch
- Use Clock Enables instead of gating clock

Shift Registers are Fast & Dense

- The CLB can handle two bits of a shift register
- Fast and dense independent of size
  - Fast connections between adjacent lookup tables
Use One-Hot Encoding for State Machines

- Shift register is always fast and dense
  - “One-hot” uses one flip-flop for each count
  - Useful for state machine encoding
- Another alternative is a Johnson Counter
  - Inverted output of last stage drives input of first stage
  - Doubles the number of states versus one-hot

![Diagram of Johnson Counter]

State Machine Design Tips

- Split complex states
- Need to minimize number of inputs, not number of flip-flops, in FPGAs
  - Use one-hot encoding for medium-size state machines (~8-16 states)
- Complex states may be improved by breaking up into additional simpler states
Use binary sequence only if necessary

- CLB can generate any sequence desired at the same speed
- Use Pre-Scaling on non-loadable counters to increase speed
  - LSBs toggle quickly
  - See Application Notes XAPP001 and XAPP014

- Use Gray code counters if decoding outputs
  - One bit changes per transition

- Consider Linear Feedback Shift Register for speed when terminal count is all that is needed
  - Or when any regular sequence is acceptable (e.g., FIFO)

Pipeline for Speed

- Register-rich FPGAs encourage pipelining
- Pipelining improves speed
  - Consider wherever latency is not an issue
  - Use for terminal counts, carry lookahead, etc.
- How to estimate the clock period
  - $2 \times \text{(number of combinatorial levels)} \times \text{(speed grade)}$
  - XC4000XL-3: 3 levels $\times 2 \times 3\text{ns} = 18\text{ ns}$ clock period
How to specify IO blocks - Schematic

- User explicitly defines what resources in the IOB are to be used
- I/Os are defined with
  - 1 pad primitive
  - At least 1 function primitive
    - 1 input element, 1 output element or both
    - Inverters may also be pulled into IOBs
- IOBs are named by net between pad and function primitives
Slew Rate Control

- Slew rate controls output speed
- Default slow slew rate reduces noise
- Use fast slew rate wherever speed is important
  - FAST parameter on output logic primitive

Use Pull-ups/Pull-downs to Prevent Floating

- Pull-up automatically connected on unused IOBs
- Outputs of unused IOBs are automatically disabled
- A PULLUP or PULLDOWN primitive can be specified on used IOBs
- Inputs should not be left floating
  - Add a pull-up to design inputs that may be left floating to reduce power and noise
Choose TTL or CMOS Thresholds

### XC4000 Configuration Options: Default

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Startup</th>
<th>Readback</th>
<th>Tie</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Rate:</td>
<td>Slow</td>
<td>Fast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold Levels (XC4000E and XC4000EX Only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputs:</td>
<td>TTL</td>
<td>CMOS</td>
<td>Read From Design</td>
<td></td>
</tr>
<tr>
<td>Outputs:</td>
<td>TTL</td>
<td>CMOS</td>
<td>Read From Design</td>
<td></td>
</tr>
</tbody>
</table>

- Configuration Pins:
  - TDO: Float, PullUp, FullDown
  - M0: Float, PullUp, FullDown
  - M1: Float, PullUp, FullDown
  - M2: Float, PullUp, FullDown
  - D0: Float, PullUp

- Perform CRC During Configuration
- Produce ASCII Configuration File
- SY4 Tolerant IOs (XC4000XL and XC4000V Only)

---

**I/O Logic**

- **4000E families have no boolean logic other than inverters in the IOBs**
- **XC4000X adds optional output logic**
  - Can be used as a generic two-input function generator or MUX
  - One input can be driven by IOB output clock signal
    - Driving from FastCLK buffer provides less than 6 ns pin-to-pin delay
  - Requires library components beginning with "O"

![Diagram](image-url)
**Fast Capture Latch (XC4000X)**

- Additional latch on input driven by output’s clock signal
- Allows capture of input by very fast clock
  - Followed by standard I/O storage element for synchronization to internal logic
  - Very fast setup (6.8 NS for 4000EX-3), 0 ns hold
  - Available on 4000X, not 4000E family

**Example**
- ILFFX or ILFLX macro includes Fast Capture Latch and IFDX
- Connect BUFGE to fast capture latch
- Opposite edge of same clock via BUFGLS drives IFDX

**Output MUX (XC4000X)**

- Fast output signal (from output clock pin) MUXes IOB output or clock enable pins to pad

- Effectively doubles the number of device outputs without requiring a larger, more expensive package
  - Pin-to-pin delay is less than 6 ns
Decrease Setup time with NODELAY

◆ NODELAY attribute
  - Removes delay element to the IFD or ILD
  - Decreases setup time, add creates hold time
  - Available on IFD/ILD macros in XC5200 and XC4000E/Spartan families

◆ XC4000X IOB - 2 Input Delay Elements
  - Full delay (default, no attribute added) : Zero Hold with respect to Global Low-Skew Buffer, Global Early Buffer
  - MEDDELAY : Zero Hold with respect to Global Early Buffer
  - NODELAY : Short Setup, positive Hold time

Output Three-State Control

◆ Free inverter on output buffer control
  - Use OBUFE macro for active-high enable
  - Use OBUFT primitive for active-low enable

◆ Three-state control also via a dedicated global net
  - Controlled by same STARTUP primitive

◆ All I/O disabled during configuration
Advanced Hardware Design Techniques

Memory Design
(RAM and ROM)
Only for XC4000 & Spartan

ROM is Equivalent to Logic

◆ When using ROM, it is simply defining logic functions in a look-up table format
  • Memory might be an easier way to define logic
  • Xilinx provides ROM library cells
◆ FPGA lookup tables are essentially blocks of ROM
  • Data is written during configuration
  • Data is read after configuration
    – Effectively operate as a ROM

As Gates

As ROM

DATA(0)=0
DATA(1)=0
DATA(2)=0
DATA(3)=1

\[ O = I_1 \cdot I_2 \]
RAM Provides 16X the Storage of Flip-Flops

◆ 32 bits versus 2 bits of storage
  • Two 16x1 RAMs or One 32x1 Single Port Ram fit in one CLB
  • One 16x1 Dual Port RAM fits in one CLB

◆ 32x8 shift register with RAM = 11 CLBs
  • Using flip-flops, takes 128 CLBs for data alone
  • Address decoders not included

RAM Types

◆ Synchronous RAM (SYNC_RAM)
  • Synchronous Write Operation

◆ Synchronous Dual-Port (DP_RAM)
  • Can read & write to different addresses simultaneously
RAM Guidelines

◆ Less than 32 words is best
  • 32x1 or 16x2 per RAM requires only one CLB
    – Delays are short, (one level of logic)
  • Data and output MUXes are required to expand depth
◆ Less than 256 words recommended per RAM
  • Use external memory for 256 words or more
◆ Width easily expanded
  • Connect the address lines to multiple blocks
◆ Recommendation: Use less than 1/2 of max memory resources
  • Maximum memory uses all logic resources of CLBs

Memory Use

◆ Most synthesis tools can synthesize ROM from behavioral HDL code, but RAMS must be instantiated
◆ Use library primitives and macros for standard size memory
  • RAM/ROM16X1S to 32X8S
  • Use S suffix for Synchronous RAM
  • Use D suffix for Dual-Port RAM
◆ Use LogiBlox to generate arbitrary size memories
How to Generate Memory

◆ Use LogiBlox utility to create arbitrary size RAM or ROM
  • Select type: ROM, Synchronous, Asynchronous, or Dual Port RAM
  • Specify Depth: number of words must be a multiple of 16, ranging from 16 to 256 words
  • Specify Width: word size ranges from 1 to 64 bits
  • Specify initialization values with attribute file

◆ LogiBLOX also creates RAM interface
  • Entity and component declaration - cut and paste into the design (VHDL designs)
  • Module declaration (Verilog designs)
  • Symbol Graphic (schematic entry designs)

Memory Generator Dialog

Specify memory type, size, name and function in the LogiBLOX GUI
Memory Definition File I

◆ Memory Definition File Header
  - Depth (optional)
    - depth `memory_depth`
  - Width (optional)
    - width `memory_depth`
  - Default
    - set the value of all memory locations that are not specified in the MEMFILE Data section
  - Radix
    - radix `integer`
  - Comments
    - ; The default radix is 10

Default 10; Defines the default ROM contents = 10_{10}=10
Radix 16; Re-defines the default radix = 16_{10}=16
Depth 10; Defines the depth = 10_{16}=16
Radix 10; Re-defines the default radix = 10_{16}=16
Width 12; Defines the width = 12_{16}=18

Memory Definition File II

◆ Memory Definition File Data Section
  - Data
    - data values
    - Data values may be separated by commas, white space, or both
  - Addressing
    - address :
    - Example
      - 8-word memory with the contents 6, 4, 5, 5, 2, 7, 5, 3, starting at address 0.
      - the contents of locations 2, 3, and 6 are defined via the Default definition
```
; memfile dec7seg.mem for LogiBLOX symbol dec7seg
; Created on Thursday, September 10, 1998 09:27:09
;
; Header Section
RADIX 10
DEPTH 16
WIDTH 7
DEFAULT 0
;
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
RADIX 2
DATA
1011000,
0000000,
0010000,
0001000,
0000011,
1000110,
0100001,
0000110,
0001110
; end of LogiBLOX memfile
```
Foundation Simulation

Start to Simulate

◆ From Project Manager
  - Press “Functional Simulation” or “Timing Simulation” button

◆ From Schematic Editor
  - Press “Simulation Toolbox” or “Simulator” icon
Simulation From Schematic

Probe tool  Stimulator tool  Simulator  Delete probes

Simulation toolbox

Logic Simulator Window

Display Comments On/Off  Measurements On/Off  Select Probes  Logical States

Delete all waveforms

Ruler On/Off

Zoom Out (LSR) waveforms button

I/O type

Signal or pin name  simulator name

logical state

Zoom In (ESR) waveforms button
The waveform window toolbar I

- **Ruler On/Off button**
  - you can enable or disable the waveform window ruler.

- **Delete all waveforms**
  - this button allows you to delete all waveforms and comments.

- **Display Comments On/Off**
  - this button allows you to toggle on/off the comment display.

- **Measurements On/Off**
  - this button enables you to perform and display precise timing measurements between any signal transitions, regardless of the scale.

- **Bus On/Off**
  - clicking on this button is meaningful only if you have defined some buses either in the waveform window or on the schematic.

The waveform window toolbar II

- **Select Components button**
  - this button is used to invoke the Components Selection window which is used to select signals and IC pins to the waveform window.

- **Select Simulator button**
  - this button invokes the Stimulator Selection window that is used to define and assign stimulators or test vectors to the desired signals.
  - The Stimulator Selection window includes the *Binary Counter*, *keyboard keys* and *asynchronous clocks*.
  - *Custom test vector key* ("Cs" key) that is used for assigning user defined test vector to any input or output and
  - *Formula key* that allows assigning formulas to clocks or Fn stimuli.

- **Logical States button**
  - invokes the logical state selection window that allows you to select and assign any logical state to a signal name or device pin.
Simulator Toolbar I

◆ Schematic editor button
  • switches control to the Foundation schematic editor program.

◆ Preferences button
  • displays Preferences window containing 5 cards: Simulation, General, Power On, Reports and Log Files.

Simulator Toolbar II

◆ Simulation mode box
  • allows you to select between
    – timing mode with selectable precision,
    – unit delay mode with all propagation delays equal to the selected simulation precision,
    – glitch mode with uneven time scale (emphasizing glitches),
    – functional mode with zero propagation delays.

◆ Power On button
  • initializes the entire design.

◆ Step simulation button
  • performs one simulation step.

◆ Simulation Step Value box
  • allows you to select or enter a length of the Simulation Step.
Simulator Toolbar III

◆ **Stop Simulation**
  - stops simulation at the current simulation cycle.

◆ **Search Anchor box**
  - allows you to select the type of an anchor (item or condition) to be searched for in the timing diagram.
  - *Breakpoints, Errors, Events, Milestones and Tags.*

◆ **Search Right and Left buttons**
  - moves the blue vertical bar cursor to the nearest anchor, located to the left or right of the current cursor position.

---

How to Add Signals to Your Sim.

◆ **Press Select Components button to open Component Selection**

◆ **Add signals to Waveform Viewer - there are three methods**
  - Select a signal or press “Ctrl” and select multiple signals, then press Add button
  - Double click a signal
  - Select signals and then drag them to Waveform Viewer
Creating and Applying Design Simulator Signals

◆ Applications of ready made test vectors using the virtual stimulator
  • This is a 16 bit software-driven binary counter that counts at a preset clock rate.

◆ Toggling signal lines with keyboard keys
  • Signals in the waveform viewer can be mapped to keys on the keyboard so enabling real time user control

◆ Formula
  • 16 signals in the simulator selection tool can each be assigned a formula.

◆ Drawing test vector waveforms
  • An editor permits waveforms to be drawn in the waveform viewer window.

◆ Using an existing Viewlogic command file

Stimulator Selection Window

◆ How Use Ready-Made Stimulators
  • Click on Add Stimulators in the Signal menu
  • Click on the selected signal name
  • After the selected signal name turns blue, click on the desired stimulator
  • The selected stimulator name appears in the Stimulator column, next to the signal line

- Keyboard keys allow you to toggle signal logical states directly from the keyboard
- TRUE outputs
- INVERTED outputs
- Formula Editor and Clock Editor
- Control buttons
- Asynchronous clocks can be defined for special clock signals
- CS - Custom Signal; indicates a manually drawn test vector
- Software-emulated binary counter
- Formula stimulators defined in the Formula Editor
Stimulator Selection Special Key

- The **Delete** button removes stimulator from the selected signal.
- The **EN** button enables the previously disabled stimulator.
- The **DS** button disables the selected stimulator.
- The **CC** button switches stimulator to the **Chip Controlled** mode, in which the stimulator will produce a weak signal that may be overridden by device or cell strong output.
- The **OV** button switches the selected stimulator to **Override** mode (the stimulator will produce a signal stronger than produced by any circuit output).
- The **Mode:**... button controls the override mode of currently entered stimulators.
  - If it is set to **OV** mode (default setting), then all entered stimulators will be set into that mode.
  - If this button is in the **CC** mode than all entered stimulators will automatically be in the **CC** mode and will be overridden by active device outputs.

Mapping Signal to Stimulator Selection

Counter, Keyboard, and Formula
Applying Formulas

◆ Examples:
  - H40L10: High for 40 ns then Low for 10 ns
  - (H40L10)20: Same as above but repeated 20 times
  - H4usL1us: High for 4 us (microseconds) then Low for 1 us
  - ((H10L10)20x30)10: High for 10 ns then Low for 10 ns, repeated 20 times, after that Unknown (X) for 30 ns. Repeat the entire signal waveform segment 10 times
  - ([2]40[A0]55)10: 02 hex for 40ns, then A0 hex for 55ns. Repeat 10 times

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>H, L</td>
<td>high and low logic level (1 and 0)</td>
</tr>
<tr>
<td>X</td>
<td>unknown (high or low)</td>
</tr>
<tr>
<td>Z</td>
<td>high impedance</td>
</tr>
<tr>
<td>0..9</td>
<td>numbers user for defining duration and repetitions</td>
</tr>
<tr>
<td>( )</td>
<td>parentheses for selecting sub-expressions</td>
</tr>
<tr>
<td>ps, ns, us, ms</td>
<td>time unit definition for duration arguments, default is ns</td>
</tr>
<tr>
<td>[ ]</td>
<td>brackets for defining hex bus value</td>
</tr>
</tbody>
</table>

Drawing Waveforms

◆ Waveform > Edit
◆ Low is a strong Low logical level
◆ High is a strong High logical level
◆ Unkn_X is a strong Unknown logical level
◆ High_Z is a week 3-state state that can be overridden by weak signals
Preferences - Simulation window

- Options -> Preferences
- **B0 period or B0 frequency** - selection of the period/frequency of the bit 0 of the software-driven 16-bit binary counter.
- **Simulation precision** - Select desired value from the Simulation Precision box. Please note that the simulation precision is different from the simulation step!

![Image of Simulation window]

Preferences - General window

- **End of Step Estimation** - if selected, displays estimated completion time of current simulation step,
- **Enable Breakpoints** - if selected, activates breakpoints
- **Prompt for Browsing Netlist Log** - if selected, displays special prompt when errors/warnings were detected during loading of the netlist.
- **Display Hidden Nets** - if selected, displays all net names in the component selection windows.
- **Enable Global Netlist Analysis** - if selected, enables to perform global netlist analysis stage during loading EDIF netlists. You can switch this option off to speed up EDIF netlist loading.

![Image of General window]
Preferences - Power On window I

- **Keep Comments** option preserves the old comments after Power On
- **Execute Preset** option executes the Selective Preset operation during Power On
- **Global Reset** option performs global reset operation during Power On
- **Model Stabilization** option waits till all operations scheduled for time 0 will be completed
- **Keep Measurements** option preserves the old measurements after Power On

![Preferences window I]

Preferences - Power On window II

- **Forced State** box allows selecting the initial state of the non-initialized internal signals of the models,
- **Output Waveforms** box lets you choose what will be done with the previous output waveforms
  - the old waveforms will be deleted during Power On (**Delete**)
  - the new waveforms will overwrite the old waveforms (**Overwrite**)
- **Pulse Global Reset Net** settings group allows automatic stimulation of Global Reset Net in Xilinx designs.
  - Xilinx FPGA family: GSR for 4k, Spartan and Virtex, GR for 5k and PRLD for 9k.
  - Net value is forced to the High State and stays in this state for specified period of time (as specified in width box).
  - NOTE: Pulse Global Reset Net does not override Stimulators set manually for GSP, GR or PRLD nets.
  - **Enable** option enables Global Reset Net stimulation.
  - **Width** box specifies the width of a pulse. A value of Width must be greater or equal than the current simulation precision but it must be less than 42.949673ms.

![Preferences window II]
Start Long Simulation

- Options -> Start Long Simulation...
- This window allows setting and starting long simulations.
- The Simulation Running Time box allows selection of the simulation time.
- Stop button to stop the simulation.
- The Start button starts simulation process.
- The Cancel button quits the Start Long Simulation window.

Simulate a macro

- Schematic Macro
  - Open a macro in Schematic Editor
  - Options > Simulate Current Macro
- Any Macro
  - Open the netlist of the macro In Logic Simulation
  - File > Simulate single component...
Simulation Script Basics

- Simulator macros allow you to specify a sequence of operations to be performed by the simulator on a design.
- ViewSim compatible
- How to run macro operations
  - Run Script File
    - Files > Run Script File...
  - Edit Script File
    - Tools > Script Editor
  - Interactive
    - Window > Command
**GENERAL**

◆ **Naming conventions**
  - **Component Names** - root/U1 (or U1) - Chip U1 at the root level
  - **Pin Names** - B11/B12/U25.Y1
  - **Signal Names** - root/RESET (or RESET) - signal RESET at the root level

◆ **Logical states**
  - Bus can expressed in Binary(\B), Octal(\O), Decimal(\D) and Hexadecimal(\H). They may also be expressed using their numerical bases, e.g. \2, \8, \16, \20

<table>
<thead>
<tr>
<th>BUS Signal State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Hi_Z</td>
</tr>
<tr>
<td>L</td>
<td>LOW</td>
</tr>
<tr>
<td>H</td>
<td>HIGH</td>
</tr>
<tr>
<td>X</td>
<td>UNKN_X</td>
</tr>
</tbody>
</table>

◆ **Time Units** - ps, ns, us, ms ,s ,m ,h
  - default : 0.1ns

**Simulation Script Reference I**

◆ **Comments:**
  - |

◆ **watch (w):**
  - watch rst clr enable

◆ **vector (v):**
  - vector Data d0 d1 ad[0:4]
    - Data = [d0, d1, ad0, ad1, ad2, ad3, ad4]

◆ **assign:**
  - assign ADBUS1 AB03\H
  - assign DATA < data_bus.dat

◆ **high (h):**
  - high U2.input
Simulation Script Reference II

◆ low (l):
  • low U21.output

◆ Force Unknown:
  • x U21.input

◆ stepsize (step):
  • stepsize 15ns

◆ clock (ck):
  • clock c 0 1 0 1 1 0 0 1
  - 1 clock cycle time = 8*15ns = 120ns

◆ cycle (c):
  • cycle 5
  - 5 clock cycle time = 5*120ns = 600ns

Simulation Script Reference III

◆ sim:
  • sim 53ns
  • sim (a single short STEP)

◆ every time do (cmd1;cmd2;...cmdn)
  • every 10ns do (display)

◆ Macro (Command) Loop:
  • (assign A < a.dat; assign B < b.dat; sim)*12

◆ SETUP Radix:
  • bin, oct, dec, hex
  • radix dec Data_In

◆ restart (Power-on operation)
◆ **set_stim (stim):**

- stim B0 clk
- B0, B1, ... B15 - denote bits of the binary counter (true value),
- N0, N1, ... N15 - denote inverted value of the binary counter bits,
- F0, F1, ... F15 - denote formula stimulators,
- C1, C2, C3 and C4 - denote clock stimulators,
- Cs - denote custom designed (waveform driven) stimulator.

◆ **SETUP Wfm:**

- wfm D0 @0=0 @1000=1 @2000=Z @3000=0
- wfm A5 @0=H (1000=L 2000=H)*10
- wfm Data0 events.dat
- wfm DATA @0=FF\H (1000=dec by 2)*8
  - inc, div, mult, rl(rotate left), rr(rotate right), sl(shift left), sr(shift right)

**Event File Example:**

- @0 = 0\H (hexadecimal)
- @100 = 1\H
- @200 = Z
- @300 = A\H

◆ **Set Stimulator Working Mode - stimuli_mode (smode)**

- stimuli_mode mode signal_name
- smode CC clk
- CC - chip controlled mode,
- OV - override (default) stimulator working mode,
- EN - connect disconnected stimulator (enable stimulator),
- DS - disconnect mode (disable stimulator),
- DL - deleting stimulator completely.
Test Vectors

Test Vector File Example:

- ab.dat test vector file
- test vectors for buses A i B
- format
- A[0:7]
- B[0:3]
- 10101101 | binary value
- 2D   | decimal value of 2;
- uses four lines B[0:3]
- AB\H | hexadecimal value of AB
- 0111
- 10001111
- Z
- The Assign macro automatically loads the consecutive data from
  the ab.dat file.

Command File Example:

- Assign A < ab.dat | assigns the first data (10101101) from ab.dat to the A bus
- Assign B < ab.dat | assigns the second data (2D) from ab.dat to the B bus
- sim | simulates the assigned bus values
- Assign A < ab.dat (AB\H) | assigns the 3rd data from ab.dat file
- Assign B < ab.dat (0111) | assigns the 4th data from ab.dat file
- sim
- Assign A < ab.dat (10001111)
- Assign B < ab.dat (Z)
- sim

Example Command File

```plaintext
restart
vector SW sw7\sw6\_p sw7\sw5\_p sw7\sw4\_p sw7\sw3\_p sw7\sw2\_p sw7\sw1\_p +sw7\sw0\_p
vector ALU alu[3:0]
vector STACK stack[3:0]
radix hex SW ALU
radix bin STACK
watch clk SW exc_p ALU STACK we rst
clock clk 0 1
| Use a clock period of 100ns. Set stepsize=50ns
step 50ns
h exc_p
assign SW 00\h
h gsr
| Viewsim uses units of 0.1 ns, so this statement
| simulates for 100 ns.
sim 1000
```

Script Editor Overview

- Implement all features and options typical for a regular text editor.
- Provide interactive communication with the logic simulator.
- The script editor can be used as a standalone editor tool, however in this mode all simulation options are unavailable.
Macro Assistant

◆ Tools > Macro Assistant

![SIM Macro Assistant](image)

Script Editor - Execute Menu I

◆ Go (F5)
  • The execution begins at the specified start point and continues up to the last line of the script.

◆ Restart (Shift+F5) ?
  • sets starting point for macro execution at the first available line.

◆ Stop (F10)
  • breaks the command script execution if the execution has been started in the free run mode or if a single command executes too long.

◆ Step (F7)
  • executes a single macro command. In this mode all nested command files (see execute command description) are treated as single command macros.
Script Editor - Execute Menu II

◆ **Trace (F8)**
  - executes a single macro command. Trace mode is used for line-by-line executing and browsing of hierarchical command scripts

◆ **Set Start Point (F4)**
  - sets starting point for macro execution at the line containing cursor (caret).

◆ **Toggle Breakpoint (F9)**
  - toggles breakpoint for free run execution mode.
Design Implementation Overview

Compile your design entry (schematic, state machine, HDL) to be physical implementation of design

User Interface
- Graphical User Interface
  - Merge into Foundation
  - Standalone Design Manager (execute: dsgnmg or xilinx)
- Command lines

Implementation Tools
- Flow Engine
- Floorplanner
- EPIC Design Viewer/Editor
- Automatic Pin Locking
- Xilinx Constraints Editor
**Design Flow Programs**

**NGDBUILD**
- Merges hierarchical EDIF or XNF files into one hierarchical file
- Creates internal netlist .ngd (Native Generic Design) files
- Contains logical components: combinatorial gates, RAMS, flip-flops, etc.

**MAP**
- Maps logical components to physical components found in Xilinx FPGA: look up tables, Flip-Flops, three state buffers, etc.
- Packs physical components into COMPS
- Creates internal .ncd (Native Circuit Design) file

**TRCE**
- Analyzes Timing
  - Use before PAR to analyze constraints
Design Flow Programs

◆ PAR
  • Places COMPS on FPGA
  • Routes the FPGA

◆ TRCE
  • Analyzes Timing
    – Use after PAR to check delays

◆ NGDANNO
  • Back-annotate timing delays for Simulation

◆ BITGEN
  • Create file to configure FPGA

Versions and Revisions I

◆ Each project may have multiple versions and revisions.
  • Versions represent logic changes in a design.
  • Revisions represent different implementations on a single design version.

◆ In Schematic Flow projects, new versions of the design and revisions on each version are associated with the Implementation phase.
  • Creating Versions
    – **Project -> Create Version**
  • The Schematic Flow Project Manager automatically creates a new version of the design when you click the Implementation phase button under the following conditions:
    – You used the Design Entry tools to make logic changes to the design (for example, you added a new block of logic, replaced an AND gate with an OR gate, or added a flip-flop).
    – You targeted a new device family for the design using the device family pulldown menu accessed from **Project -> Project Type**.
Versions and Revisions II

- Creating Revisions
  - Project -> Create Revision

Implementing a Design

- How to Implement a Design
  - use the Implementation phase button on the Project Manager's Flow tab
  - or implement your design by executing the Flow Engine steps

- Implementation steps by using the Implementation phase button
  - 1. Click the Implementation phase button on the project flowchart.
  - 2. Select Device, Speed in the Implement Design dialog box
  - 3. Select Options in the implement Design dialog box
  - 4. Click Run, it will automatically run Flow Engine
Flow Engine

◆ The Project Manager’s Implementation phase button automatically invokes the Flow Engine to process the design.

Flow Engine Control

◆ You can invoke and run the Flow Engine manually by selecting Tools -> Implementation -> Flow Engine
◆ Controlling Flow Engine Steps
  • 1. Create a new revision by selecting Project -> Create Revision.
  • 2. In the Project Manager Versions tab, select the revision.
  • 3. Select Tools -> Implementation -> Flow Engine from the Project Manager's menu bar.
  • 4. Select Setup -> Options from the menu in the Flow Engine to access the Options dialog box.
  • 5. Set the appropriate options in the Options dialog box.
  • 6. Click OK to return to the Flow Engine.
  • 7. To start the Flow Engine, do one of the following.
    – In the Flow Engine window, select Flow -> Run.
    – Select Flow -> Step to single step through the implementation process.
    – Optionally, you can select Setup -> Stop After and select where to stop processing.
The implementation reports provide information on:
- Logic trimming, logic optimization, timing constraint performance, and I/O pin assignment.

To access the reports:
- Report Browser button on Project Manager toolbar
- Or select the Reports tab from Project Flow area of the Project Manager.
  - Double click the Implementation Report Files icon to access the Report Browser
  - Yellow sparkles indicates new (not yet read)

Report Files I

Translation Report:
- Contains warning and error messages from translation processes:
  - Missing or untranslatable hierarchical blocks
  - Invalid or incomplete timing constraints
  - Output contention, loadless outputs, and sourceless inputs

Map Report:
- Erroreously removed logic.
- Logic that has been added or expanded to optimize speed.
- The Design Summary section lists the number and percentage of used CLBs, IOBs, flip-flops, and latches.

Place & Route Report includes resource summary:
- The overall placer score which measures the "goodness" of the placement.
  - Lower is better.
  - The score is strongly dependent on the nature of the design and the physical part that is being targeted
Report Files II

- The meaningful score comparisons can only be made between iterations of the same design targeted for the same part.
  - The Number of Signals Not Completely Routed should be zero for a completely implemented design.
    - If non-zero, you may be able to improve results by using re-entrant routing or the multi-pass place and route flow.
  - The timing summary at the end of the report details the design's asynchronous delays.

◆ Pad Report
  - contains a report of where each of the device pins were located in the device

◆ Asynchronous Delay Report
  - enumerates all the nets in the design and the delays of all the loads on the net

◆ Post Layout Timing Report provides timing summary
  - A timing summary report shows the calculated worst-case timing for the logic paths in your design.

Implement from Design Manager

- Command Line : dsgnmgr, xilinx
- 開始 ➔ 程式集 ➔ Xilinx Foundation Series ➔ Accessories ➔ Design Manager
Begin the Design Implementation

- Select Design -> Implement
- or click on the Right arrow

You can select options (...see later foil)
M1 Implementation Options

Design Translation Flow Chart

- LogiBLOX → NGO → UCF
- EDIF → NCF → EDIF2NGD → .NGO
- EDIF → EDIF2NGD → .NGO
- NGDBUILD
- NGD

Logical DRC Check-point
NGDBUILD
The Design Translation Control Center

◆ What NGDBUILD does

- Translation of logical design (netlist) to Xilinx internal netlist (NGD)
- Merges multiple design netlist files which make up a design into an NGD file for use by “map”
  - User constraints are also included in the NGD file

Design Mapping Flow Chart
Functions of MAP

◆ MAP transforms a logic design into an equivalent physical implementation
  • Map re-configures your design in terms of FPGA’s logic resources.
◆ MAP creates a physical constraint file (PCF) which contains all of the design constraint data applied through design translation.
◆ MAP creates the NGM file which correlates the logic design to its physical implementation - facilitating Back-annotation.

Main Implementation Menu Options

◆ User File
  • Specify constraint file (optional)
◆ Program Option
  • Implementation has four sub-menus: Optimize and Map, Place and Route, Timing, and Interface
Optimization and Map Options I

- Map optimizes your design before it is partitioned into LUTs, Flip-Flops, etc.
- The GUI includes these options:
  - **Trim Unconnected Signals** (default is On)
    - Trims all fan-out/fan-in from unconnected pins
    - Turn off to implement hierarchical blocks separately
  - **Replicate Logic** (default is On)
    - Replicate a single driver that drives multiple loads and map it as separate components that drive individual loads.

Optimization and Map Options II

- This option is useful for creating a mapping strategy that may more readily meet your timing constraints.
- It reduces the number of logic elements through which a signal must pass, thereby eliminating path delays.
- **Generate 5-Input Functions** (default is Off)
  - Map each five-input logic function to a single CLB.
  - This option can sometimes reduce the number of cell-to-cell delays at the expense of increased CLB count.
- **CLB Packing Strategy** (default is Fit Device)
  - Informs Map of how to pack COMPS with logic
    - **Fit Device** -
      - Pack logic elements that do not share common signals into the CLBs.
      - The mapper continues packing until the design fits into the selected device or no further packing is possible.
    - **OFF** -
      - Disabling this option causes only related logic (logic with common inputs) to be packed together.
      - This is useful for increasing speed in high speed designs.
**Optimization and Map Options III**

- **Pack CLB Registers for** (default is Structure)
  - Minimum Area
    - Disable register ordering for a denser design.
  - Structure
    - Enable register ordering.

- **Pack I/O Registers/Latches into IOBs for** (default is Off)
  - Normally, the mapper packs flip-flops or latches within an I/O cell only if such packing is specified by your design entry method. This option allows you to control packing after the design entry phase.
    - Inputs Only
    - Outputs Only
    - Inputs and Outputs
    - Off

**Optimization and Map Options IV**

- **Use Generic Clock Buffers (BUFGs) in place of BUFGPs and BUFGSs** (default is Off)
  - This option is useful when working with design entry tools that generate only BUFGPs or BUFGSs.
What PAR Does

◆ **Places the COMPs within an NCD based on:**
  - guided design requirements
  - location constraints (within the PCF file)
  - the design’s structure
  - estimate of the routing delays for each layout

◆ **Routes the signals within an NCD based on:**
  - guided design requirements
  - available slack for each signal (from Tspecs)

◆ **Performs “Clean-up” routing passes**
  - to reduce “design score”
  - to reduce (all) net delays
Place and Route Options I

- **Place & Route Effort Level (default is 2)**
  - Trades off place and route effort versus CPU time

- **Routing Passes (default is Auto)**
  - The Router will run until no improvement is made to meet timing constraints.
  - Set Router passes to avoid very long run times for difficult designs.
  - Start with 5 passes for 4000EX/XL chips, 7 passes for 4000E chips

---

Place and Route Options II

- **Delay Based Clean-up Passes**
  - Use this option to further route an already routed design.
  - The router makes routing decisions based on computed delay times between sources and loads on the routed nets, and reroutes to minimize the delays.
  - This option is useful if you want to manually route a portion of the design and then automatically route the design or if you want to run additional delay reduction passes.
  - Set the number of delay-based cleanup passes from 1 to 5. The default is 0.

- **Use Timing Constraints During Place and Route**
  - Select this option to use timing constraints in the design file to place and route the design within the specified constraints.
  - By default, this option is on.
Implementation Options for Fast Runtime versus PAR Effort

- Select fast placement option, 1-2 routing passes, 0 clean-up passes, and deselect "Use Timing Constraints"

Timing Report Options

- Enable the creation of the Timing Report
  - Logic Level Timing Report is created before PAR
    - Has zero net delays
    - Used to analyze constraints
  - Post Layout Timing Report is created after PAR
    - Verify that the design meets constraints
- Timing reports generated on a mapped design use 0ns delays for nets
- Post-route timing reports use actual delays based on placement and routing
Controlling the Back Annotation Netlist Format

Format options:
- VHDL
- Verilog
- EDIF

Appendix

Advanced PAR
Incremental Design Support

- When using incremental design, unconnected pins will be present
- By default, Xilinx removes fan-in and/or fan-out of unused pins
- So, disable the “Trim Unconnected Logic” option
  - Setup -> Options -> Optimize and Map

Set Guide File

- Set Guide File in Design Manager
The Guide Option

◆ Allows use of a previously placed and routed design to guide a new placement
  • Useful if there are few design changes

◆ Guide is used for Map, Place, and Route
  • Map may take longer to execute, but PAR will be faster

![Diagram showing the guide option process]

Effective use of Guide

◆ Guide uses signal and component names to determine edited parts of the design

◆ Name all nets
  • Do not change names

◆ Minimize changes to the design
  • Any new hierarchy changes all names below
  • Avoid any changes to synthesized logic

◆ Synthesis users: please try to freeze the design with “set_don’t_touch” or like command
  • Otherwise, guide option may not be useful
Advanced PAR

- After placement is complete, routing passes begin
- There are two routing stages:
  - Iterative phase: Goal is to route all nets, and meet all timing specifications
  - Cleanup phase: after iterative routing is complete, cleanup minimizes delays on all paths. There are two types:
    - Cost-based: minimizes "cost" of nets. Example: long routing connections (such as those used by three state buffers) have a higher cost than shorter connections. Router thus prefers shorter interconnections.
    - Delay-based: calculates individual connection delays and will choose faster routes if available

Running Re-Entrant Routing on FPGAs I

- Use re-entrant routing to further route an already routed design.
  - The design maintains its current routing and additional routing is added.
  - The router reroutes some connections to improve the timing score or to finish routing unrouted nets.
- To Perform Re-Entrant Routing
  - Select Setup -> FPGA Re-entrant Route from the Flow Engine.
Running Re-Entrant Routing on FPGAs II

◆ Cost-Based Cleanup Passes (1-5, default: 1)
  - Cost-based cleanup passes reroute nets if the new routing uses less costly resources than the original configuration.
  - Cost is based on pre-determined cost tables.
  - Cost-based cleanup usually has a faster runtime than the delay-based cleanup, but does not reduce delays as significantly.

◆ Delay-Based Cleanup Passes(1-5, default: 0)
  - Delay-based cleanup passes reroute nets if new routing will minimize the delay for a given connection.
  - Delay-based cleanup usually produces faster in-circuit performance.

◆ Re-entrant Route Passes(1-2000, default: auto)
  - Re-entrant routing runs additional timing driven or non-timing driven routing passes, depending on whether you specified timing constraints.
  - Placement and routing from previous place and route runs are preserved.

FPGA Multi-Pass Place & Route I

◆ In Design Manager, Select Design Version
◆ Design -> FPGA Multi-Pass Place & Route

◆ Initial Placement Seed (Cost Table)
  - Specify a placement initialization value with which to begin the place and route attempts.
FPGA Multi-Pass Place & Route II

- Each subsequent attempt is assigned an incremental value based on the starting strategy value.
- The number you choose corresponds to a cost table index and results in different place and route strategies.
- Cost tables assign weighted values to relevant factors such as constraints specified in the input file (for example, certain components must be in certain locations), the length of connections, and the available routing resources.
- Choose a number from 1 to 100. The default is 1.

◆ Place & Route Passes to Execute
- Specify the number of place and route iterations to attempt. Choose a number from 1 to 100. The default is 1.

FPGA Multi-Pass Place & Route III

◆ Save N Best Passes
- Specify the number of place and route iterations to save.
- Choose a number from 1 to 100. The default is 1.
- This option compares every result to every other result and leaves you with the best iteration attempts.
- The best outputs are determined by a score assigned to each output design.
- The lower the score, the better the design.
Advanced Place and Route Options ...

◆ How many passes should be run? Here are a few estimates based on family:
  - 3K, 5K, and smaller 4KE/XL devices ----------- 15+ passes
  - Mid-range 4000E/4000XL devices ----------- 8 -12 passes
  - XC4028EX/XL and XC4036EX/XL ----------- 3-6 passes
  - XL devices > XC4036 ----------- <3 passes

◆ Guidelines are based on length of time to run passes
  - Guidelines above assume a run time of ~10 hours on a SPARC 10 workstation

◆ Other factors which effect runtime:
  - Loose timespecs decrease runtime of each pass
  - As percent of device used by design increases, run time increases
  - Larger devices will take longer to place and route
Timing Analyzer

- Analyze delays before and after implementation
Timing Analyzer Benefits

◆ Combines block delays from data book with net delays from implementation files
◆ Quickly identifies critical paths and timing hazards
◆ Report shows all elements in path, each element's delay, and total delay
  • Can determine if slow paths are due to block delays (design) or net delays (implementation)

<table>
<thead>
<tr>
<th>Element</th>
<th>Delay</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAD to IOB.I</td>
<td>2.2</td>
<td>block</td>
</tr>
<tr>
<td>IOB.I to CLB1.F1</td>
<td>1.1</td>
<td>net</td>
</tr>
<tr>
<td>CLB1.F1 to CLB1.X</td>
<td>2.7</td>
<td>block</td>
</tr>
<tr>
<td>CLB1.X to CLB2.F3</td>
<td>1.2</td>
<td>net</td>
</tr>
<tr>
<td>CLB2.F3 to Clock</td>
<td>2.1</td>
<td>block</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9.3</strong></td>
<td></td>
</tr>
</tbody>
</table>

Basic Path Type - Clock to Setup

◆ A clock-to-setup (C2S) path starts at the Q output of a flip-flop or latch and ends at an input to another flip-flop, latch, or RAM, where that pin has a setup requirement before a clocking signal.

◆ It includes
  • the clock-to-Q delay of a flip-flop,
  • the path delay from that flip-flop to the next flip-flop,
  • the setup requirement of the next flip-flop.
Basic Path Type - Clock to Setup

- **Same Clock**

- **Rising to Falling Edge**

- **Different Clocks**

- **Falling to Rising Edge**

- **Clock 1**

- **Clock 2**

- **Path Delay**
  - Minimum clock period
  - Minimum clock low time
  - Minimum clock period for clock 1 & clock 2
Basic Path Type - Clock to Pad

- A clock-to-pad (C2P) path starts at the Q output of a flip-flop or latch and ends at an output of the chip.
- It includes:
  - the clock-to-Q delay of the flip-flop
  - the path delay from that flip-flop to the chip output.
- The clock-to-pad path time is the maximum time required for the data to leave the source flip-flop, travel through logic and routing, and leave the chip.

Basic Path Type - Pad to Setup

- A pad-to-setup (P2S) path starts at an input of the chip and ends at an input to a flip-flop, latch, or RAM.
- The pad-to-setup path time is the maximum time required for the data to enter the chip, travel through logic and routing, and arrive at the output before the clock or control signal arrives.
Basic Path Type - Pad to Pad

- A pad-to-pad (P2P) path starts at an input of the chip and ends at an output of the chip.
- The pad-to-pad path time is the maximum time required for the data to enter the chip, travel through logic and routing, and leave the chip.
- It is not controlled or affected by any clock signal.

Basic Path Type - Clock Input

- A clock input path starts at either an input of the chip or at the output of a flip-flop, latch, or RAM. It ends at any clock pin on a flip-flop or latch enable.
- The clock input path time is the maximum time required for the signal to arrive at the flip-flop clock input.
**Basic Path Type - Clock Skew**

◆ **Clock skew is the difference in arrival time for clocks that are on the same clock net.**

◆ **When the destination is clocked before the source, the clock skew is called negative clock skew.**
  
  • Negative clock skew means that the clock period must be longer than the path delay plus the amount of clock skew between the flip-flops.

---

**Basic Path Type - Clock Skew**

◆ **When the source is clocked first, the clock skew is called positive clock skew.**
  
  • Positive clock skew means that the clock period could be shorter than the path delay by the minimum amount of clock skew.
Invoking Timing Analyzer

◆ **In Project Manager**
  - Select revision in Versions tab
  - Tools -> Simulation/Verification -> Interactive Timing Analyzer

◆ **In Design Manager Toolbox**
  - Select revision
  - Click icon

Creating Reports

◆ **Timing Constraint Analysis**
  - The Timing Constraints Analysis report compares the design's performance to the timing constraints.
  - Report Paths in Timing Constraints
  - Report Paths Failing Timing Constraints
  - Report Paths Not Covered by Timing Constraints

◆ **Advanced Design Analysis**
  - The Advanced Design Analysis report provides a set of summary statistics for the paths from the timing requirements submitted for analysis.

◆ **Custom Analysis**
  - Analyze all specified paths. Filters from the Custom Filters and Common Filters submenus are applied during analysis.
Use Filters to Specify Paths to Report

- **Need to limit report file to paths of interest**
  - Can specify by source/destination, type, or included/excluded nets

Limiting Paths by Sources/Destinations

- **Can be** Flip-Flops, Pads, Nets, Pins, CLBs, Clocks, or All
- **Then select specific elements from list**
Limiting Paths by Nets They Contain

- Can select paths by nets to include/exclude

Control Path Tracing

- Specifies Timing Analyzer path tracing. (Only for FPGA)
  - Path Type: Default
  - Asynchronous Set/Reset to output: Disabled
  - Data to output for transparent latch: Disabled
  - RAM data to output: Disabled
  - RAM WE to output: Enabled
  - TBUF tristate control to output: Enabled
  - TBUF input to output: Enabled
  - I/O pad to input: Enabled
  - I/O tristate control to pad: Enabled
  - Bidirectional tristate I/O output to input: Enabled
  - I/O output to pad: Enabled
Report Options I

◆ Summary Report Only
  • Generates a summary report containing only the path source and end point.
  • Lists one delay path per line and does not display cumulative delays through CLBs.
  • Applies only to FPGAs.

Report Options II

◆ Maximum Paths per Timing Constraint
  • Sets the limit for the number of paths reported per timing constraint.
  • For FPGAs, the default is one path reported per timing constraint.
  • For CPLDs, the default is unlimited.
  • Not specifying a value is equivalent to "no limit."

◆ Wide Report
  • Creates a report formatted into 132 characters per line,
  • If not selected, a "normal" report (80 characters per line) is generated.
Recommended Verification Flow

1. Netlist
2. Implement
3. Timing Analysis
4. Bitgen Prom File Formatter
5. Download
6. FUNCTIONAL SIMULATION
7. TIMING SIMULATION
8. IN-CIRCUIT VERIFICATION
Brief Review of Constraint Flow

LOGICAL DOMAIN

- EDIF
  - .ncf
- .ucf
  - User netlist and logical constraints

DESIGN TRANSLATION

- NGDBUILD
- MAP

PHYSICAL DOMAIN

- .ncd
- .pcf
  - Mapped design and physical constraints

- PAR
- TRCE
- EPIC
- NGDANNO
- DRC
Types of Constraints

◆ Constraint commands define the maximum allowable delay or placement along paths
◆ Constraints control XACT tools during implementation
◆ Timing and location constraints can be specified in schematic or user constraint (UCF) file
  • Some synthesis tools create a NCF file for constraints
◆ Timing Constraints
  • Specify maximum allowable delay along logic paths
◆ Location Constraints
  • Specify location of components on FPGA

User Constraint File
TNM Attributes

◆ Timing name (TNM)
  - be used to identify the elements that make up a group and give them a name that can later be used in an actual timing specification.
  - TNMs
    - on Nets
    - on Macros or Primitive Pins
    - on Primitives
    - on Macro Symbols
  - In Schematic
    - TNM=CLK_GRP
  - In UCF
    - NET CLK TNM=CLK_GRP;

TIMEGRP Constraints

◆ Time Group (TIMEGRP)
  - use existing TNMs to create new groups or to define a group based on the output nets that group sources.
  - In Schematic (TIMEGRP Symbol)
    - NEWGRP=OLD1:OLD2
  - In UCF
    - TIMEGRP NEWGRP=OLD1:OLD2;
Basic Timing Constraining

◆ **PERIOD**
  - PERIOD is the duration of the clock and can be configured to have different duty cycles.
  - Derived clocks can be defined as a function of another clock (*, /)
  - PERIOD is preferred over FROM:TO constraints; M1 tools will have a faster runtime.
  - PERIOD should cover most of design.

◆ **OFFSET**
  - OFFSET is the relationship of the data to the clock at the external pins of the device.
  - OFFSET incorporates the internal global clock delays; FROM:TO does not.

### Period Examples

◆ **By net**
  - NET CLK50 PERIOD = 20 ns ;
  - NET CLK20 PERIOD = 50 HIGH 20 ;

◆ **By group**
  - NET CLK50 TNM = CLK50_GRP ;
  - NET CLK25 TNM = CLK25_GRP ;
  - TIMESPEC TS_CLK_FULL = PERIOD CLK50_GRP 20;
  - TIMESPEC TS_CLK_HALF = PERIOD CLK25_GRP TS_CLK_FULL * 2 ;
OFFSET Examples

The following two UCF files are equivalent:

```
NET CLOCK PERIOD=40;
##External (shown in diagram)
NET ADD0_IN OFFSET = IN 14 AFTER CLOCK;
NET ADD0_OUT OFFSET = OUT 25 BEFORE CLOCK;
```

```
NET CLOCK PERIOD=40;
##Internal (not shown in diagram):
NET ADD0_IN OFFSET = IN 26 BEFORE CLOCK;
NET ADD0_OUT OFFSET = OUT 15 AFTER CLOCK;
```

Basic Global Timing Constraints

( using the FROM-TO Syntax)

UCF TIMESPEC command using default keywords:

```
TIMESPEC TS_C2S=FROM:FFS:TO:FFS:30;
TIMESPEC TS_P2S=FROM:PADS:TO:FFS:25;
TIMESPEC TS_P2P=FROM:PADS:TO:PADS:26;
TIMESPEC TS_C2P=FROM:FFS:TO:PADS:9;
```
Specific Delays from one group to another

- Qualifying predefined groups to create path-specific constraints:

```
TIMESPEC TS_FIFOS = FROM : RAMS(FIFORAM<*>*) : TO : FFS(MY_REG*) : 25;
```

Note: The pattern matching is on the output signal of the FFS/RAMS, not the symbol name. Use INST to pattern match on the symbol name.

Specific Delays going through specific logic (TPTHRU)

- Forces the path through specific logic.
- The TPTHRU attribute is attached to net / instance / macro in top blob.

```
NET $3M17/ON_THE_WAY TPTHRU = ABC;
TIMESPEC TS_FIFOS=FROM:RAMS(FIFORAM<*>*) : THRU:ABC:TO:FFS(MY_REG*) : 25;
```
Specific Delays Excluding Logic

◆ You can create subgroups based on names with “EXCEPT”

◆ Example:

• Assume this design has a data busses that all start with “DATA”. Use the EXCEPT command to create a group with all the pads except the data pads.

```
TIMEGROUP CTRL_PADS = PADS: EXCEPT (DATA*);
TIMEGROUP DATAPINS = PADS (DATA*);
TIMESPEC TS_IO1=FROM:CTRL_PADS:TO:FFS:20;
TIMESPEC TS_IO2=FROM:FFS:TO:CTRL_PADS:20;
TIMESPEC TS_IO3=FROM:CTRL_PADS:TO:CTRL_PADS:30;
TIMESPEC TS_IOCTL=FROM:DATAPINS:TO:FFS:15;
```

Constraining Between Rising & Falling Clock Edges

◆ Define clock groups, the ("*" ) covers all FFS in your design

• TIMEGRP RFFS = RISING FFS ("*");
• TIMEGRP FFFS = FALLING FFS ("*");

◆ Define timing constraints

• TIMESPEC TS_R2F=FROM:RFFS:TO:FFFS:30;
• TIMESPEC TS_F2R=FROM:FFFS:TO:RFFS:30;

◆ Remember, the PERIOD constraint will automatically account for two-phase clocks.
Constraining Between Multiple Clock Domains

◆ Define clock groups
  - NET CLK_A TNM=A_GRP;
  - NET CLK_B TNM=B_GRP;

◆ Define timing constraints
  - TIMESPEC TS_CLKA=PERIOD A_GRP 20;
  - TIMESPEC TS_CLKB=PERIOD B_GRP TS_CLKA*2;
  - TIMESPEC TS_CLKA2B=FROM:A_GRP:TO:B_GRP:20;
  - TIMESPEC TS_CLKB2A=FROM:B_GRP:TO:A_GRP:20;
    - Covered by TS_CLKA therefore not needed

Creating new synchronous endpoints (TPSYNC)

Allows definition of synchronous points that are not FFS, RAMS, PADS or LATCHES.
  - Commonly used with three-state buffers.
  - Example:
    NET $3M17/BLUE TPSYNC = BLUE_S;
    TIMESPEC TS_1A=FROM:FFS:TO:BLUE_S :15 ;
Ignoring Paths (TIG)

- Never changing input signal
  \[ \text{NET CHIP\_MODE TIG;} \]

- Ignore a signal for a specific timespec
  \[ \text{NET SLOW\_SIG TIG=TS\_01;} \]

- Ignore false paths between registers
  \[ \text{TIMEGROUP TS\_TIG01=FROM:FFS(REGA*) :TO:FFS(REGB*):TIG;} \]
  Note: May have to use INST to create groups for synthesis designs.

Timing Constraint Priority I

- It is legal to constrain the same paths more than once
  - Known as a constraint conflict
  - Multiple sources constraining the same path
    - UCF and schematic could constrain same path
  - Multiple constraints on one net within one source

- Resolution of conflicting constraints from multiple sources:
  - Lowest Priority - input netlist or .ncf file
  - .ucf file
  - Highest Priority - .pcf file (usually from MAP)

Note: this priority only applies to timespecs with identical TSidentifiers (e.g. TS\_03 =...)}
Timing Constraint Priority II

◆ Within a particular source:
  • Highest Priority
  • Lowest Priority

Timing ignores \( \text{TIG} \)

FROM : THRU : TO specs
Source and destination defined by user
Source or destination defined by user
Source and destination are pre-defined groups

FROM : TO specs
Source and destination defined by user
Source or destination defined by user
Source and destination are pre-defined groups

PERIOD specs
“Allpaths” type specs (.pcf only)

Timing Constraint Priority III

◆ Same path constrained with different FROM:TO statements
  • Highest Priority
    - Source and destination defined by user
    - Source or destination defined by user
  • Lowest Priority
    - Source and destination are pre-defined groups

◆ You can explicitly assign priorities
  • Syntax
    – \((\text{SOME\_NORMAL\_TIMESPEC}) : \text{PRIORITY} : \text{integer}\)
  • Low numbers specify high priority
**SKEW I**

- **SKEW** is the difference in the arrival time of the clock pulse between a source and destination register (or other synchronous element).

  If the clock pulse arrives at the source reg first followed by the dest reg, then it is “positive skew”; if it arrives at the dest followed by the source, then it is “negative skew”.

- Some Positive skew can be beneficial; it will decrease the required setup time:

- Too much Positive skew can create a race condition, hold-time violation:

**SKEW II**

- Negative skew is usually undesirable; it will increase the required setup time:

  - If you use the global clock resources, then there should be no danger of hold-time violations for internal paths.

  - **TRACE** can check for race conditions; set env. variable XILINX_DORACECHECK to activate this (same environment variable will activate skew-checking).
MAXSKEW: Limiting SKEW

- Signal SKEW may also be constrained using the MAXSKEW constraint
  - NET $1I3245/$SIG_6 MAXSKEW=3;
  - i.e. specifies a maximum of 3ns difference between the source of net $1I3245/$SIG_6 and all its destinations is permissible
- May use to control skew of logic driven clocks (or any clock using non-global resources)
- Cannot constrain skew of global nets (Makes no sense as skew is fixed)

Placement & Other Constraints

Using LOC, BLKNM, and other physical constraints
Logical Location Constraints

◆ “Absolute” Location constraints place a component at a specific site on the FPGA

◆ Example:

```
INST U45 LOC = CLB_R1C5;
```

- Occupies Row 1, Column 5 in the FPGA

◆ “Relative” (RLOC) location constraints specify the relative placement of components within a macro

Absolute Location Constraints in the “Logical Domain”

◆ “LOC” constraint used to locate:

- BUFTs, FFs, MAPs, CLBs, PADs, WANDs, decoders, global buffers

◆ Syntax:

- single components (e.g. CLBs)

```
INST U45 LOC=CLB_R1C5;
```

- ranges of components (but not IOs)

```
INST U46 LOC=CLB_R2C2:CLB_R4C6;
```

- multiple sites for single component

```
INST U50 LOC=CLB_R1C1; CLB_R2C1;
INST /top/ddf LOC=CLB_R*C2;
```

- Pin assignments use similar syntax:

```
INST QOUT<3> LOC = P32;
NET outa LOC = P18;
```
Relative Location (RLOC) Constraints

◆ The following symbols (primitives) accept RLOCs.
  • Registers, FMAP, HMAP, F5MAP, CY4, CY_MUX, ROM, RAM, RAMS, RAMD, BUFT, WAND primitives that do not have a DECODE attribute attached

◆ Guidelines for Specifying Relative Locations
  • RLOC=RmCn [extension]
    INST xcv/U0 RLOC=R0C0;
    INST xcv/U1 RLOC=R1C0;
  • RLOC_RANGE=Rm1Cn1:Rm2Cn2
    INST xcv/U2 RLOC_RANGE=R2C0:R4C2;
  • Site of upper left hand corner of pre-placed macro
    INST xcv RLOC_ORIGIN=CLB_R5C4;

Mapping Constraints

◆ Force logic into the same CLB.
  • Sometimes MAP doesn’t make the best decisions. This allows the user to map logic together.

◆ Syntax
  INST state_reg_1 BLKNM=STATE1;
  INST state_reg_1 BLKNM=STATE1;
  INST my_FMAP_logic BLKNM=STATE1; #Can constrain FMAPs but not gates

◆ This will force my_FMAP, state_reg_1, state_reg_2, and into the same CLB (“STATE1”)

Note: The remaining resources are still up for grabs by MAP (in this case, one of the FG’s is still available).
Implementation Constraints

◆ Physical implementation may be controlled in the UCF file, such as:

- FAST : Set Faster IO Slew rate
  - eg \text{INST} \$1I87/OBUF\ FAST
- PART : Define Part-type to be used
  - eg \text{CONFIG} \PART=4005E-PQ160C-5;
- BUFG : Force signal to onto global net (CPLD only)
  - eg \text{INST} \text{clkgen/fastclk} \ BUFG;
- INIT : Define initial RAM/ROM Contents (primitives only)
  - eg \text{INST} \$1I3245/ROM2 \ INIT = 5555;

Conclusion

Overview of Constraints
Basic constraints file

- **Most generic timing constraints for fastest PAR runtime**
  
  NET CLK1 PERIOD = 40;
  NET OUT* OFFSET = OUT 13 AFTER CLK1;
  TIMESPEC TS01 = FROM PADS TO PADS 40;

More specific constraints file I

- **Clocks**
  
  NET CLK TNM=CLK;
  NET CLK2 TNM=CLK2;
  TIMESPEC TS_CLK01=PERIOD CLK 40;
  TIMESPEC TS_CLK02=PERIOD CLK2 50;

- **Ignore paths between 2 async clocks**
  
  TIMESPEC TS_TIG1=FROM:CLK:TO:CLK2:TIG;
  TIMESPEC TS_TIG2=FROM:CLK2:TO:CLK:TIG;

- **Generic path to outputs**
  
  TIMESPEC TS_IO1=FROM:FFS:TO:PADS:20;

- **Two cycle path to slow outputs**
  
  TIMESPEC TS_IO2=FROM:FFS:TO:PADS(SLOW*):TS_IO1*2;
More specific constraints file II

◆ Offset for late input signal
  
  NET LATE_INPUT OFFSET=IN:30:AFTER:CLK;

◆ Ignore static input signal
  
  NET CHIP_MODE TIG;

◆ Static control registers
  
  INST CONTROL_BLOCK/CTRL_REG* TNM=CTRL_REG;
  TIMESPEC TS_CLK03 = FROM:CTRL_REG:TO:FFS=TS_CLK01*2;

◆ Fast OBUF attached to component
  
  INST RAM_CS FAST;

◆ Prohibit Pins
  
  CONFIG PROHIBIT = P6;

Constraint Recommendations

◆ Do not use the same TIMESPEC name for more than one path. BAD
  
  Example:
  
  • TIMESPEC TS01: FROM X TO Y 20;
  • TIMESPEC TS01: FROM E TO F 30;

◆ Keep constraints in one source
  
  • Either UCF file or in schematics, but not both

◆ OVER-constraining your design is bad news!
  
  • Design Performance suffers
    – Critical timing paths get the best placement and fastest routing options
    – As the number of critical paths increases, routability decreases
  • Run times increase

◆ Reference - Libraries Guide Chapter 12
The Constraints Editor is a Graphical User Interface (GUI) that you can use to add new constraints to your design.

Input files to the Constraints Editor are:
- NGD (Native Generic Database) file. This file serves as input to the mapper, which generates the physical design database (NCD file).
- Corresponding UCF (User Constraint File).

By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

Upon successful completion, the Constraints Editor writes out a valid UCF file.
 Constraints Editor Window

Opening Constraints Editor

◆ To open the Constraints Editor
  • 開始 > 程式集 > Xilinx Foundation Series > Accessories > Constraints Editor

◆ Before you new or open a constraints file in Constraints Editor, you should run Flow Engine to make a .ngd
  • When you new or open a UCF file, Constraints Editor will load the .ngd file.
New/Open an UCF file

◆ Edit a new UCF file
  • File > New
  • New Constraints File
    – key in the UCF filename
  • Open Design File
    – Select an existing .ngd file

◆ Open an existing UCF file
  • File > Open
  • Open Constraints File
    – Select an existing UCF file
  • Open Design File
    – Select an existing .ngd file

Using the Global Tab Window

◆ Specifying constraints
  • Clock Period
  • Pad to Setup
  • Clock to Pad
  • Pad to Pad
Clock Period

On the Global tab window, in the Period column and in the row associated with the appropriate clock net name, double-click the left mouse button. This opens the Clock Period dialog box.

NET "CLK" TNM_NET = "CLK";
TIMESPEC "TS_CLK" = PERIOD "CLK" 20.000000 ns HIGH 50.000000 %;

Pad to Setup / Clock to Pad

OFFSET = IN 20.000000 ns BEFORE "CLK";
OFFSET = OUT 20.000000 ns AFTER "CLK";
Using the Ports Tab Window

◆ Specifying Constraints
  - Location
  - Pad to Setup
  - Clock to Pad
  - I/O Configuration Options
    - Clicking the I/O Configuration Options box adds a column titled Fast/Slow to the Ports tab window.
  - Prohibit I/O Locations

Prohibit I/O Locations

◆ This disallows the use of an I/O site by PAR and EPIC

CONFIG PROHIBIT = "P16";
Using the Advanced Tab Window

**Creating and Specifying Constraints**

- Groups by Elements (TIMEGRP)
- Groups by Nets (TNM)
- Timing THRU Points (TPTHRU)
- Pad to Setup Requirements (OFFSET IN BEFORE)
- Clock to Pad Requirements (OFFSET OUT AFTER)
- Slow/Fast Path Exceptions (FROM TO)
- Multi-cycle Paths (FROM TO)
- False Paths (FROM TO TIG)
- False Paths by Nets (NET TIG)
- Manage Editable
- View Source Constraints

---

**Time Group / Time Name**

**Groups by Elements (TIMEGRP)**

```
TIMEGRP "AAA" = FFS( "$I24/Q0" "$I24/Q1" "$I24/Q2" );
```

**Groups by Nets (TNM)**

```
NET "$Net00001_" TNM_NET = "NNN";
```
Timing THRU Point

- Timing THRU Points (TPTHRU)
  NET "$I21/$Net00344_" TPTHRU = "TTT";
  NET "$I21/CEO" TPTHRU = "TTT";

Pad to Setup / Clock to Pad

- Pad to Setup Requirements (OFFSET IN BEFORE)
  TIMEGRP "INPAD" OFFSET = IN 20.000000 ns BEFORE "CLK" TIMEGRP "AAA";

- Clock to Pad Requirements (OFFSET OUT AFTER)
  TIMEGRP "OUTPAD" OFFSET = OUT 20.000000 ns AFTER "CLK" TIMEGRP "NNN";
**Slow/Fast Path Exceptions (FROM TO)**

- Allows you to create a time specification that specifies an explicit maximum delay between groups of elements and to define intermediate points to which the time specification applies.

\[
\text{TIMESPEC } "\text{TS}_A" \text{ = FROM } "\text{FFS}" \text{ THRU } "\text{TTT}" \text{ TO } "\text{FFS}" \text{ 20.000000 ns;}
\]

**Multi-cycle Paths (FROM TO)**

- Allows you to create a time specification that designates the maximum delay between groups of elements and to select intermediate points to which the time specification applies.
- Unlike the Slow/Fast Path Exceptions constraint, Multi-cycle Paths is not explicit but relative to another time specification.

\[
\text{TIMESPEC } "\text{TS}_B" \text{ = FROM } "\text{FFS}" \text{ TO } "\text{FFS}" \text{ "TS}_A" \text{ * 1.000000;}
\]
False Paths (FROM TO TIG)

◆ Allows you to create a time specification designating that paths between selected groups and optionally selected intermediate points will be ignored for timing purposes.

\[ \text{TIMESPEC "TS_C" = FROM "FFS" TO "FFS" TIG;} \]

False Paths by Nets (NET TIG)

◆ Allows you to select certain nets in a time specification that will be ignored.

\[ \text{NET "F15OUT" TIG= TS_A, TS_B;} \]
Floorplanning

Example Design - Slot Decoder

- Slot comparators include BUFTs
  - Best aligned in columns
- All drive priority encode and mux
  - Best aligned in row above or below the comparators
Slot Decoder Floorplanning

◆ Structure difficult for automatic tools to recognize

<table>
<thead>
<tr>
<th></th>
<th>Ctr</th>
<th>CMP1 CLBs &amp; BUFTs</th>
<th>CMP2 CLBs &amp; BUFTs</th>
<th>CMP3 CLBs &amp; BUFTs</th>
<th>CMP4 CLBs &amp; BUFTs</th>
<th>CMP5 CLBs &amp; BUFTs</th>
<th>CMP6 CLBs &amp; BUFTs</th>
<th>CMP7 CLBs &amp; BUFTs</th>
<th>CMP8 CLBs &amp; BUFTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>µP</td>
<td>µP</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
<td>µP i/f CLBs</td>
</tr>
<tr>
<td>DATA</td>
<td>DATA</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
<td>Mux</td>
</tr>
<tr>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
<td>BUFTs&amp; I/O</td>
</tr>
</tbody>
</table>

Automatic vs. Manual Placement

◆ Automatic tools find best implementations for most designs
  • Optimizing speed, density, and runtime
◆ Problems with routing are usually caused by poor placement
◆ Floorplanning is interactive control of logic placement to improve implementation
  • Can improve speed or routability of difficult designs
◆ Floorplanning is the most effective type of manual intervention
  • Manual routing is not necessary
When to Floorplan

◆ **Best with more structured designs**
  - Harder for automatic tools, but easier for user

◆ **May also be required for:**
  - High utilization, especially of high-density devices
  - Specific architecture usage such as long lines or decode
  - Critical path optimization

When Not to Floorplan

◆ **Use floorplanning carefully**
  - Always initially translate design with no constraints
  - Use XACT-Performance or increase Effort Levels to improve implementation

◆ **Good floorplanning requires some knowledge of architecture to be better than automatic placement**
Methods of Floorplanning

◆ Schematic constraints
◆ Location constraints in .UCF file applied to place & route
◆ XACT EPIC (not covered in class)
  • Allows specific placement of blocks
  • Shows details of interconnect
  • Must create guide file or constraints to preserve
◆ XACT Floorplanner
  • Can specify absolute locations or ranges
  • Ratsnest-view instead of details of interconnect
  • Can run place & route directly from Floorplanner

Features of the Floorplanner

◆ Interacts at a high level of the design hierarchy, as well as with low-level elements such as I/Os, function generators, tristate buffers, flip-flops, and RAM/ROM
◆ Captures and imposes complex patterns, which is useful for repetitive logic structures such as interleaved buses
◆ Automatically distributes logic into columns or rows
◆ Uses dynamic rubberbanding to show the ratsnest connections
◆ Finds logic or nets by name or connectivity
◆ Permits design hierarchy rearrangement to simplify floorplanning
◆ Groups logic by connectivity or function
◆ Identifies placement problems in the Floorplan window
◆ Provides online help
Invoking the Floorplanner

- Select from Foundation Project Manager
  - Tools > Implementation > Floorplanner
- Select from Design Manager toolbox
  - Design will be mapped if not already available
Screen Components

◆ **Design window on left**
  - Shows design resources hierarchically
  - When window is selected, third menu is Hierarchy

◆ **Floorplan window on right**
  - Shows device resources
  - Shows placement of logic floorplanned in schematic
  - When window is selected, third menu is Floorplan

Using Design Window

◆ **Shows number of resources in each hierarchical block**
  - FG = 4-input lookup table, FGH = 3-input lookup table

◆ **Initial view is of lowest-level macros shown as stacks**
  - Each is color-coded
  - Use + to expand to primitive resources
  - Use - to contract to higher-level macros
Selecting Logic or Nets

- **Edit > Find... (Ctrl + F)**
  - Can find logic or nets by name, type, or connectivity

- **Edit > Select Loads, Edit > Select Sources**
  - Can select loads or sources of currently selected logic

Use Toolbar to Control Floorplan Window

- Ratsnest on/off
- Turn resource graphics on/off
- Turn text labels on/off
- Zoom in/out
- Full to area
- Full to selected
Analyzing Automatic Placement

- **View -> Placement**
  - Requires a placed design NCD file
- **Can see relative placement and utilization**
  - Closer placement usually means faster nets
    - Over-compacted design will be harder to route
    - Use View -> Congestion

---

Example of Designing for Interconnect

- Serial carry should be propagated on XC3000 direct interconnect
- Prescaler output should be distributed on long line for high speed and low skew
- See XAPP002 example
Spreading Out Logic

- Reduce congestion by spreading logic out over chip
  - While keeping distances short
- Requires selecting logic and moving it in floorplan window

Selecting Logic in Floorplan Window

- Click on resource to select it
- Double-click to select entire macro
- Selected logic flashes and shows ratsnest of connections
- Selected logic is reversed in design window
  - Boxed nodes indicate some selected logic is below it
Moving Logic in Floorplan Window

◆ Select a second time to attach logic to cursor
◆ Move cursor to desired drop point and select it
  • Circle with diagonal indicates illegal location
    – e.g., CLB resource on an IOB location
    – Resource definition changes (such as moving CLB flip-flops to IOBs) can only be done in schematic
◆ Ratsnest shown to all sources and loads of resource
  • Only if they are already placed in floorplan window

Selecting Logic in Design Window

◆ Select by clicking on name
◆ Attach to cursor by clicking on icon stack
  • Add-Select with middle mouse button or Ctrl key
  • Multiple select by dragging mouse
    – Both also work in Floorplan window
◆ Logic selected in design window is also selected in floorplan window, if located there
◆ Can move logic between windows
  • To place or unplace
Dropping a Stacked Macro

◆ A macro is dropped into Floorplan in columns by default
  • Each design resource goes into next available floorplan resource in column

Changing Drop Direction

◆ Toolbar icons allow dropping bottom to top, left to right, or right to left
  • Can change before selecting logic or while dragging
◆ Can change bit order or spacing
◆ Arrows attached to logic on cursor indicates current drop direction
Flipping Logic

- Align related functions
- Toolbar buttons reverse everything within selected logic

Advantages of Vertical Orientation

- Bidirectional data bus lines run horizontally
  - Enable lines run vertically
- Large registered functions align vertically
  - Clock lines run vertically
  - Most non-clock, non-BUFT long lines run vertically
  - Carry logic runs vertically
- XC3000 data flow is left-to-right
Using Clock Routing Effectively

- Use BUFG as first choice in schematic
- Consider floorplanning when using many clocks
  - XC3000: more than two
  - XC4000/XC5000: more than four
- Floorplanning requirements
  - Minimize number of clocks per column
  - Line up logic with common controls in one column

Carry Logic

- Carry logic in XC4000/XC5000 CLBs is propagated on special carry interconnect
- Runs vertically
  - Arithmetic functions must be aligned vertically
  - RPMs always run up the column (LSB at bottom)
Grouping Logic into CLBs

- Make sure connected logic and flip-flops are in same CLB
- Use default grouping to start
  - Let Flow Engine place logic to see automatic grouping
- Automatic tools tend to use more CLBs than required
  - Allows place & route to improve routing
  - Do not over-pack CLBs
- Group logic together only if it is to be placed together
- XC3000 does not allow re-grouping in Floorplanner
  - Use Pack Design for slight reduction in CLBs to fit in smaller device

CLB Mapping Control in Schematic

- Allows user to specify from schematic how logic is mapped into CLBs
- XC3000
  - CLBMap can specify entire CLB
- XC4000/XC5000
  - FMap specifies a function generator in a CLB
  - HMap specifies an XC4000 H function generator in a CLB
Placing Resources Individually

◆ Select toolbar button
  • Make sure Distribute mode is selected
  
  ◆ Place in any pattern desired

Capturing a Footprint

◆ Can copy any pattern from one module to another
  • Select placed logic
  • Capture pattern
  • Select logic with same resources (before or after placement)
  • Impose pattern
Allocating Larger Area than Minimum

- Select logic
- Select Assign Area toolbar button
- Drag over area to allocate to logic

Constraining to One Quadrant

- Most non-clock long lines can be split
  - Doubles signals on long lines and improves speed
  - Constrain logic to one quadrant to allow splitting
Floorplanning First

◆ Floorplan before auto place
◆ To see resource requirements
◆ To provide I/O placement requirements or other required placement
◆ Stop Flow Engine after mapping
  • Open MAP file into floorplanner

Floorplanning After Place & Route

◆ To analyze automatic placement
◆ Open MAP file into floorplanner
◆ Read placed NCD design file
Iterative Floorplanning

◆ Place & route part of design
  • Automatically updates floorplanner with placement
◆ Select another part of the design to place & route
  • Use guide option to maintain previous routing or allow tools to re-route to optimize across entire design

Analyzing Placement

◆ BUFTs should be vertically aligned and not cross longline splitters
◆ Clock enables should be vertically aligned
  • IOB driving it should be close to column
◆ RAM control signals should be aligned to use longlines
◆ RPMs, BUFTs, and other macros should have source and load close together
◆ Structured elements should not cross longline splitters
◆ Avoid over-constraining!
  • Prefer stack mode over distribute mode
Save Results

◆ File > Save
  - FNF
    - The FNF file is the Floorplanner’s database. It can be saved by using the File > Save command.
  - MFP
    - This file is generated when the FNF file is saved in the Floorplanner. It is used as an input to MAP to transfer physical constraints from the Floorplanner back to the automatic implementation tools.

Floorplanning via Schematic Location Constraints

◆ On flip-flops, Maps, pads, or macros containing them
◆ Can specify individual locations or ranges

(D Q)

(XC4000/XC5000) LOC=CLB_R2C4
(XC3000) LOC=BD
Assigning IOB Locations in Schematic

- Use LOC properties on IOB primitives

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin number from data sheet</td>
<td>LOC=P12</td>
</tr>
<tr>
<td>Place IOB on a side of the device:</td>
<td>LOC=L</td>
</tr>
<tr>
<td>T=top, L=left, B=bottom, R=right</td>
<td>LOC=BR</td>
</tr>
<tr>
<td>Place IOB on a half-edge:</td>
<td>LOC=L;LOC=T</td>
</tr>
<tr>
<td>One or another location:</td>
<td>LOC=L;LOC=T</td>
</tr>
</tbody>
</table>

Locking I/O Placement

- Avoid it until as late as possible in design
- Or until design is 75-80% complete & timing requirements met
  - Minimize changes to I/O-related logic
  - Then re-align I/Os in logical pattern with minimal movement
- More flexibility on board than inside FPGA
Lock Device Pins

◆ Allows you to include pinout information in the UCF file for use with other Xilinx implementation tools.
  • In Project Manager, Tools > Implementation > Lock Device Pins
  • In Design Manager, Design > Lock Pins…

◆ After the constraints are generated, you can review the Lock Pins report
  • In Project Manager, Tools > Implementation > View Locked Pins Report
  • In Design Manager, Utilities > Lock Pins Report

I/O Placement

◆ To partially control layout, specify edge locations only
  • Useful for partial designs or incremental design

◆ Avoid placing all related IOBs adjacent to each other
  • Congests routing resources in area

◆ Consider whether to use or prohibit dual-purpose configuration pins
I/O Connections to Long Lines

- Allow I/Os connecting to bidirectional buses to be on left or right side
  - Special connections from horizontal long line to I/O on sides
  - If floorplanning buses, place I/O directly to left or right of bus location
- If locking I/O, order from MSB to LSB with MSB at top
  - Consistent with RPMs

Floorplanning via RLOC Constraint

- Defines relative position instead of absolute
  - Not accessible in floorplanner
- XC4000/XC5000 CLBs only

```
RLOC= R0C0  RLOC= R0C1  RLOC= R0C2
RLOC= R1C0  RLOC= R1C1  RLOC= R1C2
RLOC= R2C0  RLOC= R2C1  RLOC= R2C2
```
Floorplanning via User Constraints File

- UCF file used by place & route for additional constraints beyond those in design file
  - Works with any design entry format
- UCF file created with a text editor
  - See documentation for syntax of constraints
  - Use report files for syntax of design

Summary

- Floorplanning is interactive grouping and placement of logic
- Floorplanning is the most important point where the user can help the automatic implementation tools
- Tools for floorplanning include:
  - Schematic mapping and location constraints
  - Relative location constraints in XC4000/XC5000 design
  - Location constraints in CST file
  - Graphical floorplanners
- Vertical orientation is best for most logic functions
- XC3000 has left-to-right data flow
Configuration

Bitstream Generator

Runs if Produce Configuration Data selected (default)
Configuration Options I

◆ Configuration Rate

- The XC4000 uses an internal configuration clock, CCLK, when configuring in a master mode. The configuration rate option allows you to select the rate for this clock. The following options are available. The default is Slow.
  - Slow
    - Select Slow to set the configuration clock rate to 1MHz.
  - Fast
    - Select Fast to set the configuration clock rate to 8 MHz.

Configuration Options II

◆ Threshold Levels

- Inputs: Default is TTL.
  - TTL(1.2V threshold) - specify TTL-compatible inputs.
  - CMOS(2.5V threshold) - specify CMOS-compatible inputs.
  - Read from Design - Select Read from Design to specify the TTL/CMOS input level included in the physical constraints (PCF) file.
- Outputs: Default is TTL.
  - This option is only available for XC4000 devices.
    - TTL
    - CMOS
    - Read from Design

◆ Configuration Pins

- TD0, M0, M1, M2, Done
  - Float, Pullup, Pulldown
Configuration Options III

◆ Perform CRC During Configuration
  • This option enables Cyclic Redundancy Checking (CRC) error checking during configuration.
  • If enabled, the software calculates a running CRC and inserts a unique four-bit partial check at the end of each data frame in the configuration bitstream.
  • If disabled, the device performs a simple check for the 0110 pattern at the end of each frame in the configuration data. By default, this option is on.

◆ Produce ASCII Configuration File
  • This option creates a rawbits (RBT) file in addition to the binary BIT file.
  • The RBT file is a text file that contains ASCII 1s and 0s.
  • These characters represent the actual bits in the configuration bitstream that are downloaded to the FPGA. By default, this option is off.

Choose a Configuration Method

◆ Configuration mode selected by control pins M0, M1, M2
◆ Configuration data can be supplied by:
  • Memory (e.g. PROM), Parallel Master Mode up incrementing address
  • Decrementing address Parallel Master Mode down
  • Serial memory (XC17xxx) Serial Master Mode
  • Processor (parallel) Peripheral Mode
  • Processor and clock (XC4000/XC5000) Synchronous Peripheral
  • Processor; byte-wide load (XC4000EX/XC5000) Express Mode
  • External master FPGA or processor (serial) Slave Mode
**Master Serial Mode**

- FPGA automatically loads itself from external serial PROM
  - Requires one small chip, few connections
  - Xilinx offers serial PROMs from 18K to 256K
- Internal CCLK clocks address counter in the serial memory

**Master Parallel Mode**

- FPGA automatically loads itself from external byte-wide PROM
  - Can use part of existing PROM
- Internal CCLK clocks address counter in FPGA
  - Up from 0 or Down from all 1’s, specified by mode
  - 16-18 bit address for large FPGA, or large daisy chain
- Data still gets serialized internally (configures at same rate)
Peripheral Mode

- FPGA loads under microprocessor control as a peripheral
- May not require extra connections if FPGA is already connected to a processor
  - Byte of data supplied by external controller
  - FPGA reads data when Chip Selects and Write asserted
  - FPGA Ready/Busy signal indicates ready for next byte

Synchronous Peripheral Mode

- XC4000/XC5000 only
- User generates configuration clock, CCLK
- Accepts byte-wide data
Express Mode

- XC4000X/XC5000 only
- Only mode that configures in parallel
- Byte loaded on each CCLK
  - CCLK max = 10 MHz
  - Equivalent to 80 MHz in other modes
  - Supports daisy chain if all XC5000 Express Mode

Slave Mode

- Simple two-wire interface
- FPGA reads one bit of data on each clock supplied externally
- Can be controlled by microprocessor or DMA controller
- Used by XChecker download cable
Daisy Chain

- Can program multiple devices with independent configurations from one PROM
- Control from lead FPGA in daisy chain
  - Lead FPGA can be in any configuration mode
  - Following FPGAs must be in slave mode
- Automatically created by PROM Formatter

![Daisy Chain Diagram]

XC4000/XC5000 Configuration Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>M2, M1, M0</th>
<th>CCLK</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Serial</td>
<td>0 0 0</td>
<td>output</td>
<td>Bit-Serial</td>
</tr>
<tr>
<td>Slave Serial</td>
<td>1 1 1</td>
<td>input</td>
<td>Bit-Serial</td>
</tr>
<tr>
<td>Master Parallel Up</td>
<td>1 0 0</td>
<td>output</td>
<td>Byte-Wide, increment from 00000</td>
</tr>
<tr>
<td>Master Parallel Down</td>
<td>1 1 0</td>
<td>output</td>
<td>Byte-Wide, decrement from 3FFFF</td>
</tr>
<tr>
<td>Peripheral Synchronous</td>
<td>0 1 1</td>
<td>input</td>
<td>Byte-Wide</td>
</tr>
<tr>
<td>Peripheral Asynchronous</td>
<td>1 0 1</td>
<td>output</td>
<td>Byte-Wide</td>
</tr>
<tr>
<td>Express</td>
<td>0 1 0</td>
<td>input</td>
<td>Byte-Wide</td>
</tr>
</tbody>
</table>
Master/Slave Serial Mode Circuit

Master Parallel Mode Circuit
Synchronous Peripheral Mode Circuit

Asynchronous Peripheral Mode Circuit
PROM File Formatter

- Translates bitstreams into PROM programmer files
- Supports standard ASCII HEX formats for programmers
  - Intel MCS (MCS)
  - Motorola Exormax (EXO)
  - Tektronix Hex (TEK)

In Project Manager
Tools > Device Programming > PROM File Formatter

PROM Formatter Screen

- Current revision’s bitstream is loaded automatically
PROM Properties

- File -> PROM Properties

<table>
<thead>
<tr>
<th>PROM File Format</th>
<th>Number of Addresses (Bytes)</th>
<th>AddressRange</th>
<th>Total Number of Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEKHEX</td>
<td>65 536</td>
<td>0000:FFFF</td>
<td>524 288</td>
</tr>
<tr>
<td>MCS-86</td>
<td>1 048 576</td>
<td>00000:FFFFF</td>
<td>8 388 608</td>
</tr>
<tr>
<td>EXORmacs</td>
<td>16 777 216</td>
<td>000000:FFFFFFF</td>
<td>134 217 728</td>
</tr>
<tr>
<td>HEX</td>
<td>No limit</td>
<td>No limit</td>
<td>No limit</td>
</tr>
</tbody>
</table>
**Tie Off Unused Interconnect**

- Tie option ties off all unused interconnect to logic 0
- Tie is highly recommended for production
  - Decreases power consumption by a few mA
  - Decreases on-chip noise

**Consider Configuration Pins in Design**

- M0, M1, M2 determine configuration mode
  - Some can be used as I/O after configuration
- CCLK is configuration clock
- DONE goes high at end of configuration
- PROGRAM can be pulled low to re-start configuration
- LDC is Low During Configuration; could use as PROM enable
- INIT goes high after initial clearing of configuration
  - Goes back low if CRC error occurs
- Avoid contention between configuration and logic functions
Configuration Sequence

- I/O disabled
- HDC=High
- LDC=Low

INIT Signal Low

- Power-On Time Delay
- Clear Configuration Memory
- Configuration Program Mode
- Start-Up
- Operational Mode

Low on PROGRAM

Configuration Debugging

- Check VCC and rise time
- Check configuration pins and contention
- Check for clean connections between FPGA and source
- Watch timing requirements in peripheral and slave modes
- Examine DOUT and CCLK outputs even if not used
- Try XChecker cable or different FPGA
- Debug daisy chain one at a time, in sequence order
  - Put latest family first
“INIT does not go high” or “DONE does not go high” I

◆ Verify that you compiled your design for the correct part type and package.

◆ Make sure that you set the target FPGA for the serial slave mode. The xchecker only downloads in Slave Serial. The mode pins <M0 M1 M2> on the FPGA should be set to <111>

◆ Check all connections for slave serial. Remember that connections to the FPGA differ slightly across the different families.

◆ Check the bitstream options in makebits or bitgen and verify that you selected a pull-up for the DONE pin.

◆ Check for noisy signals that may corrupt the data or CCLK going to the target FPGA.

“INIT does not go high” or “DONE does not go high” II

◆ Read the "Hardware & Peripherals", chapter 3, for information on how to properly connect the xchecker to the target FPGA. Available at ftp://ftp.xilinx.com/pub/documentation/xactstep6/hardware.pdf

◆ For further debugging, read the app note, "FPGA Configuration Guidelines". Available at http://www.xilinx.com/xapp/xapp090.pdf
FPGA Demoboard Layout

To Learn More…
Check Hardware User Guide

FPGA Demoboard I

- One socket for an XC3000 device
- One socket for an XC4000 device
- One 17XX socket for each FPGA
- An XChecker/Download cable header for each FPGA
- Daisy-chain configuration with the XC4000 device at the head of the chain
- 8 DIP switches to set up the XC4000 and XC3000 FPGAs
- 16 I/O lines that connect the two FPGAs
- An external relaxation oscillator for the XC3000
- The XC4000 OSC4 library symbol, uses pin 19 of the XC4003E to drive the XC3000 TCLKIN on pin 11 of the XC3020A
- The XC4000 OSC4, uses pin 13 to drive the XC3000 alternate clock buffer (BCLKIN) on pin 43
FPGA Demoboard II

- 8 DIP switches set logic input levels; switch outputs drive both FPGAs; closing switches drive signals to logic 1's
- Program, Reset, and Spare Pushbutton switches, which are common to both FPGAs
- XC3000 displays that use eight LED bars in one row and one 7-segment LED, shown in the "FPGA Demonstration Board Displays" figure
- XC4000 displays that use eight LED bars in one row and two 7-segment LEDs, shown in the "FPGA Demonstration Board Displays" figure
- Space for an optional +5 V regulator for battery operation
- Space for an optional crystal oscillator
- Headers for FPGA probe points
- Prototype area on PC board

### Configuring the XC4003E from the XChecker/Download Cable

<table>
<thead>
<tr>
<th>Switch</th>
<th>Name</th>
<th>Position</th>
<th>Switch</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1-1</td>
<td>INP</td>
<td>X</td>
<td>SW2-1</td>
<td>PWR</td>
<td>X</td>
</tr>
<tr>
<td>SW1-2</td>
<td>MPE</td>
<td>X</td>
<td>SW2-2</td>
<td>MPE</td>
<td>OFF</td>
</tr>
<tr>
<td>SW1-3</td>
<td>SPE</td>
<td>X</td>
<td>SW2-3</td>
<td>SPE</td>
<td>OFF</td>
</tr>
<tr>
<td>SW1-4</td>
<td>M0</td>
<td>X</td>
<td>SW2-4</td>
<td>M0</td>
<td>ON</td>
</tr>
<tr>
<td>SW1-5</td>
<td>M1</td>
<td>X</td>
<td>SW2-5</td>
<td>M1</td>
<td>ON</td>
</tr>
<tr>
<td>SW1-6</td>
<td>M2</td>
<td>X</td>
<td>SW2-6</td>
<td>M2</td>
<td>ON</td>
</tr>
<tr>
<td>SW1-7</td>
<td>MCLK</td>
<td>OFF</td>
<td>SW2-7</td>
<td>RST</td>
<td>X</td>
</tr>
<tr>
<td>SW1-8</td>
<td>DOUT</td>
<td>OFF</td>
<td>SW2-8</td>
<td>INIT</td>
<td>OFF</td>
</tr>
</tbody>
</table>
### Configuring the XC3020A from the XChecker/Download Cable

<table>
<thead>
<tr>
<th>Switch</th>
<th>Name</th>
<th>Position</th>
<th>Switch</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1-1</td>
<td>INP</td>
<td>X</td>
<td>SW2-1</td>
<td>PWR</td>
<td>X</td>
</tr>
<tr>
<td>SW1-2</td>
<td>MPE</td>
<td>OFF</td>
<td>SW2-2</td>
<td>MPE</td>
<td>X</td>
</tr>
<tr>
<td>SW1-3</td>
<td>SPE</td>
<td>OFF</td>
<td>SW2-3</td>
<td>SPE</td>
<td>X</td>
</tr>
<tr>
<td>SW1-4</td>
<td>M0</td>
<td>ON</td>
<td>SW2-4</td>
<td>M0</td>
<td>X</td>
</tr>
<tr>
<td>SW1-5</td>
<td>M1</td>
<td>ON</td>
<td>SW2-5</td>
<td>M1</td>
<td>X</td>
</tr>
<tr>
<td>SW1-6</td>
<td>M2</td>
<td>ON</td>
<td>SW2-6</td>
<td>M2</td>
<td>X</td>
</tr>
<tr>
<td>SW1-7</td>
<td>MCLK</td>
<td>OFF</td>
<td>SW2-7</td>
<td>RST</td>
<td>X</td>
</tr>
<tr>
<td>SW1-8</td>
<td>DOUT</td>
<td>OFF</td>
<td>SW2-8</td>
<td>INIT</td>
<td>OFF</td>
</tr>
</tbody>
</table>

### FPGA Board General Components

- **RESET Pushbutton (SW4)**
  - When you press the RESET pushbutton it can apply an active-Low Reset signal to the FPGAs and configuration PROMs, depending on how the Reset signal routing is configured.

- **SPARE Pushbutton (SW5)**
  - The SPARE pushbutton applies an active-Low signal to the XC3020A on pin 16, and to the XC4003E on pin 18.

- **PROG Pushbutton (SW6)**
  - The PROG pushbutton applies an active low signal to the DONE/PROGRAM input on the XC3020A FPGA socket at pin 45 and to the PROGRAM input on the XC4003E FPGA socket at pin 55.
LED Indicators

<table>
<thead>
<tr>
<th></th>
<th>XC3020APin</th>
<th></th>
<th>XC4003EPin</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>37</td>
<td>D9</td>
<td>61</td>
</tr>
<tr>
<td>D2</td>
<td>36</td>
<td>D10</td>
<td>62</td>
</tr>
<tr>
<td>D3</td>
<td>41</td>
<td>D11</td>
<td>65</td>
</tr>
<tr>
<td>D4</td>
<td>33</td>
<td>D12</td>
<td>66</td>
</tr>
<tr>
<td>D5</td>
<td>32</td>
<td>D13</td>
<td>57</td>
</tr>
<tr>
<td>D6</td>
<td>31</td>
<td>D14</td>
<td>58</td>
</tr>
<tr>
<td>D7</td>
<td>28</td>
<td>D15</td>
<td>59</td>
</tr>
<tr>
<td>D8</td>
<td>29</td>
<td>D16</td>
<td>60</td>
</tr>
</tbody>
</table>

7-Segment I/O Connections

<table>
<thead>
<tr>
<th>Segment</th>
<th>XC3020A</th>
<th>XC4003E</th>
<th>XC4003E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display U6</td>
<td>U7</td>
<td>U8</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>38</td>
<td>39</td>
<td>49</td>
</tr>
<tr>
<td>b</td>
<td>39</td>
<td>38</td>
<td>48</td>
</tr>
<tr>
<td>c</td>
<td>40</td>
<td>36</td>
<td>47</td>
</tr>
<tr>
<td>d</td>
<td>56</td>
<td>35</td>
<td>46</td>
</tr>
<tr>
<td>e</td>
<td>49</td>
<td>29</td>
<td>45</td>
</tr>
<tr>
<td>f</td>
<td>53</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>g</td>
<td>55</td>
<td>44</td>
<td>51</td>
</tr>
<tr>
<td>decimal point</td>
<td>30</td>
<td>37</td>
<td>41</td>
</tr>
</tbody>
</table>
## Eight General-Purpose Input Switches (SW3)

<table>
<thead>
<tr>
<th>Switch</th>
<th>XC3020A</th>
<th>XC4003E</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW3-1</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>SW3-2</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>SW3-3</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>SW3-4</td>
<td>17</td>
<td>24</td>
</tr>
<tr>
<td>SW3-5</td>
<td>19</td>
<td>25</td>
</tr>
<tr>
<td>SW3-6</td>
<td>21</td>
<td>26</td>
</tr>
<tr>
<td>SW3-7</td>
<td>23</td>
<td>27</td>
</tr>
<tr>
<td>SW3-8</td>
<td>24</td>
<td>28</td>
</tr>
</tbody>
</table>
Use XChecker Cable to Simplify Verification

- **Downloading allows quick verification of design in circuit**
  - Bitstream downloaded via computer’s serial port directly into FPGA
  - No PROM programming required
  - Design changes and verifications made quickly

- **Readback sends configuration data and flip-flop values back out of chip**
  - Verifies correct configuration
  - Allows in-circuit “probing” of all signals
  - Can occur while the FPGA is running
  - Uses no CLBs or routing resources
Readback Operation

- **Readback Trigger input starts readback operation**
  - Connect to RT on XChecker cable
- **Internal configuration clock (CCLK) drives readback logic**
- **Data is sent out serially on Readback Data pin**
  - Connect to RD on XChecker cable
- **Trig and Data are on dedicated pins on XC3000**
  - XC4000/XC5000 allow user programming of pin locations

---

Enabling Configuration Readback I

- **XC3000 controlled via Bitstream Generator**
  - Default is enabled
  - Data and trigger connected to Mode pins
- **XC4/5000 controlled via schematic and Bitstream Generator**
  - Include Readback symbol in schematic
  - Connect TRIG and DATA to I/O pins
  - Can use MD0 and MD1
Enabling Configuration Readback II

**XC4/5000 READBACK controlled via Verilog**

```verilog
module debug_ckt (DEBUG_CLK_8M, SYS_CLK_15HZ, GSRT, EXT_CLK, CLK_SEL, CLK);

input DEBUG_CLK_8M;
input SYS_CLK_15HZ;
input GSRT;
input EXT_CLK;
input CLK_SEL;
output CLK;

wire nGSRT;
wire TRIG, intTRIG;
wire DATA, intDATA;
reg q_CLK_SEL;
reg mux_out;
reg CLK;

assign nGSRT = !GSRT;
STARTUP START_INST(.GSR(nGSRT));
READBACK READ_INST(.TRIG(intTRIG), .DATA(intDATA));
IBUF TRIG_BUF(.I(TRIG), .O(intTRIG));
OBUF DATA_BUF(.I(intDATA), .O(DATA));
MD0 TRIG_PAD(.I(TRIG));
MD1 DATA_PAD(.O(DATA));
```

Enabling Configuration Readback III

**XC4/5000 READBACK controlled via VHDL**

```vhdl
component READBACK
port (TRIG: in std_logic;
      DATA: out std_logic);
end component;

component IBUF
port (I: in std_logic;
     O: out std_logic);
end component;

component OBUF
port (I: in std_logic;
     O: out std_logic);
end component;

component MD0
port (I: out std_logic);
end component;

component MD1
port (O: in std_logic);
end component;

begin
nGSRT<=not GSRT;
START_INST: STARTUP port map (GSR=>nGSRT);
READ_INST: READBACK port map (TRIG=>intTRIG, DATA=>intDATA);
TRIG_BUF: IBUF port map (I=>TRIG, O=>intTRIG);
DATA_BUF: OBUF port map (I=>intDATA, O=>DATA);
TRIG_PAD: MD0 port map (I=>TRIG);
DATA_PAD: MD1 port map (O=>DATA);
```

Enabling Flip-Flop Readback

**XC3000**
- always enabled with configuration readback

**XC4/5000**
- leave default configuration template option Capture CLB and IOB Outputs When TRIG Goes Active

### Available Readback Data

**Data includes all storage elements in device**
- XC4000/XC5000 readback data includes all outputs of CLBs and IOBs

**XC4000/XC5000 data is captured when readback is triggered**

**XC3000 data is captured as readback progresses**
- May want to stop system clock for logic verification
- Requires XChecker control of system clock
XChecker Cable Features

- Controlled by Hardware Debugger
- Download to single FPGA or daisy chain
  - Typically used during design and development
  - Design prototype system for XChecker cable connection
- Readback from single FPGA
  - Verification of bitstream
  - Logic verification

Cable Setup

- Connect cable to computer’s serial port
  - Power up cable via target’s VCC and GND
- Hardware Debugger should find cable automatically
- Cable -> Communications... allows change to port
Downloading a Design

- Connect cable to target VCC, GND, PROG, DONE, DIN, CCLK
- Put device into slave mode
- Select Download -> Download Design or the “lightning” toolbar icon
- Can verify configuration with Download -> Verify Bitstream or “checkmark” toolbar icon
  - Requires configuration readback to be enabled

Use Synchronous Debugging

- Debugging session allows readback of logic state
- Selected from Debug menu or by selecting toolbar icon (“Bug” with “clock”)

- Synchronous mode controls system clock via CLKO
  - Clock stops between each readback (required in XC3000)
  - Clock source is internal cable clock or external system clock via CLKI
Control Panel Defines Debug Session

◆ **Readback Control Panel**
  - View > Control Panel

◆ **Allows direct control of:**
  - System clock source definition and application
  - Readback trigger source definition and application
  - Number of readbacks
  - Display options

Use XChecker or System as Clock Source

◆ Can use one of four clock speeds in cable, or route system clock into cable for user control
Specify When Readback is to be Triggered

- Can trigger on external event
- Specify number of clocks between multiple readbacks

Choose Signals to Display

- Can define groups first with Groups... button
- Can select flip-flops in XC3000, RAM bits in XC4000, and combinatorial outputs in XC4000/XC5000
  - Filter for desired signals
Activating Readback

- Select Read button in Control Panel
- Waveform opens automatically
  - Note waveform reflects several steady state conditions, not timing
Xilinx Foundation Lab

Fibonacci Generator

0, 1, 1, 2, 3, 5, 8, d, 15, 22, 37, 59, ..... 

Lab 1 - New a Project

File > New Project

依下圖填入相關資料
Lab 2 - myor8 I

- 用基本的邏輯閘兜電路
- Click Schematic Editor Icon
- File > Scratchpad
- 繪出下圖

```
10
11
12
13
14
15
16
17
```

Lab 2 - myor8 II

- File > Save as ... (myor8)
- 更改電路圖紙張大小
  - File > Page Setup ...

![Page Setup Screenshot](image)
Lab 2 - myor8 III

◆ 產生 Symbol
  • Hierarchy > Create Macro Symbol From Current Sheet

Lab 3 - HEX2LED I

◆ 撰寫 VHDL 來產生七段顯示器的解碼電路
◆ 回到 Project Manager, 點選 HDL Entry Icon
◆ 編寫新的 VHDL 電路
  • 選擇 “Use HDL Design Wizard”
  • Language 處選 “VHDL”
  • Name 處填入 “HEX2LED”
  • Port 處填入如右圖 — 記得按 “Advance …”, 確認其為 STD_LOGIC_VECTOR
Lab 3 - HEX2LED II

- 修改VHDL
  - 將游標移至 `<<enter your statements here>>` 的下一行
  - Tools > Language Assistant > Synthesis Template > HEX2LED Converter > Use
  - VHDL內容如下頁所示

- 設定 Configuration
  - Synthesis > Configuration

- 檢查語法
  - Synthesis > Check Syntax

Lab 3 - HEX2LED III

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity HEX2LED is
  port (
    HEX: in STD_LOGIC_VECTOR (3 downto 0);
    LED: out STD_LOGIC_VECTOR (6 downto 0)
  );
end HEX2LED;

architecture HEX2LED_arch of HEX2LED is
begin
  -- <<enter your statements here>>
  -- HEX-to-seven-segment decoder
  -- HEX: in STD_LOGIC_VECTOR (3 downto 0);
  -- LED: out STD_LOGIC_VECTOR (6 downto 0);
  -- segment encoding
  -- 0
  -- 5
  -- 4 | 1
  -- --- < 6
  -- 3

  with HEX SELect
  LED<=
"1111001" when "0001", --1
"0100100" when "0010", --2
"0110000" when "0011", --3
"0011001" when "0100", --4
"0010010" when "0101", --5
"0000010" when "0110", --6
"1111000" when "0111", --7
"0000000" when "1000", --8
"0010000" when "1001", --9
"0001000" when "1010", --A
"0000011" when "1011", --B
"1000101" when "1100", --C
"0100001" when "1101", --D
"0000110" when "1110", --E
"0001110" when "1111", --F
"1000000" when others; --0

end HEX2LED_arch;
```
Lab 3 - HEX2LED IV

◆ 設定合成選項 (用內定值)
  - Synthesis > Options
◆ 自動合成，並產生Symbol
  - Project > Create Macro

![FPGA Express Options](image)

Lab 4 - scount8 I

◆ 利用State Editor來製作 8 bits counter - scount8
◆ 回到Project Manager, 點選 State Editor Icon
◆ 建立State Machine的架構
  - 選擇 "Use HDL Design Wizard"
  - Language 處選 "Verilog"
  - Name 處填入 "scount8"
  - Port 處填入如右圖
    - 填Output時，記得按 "Advance ...", 改成 Registered
Lab 4 - scount8 II

- The number of machines 选 one
- 依下圖繪製State Diagram
- 觀察相對應的 Verilog code
  - Synthesis > HDL Code Generation

Lab 4 - scount8 III

SCOUNT8

#diagram ACTIONS

- RESET
- ~CLK
- ~CE

Sreg0
Lab 4 - scount8 IV

- 設定合成選項
  - Synthesis > Options
- 自動合成，並產生Symbol
  - Project > Create Macro

![FPGA Express Options](image)

Lab 5 - ROM16x7

- 利用LogiBLOX的ROM來產生七段顯示器的解碼電路
- 從Schematic Editor中開啓LogiBLOX
  - Options > LogiBLOX...
- 依下圖填入適當資料，然後選 OK

```
rom16x7.mem 內容

depth 16
width 7
radix 2
data
1000000,
1111001,
0100100,
0110000,
0011001,
0010010,
0000010,
1011000,
0000000,
0010000,
0000100,
0000011,
1000110,
0100001,
0000110,
0001110
```
Lab 6 - 建立Multi Sheet的Top Schematic I

◆ 建立第一張Top Schematic
  - File > Open > Browse (MYFIB1.SCH)
  - 依下頁電路圖完成電路
  - 存檔, File > Save
  - 檢查MYFIB1.SCH是否加入Project中，若否，則在Project Manager中選Document > Add... > MYFIB1.SCH

◆ 建立第二張Top Schematic
  - File > New (MYFIB2.SCH)
  - 依下圖完成電路
  - 存檔, File > Save
  - 檢查MYFIB2.SCH是否加入Project中，若否，則在Project Manager中選Document > Add... > MYFIB2.SCH
Lab 6 - 建立Multi Sheet的Top Schematic III

L2, L4 (OUT)

L3, L5 (PAD)

Lab 6 - 建立Multi Sheet的Top Schematic IV

MYFIB2.SCH
Lab 7 - Simulate MYFIB I

◆ 在Schematic Editor中點選simulation toolbox - SC Probes，然後就出現 SC Probes

◆ 點選 SC Probes 左上角的 Probe tools，然後再用滑鼠點選所要觀察的 Net Name或Bus Name
  • 請選 ENABLE, CLK, RESET, FIB, PREV, NEXT, FIBCC, P_DISPA, P_DISPB, P_LED

◆ 點選 SC Probes 上面的 Simulator - SC Probes，經過轉換，即出現 Logic Simulator

Lab 7 - Simulate MYFIB II

◆ 喊出Stimulator Selection，然後給與inputs nets不同的輸入訊號
  • 點選或選擇Signal > Add Stimulator叫出Stimulator Selection
  • 用Stimulator Selection設ENABLE的訊號為q (由鍵盤按鍵q來控制訊號為high或low),
    CK的訊號為B1, RESET的訊號為CS
  • CS須由使用者自定訊號，請選Waveform > Edit叫出Test Vector State Selection再用其
    編輯訊號，請參考下頁之圖

◆ 利用Stimulator進行模擬
  • 點Simulation Step button表示模擬一小短段時間
  • 可改變Simulation Step的時間長短

Simulation Step

10ns
Lab 7 - Simulate MYFIB III

Lab 8 - Simulation from Macros I

1. 利用 Macro 以 Batch 方式進行模擬
2. 編輯 Command file
   - Tools > Script Editor...(test.cmd)
   - 請參考右圖
3. 執行 Command file in Logic Simulator
   - 在執行 Macro 前必須先清除之前的訊號
   - Signal > Delete Signals > All
   - File > Run Script File... (test.cmd)
Lab 8 - Simulation from Macros II

Run Script 的結果

Lab 8 - Simulation from Macros III
Lab 9 - Implementation I

◆ Click Project Manager 中的 “Implementation” Button
◆ 觀察 Device 與 Speed 是否正確
◆ 選 Options...
  • 進入Implementation Template
  • 勾選 Timing Report 中的 Produce Logic Level Timing Report
  • 確定
  • OK
  • Run

Lab 9 - Implementation II

◆ 執行完 Implement 之後請選 Reports，觀察結果
  • 觀察 Map Report
    Design Summary
    --------------------------
    Number of errors: ___
    Number of warnings: ___
    Number of CLBs: ____ out of 100 23%
    CLB Flip Flops: ___
    4 input LUTs: ___
    3 input LUTs: ___(___ used as route-throughs)
    16X1 ROMs: ___
    Number of bonded IOBs: ____ out of 61 40%
    IOB Flops: ___
    IOB Latches: ___
    Number of RPM macros: ___
    Total equivalent gate count for design: ______
    Additional JTAG gate count for IOBs: ______
Lab 9 - Implementation III

- **Observation Logic Level Timing Report**
  Design statistics:
  - Minimum period: _____ ns (Maximum frequency: _____ MHz)
  - Maximum net delay: ______ ns

- **Observation Place & Route Report**
  Device utilization summary:
  - Number of External IOBs: _____ out of 61 _____%
  - Flops: ______
  - Latches: ______
  - Number of CLBs: _____ out of 100 _____%
  - Total CLB Flops: _____ out of 200 _____%
  - 4 input LUTs: _____ out of 200 _____%
  - 3 input LUTs: _____ out of 100 _____%
  - The Score for this design is: _____

- **Observation Post Layout Timing Report**
  Design statistics:
  - Minimum period: _____ ns (Maximum frequency: _____ MHz)
  - Maximum net delay: ______ ns

Lab 10 - Timing Analysis I

- **Calling Timing Analyzer**
  - In Project Manager, click Timing analyzer Button

- **Analysis Results and Observation**
  - Click Tool Bar’s button
    Design statistics:
    - Minimum period: _____ ns (Maximum frequency: _____ MHz)
    - Maximum net delay: ______ ns

- **Utilizing Filter to Select Specific Path**
  - Path Filters > Custom Filters > Select Sources...
  - Path Filters > Custom Filters > Select Destinations...
  - Analyze > Custom
  - Please refer to the next page for selecting Source and Destination (From PREV[0] To FIB[0])
  - Maximum Delay is _____ ns.
Lab 10 - Timing Analysis II

Lab 11 - User Constraints I

◆ 利用 User Constraint File 設定 Pad Location
  • 在 Project Manager click “myfib.ucf”
  • 修改 myfib.ucf, 請加入下面 constraints 至檔案中
  • 重新執行 Implementation
  • 觀察 Place & Route Report, Pads 位置是否與設定相同
  • 利用 Timing Analyzer 或 Post Layout timing Report 觀察 timing

Design statistics:
  Minimum period: ___________ ns (Maximum frequency: ________ MHz)
  Maximum net delay: ___________ ns

NET ENABLE LOC=P19;
NET CLK LOC=P18;
NET RESET LOC=P20;
NET P_DISPA<0> LOC=P39;
NET P_DISPA<1> LOC=P38;
NET P_DISPA<2> LOC=P36;
NET P_DISPA<3> LOC=P35;
NET P_DISPA<4> LOC=P34;
NET P_DISPA<5> LOC=P33;
NET P_DISPA<6> LOC=P32;
NET P_DISPA<7> LOC=P31;
NET P_DISPB<0> LOC=P47;
NET P_DISPB<1> LOC=P46;
NET P_DISPB<2> LOC=P45;
NET P_DISPB<3> LOC=P50;
NET P_DISPB<4> LOC=P51;
NET P_LED<0> LOC=P60;
NET P_LED<1> LOC=P59;
NET P_LED<2> LOC=P58;
NET P_LED<3> LOC=P57;
NET P_LED<4> LOC=P56;
NET P_LED<5> LOC=P55;
NET P_LED<6> LOC=P54;
NET P_LED<7> LOC=P53;
NET P_DISPB<0> LOC=P49;
NET P_DISPB<1> LOC=P48;
Lab 11 - User Constraints I

◆ 利用 User Constraint File 設定 Timing Constraints
  • 在 Project Manager 点擊 "myfib.ucf"
  • 修改 myfib.ucf，請加入下面 constraints 至檔案中
  • 重新執行 Implementation
  • 利用 Timing Analyzer 或 Post Layout timing Report 觀察 timing

Design statistics:
  Minimum period: _______ ns (Maximum frequency: ______ MHz)
  Maximum path delay from/to any node: ______ ns

TIMEGRP FA = FFS(PREV<0>) ;
TIMEGRP FB = FFS(FIB<0>);

TIMESPEC TSA = FROM : FA : TO : FB : 26 ns;
NET CLK_B PERIOD=28.5 ns;

Lab 12 - Timing Simulation

◆ 執行 Project Manager 中的 Timing Simulation
◆ 依 Function Simulation 的 Macro 方式模擬
  • Tools > Script Editor... (test.cmd)
  • 修改 step 20ns 成 step 15ns，然後存檔成 test2.cmd
  • File > Run Script File...(test2.cmd)
Lab 13 - Configuration I

◆ 连接硬体
  - 将 Xchecker Cable 的端接到电脑的 COM2
  - 将 Xchecker Cable 的另一端利用排线接至 FPGA Demoboard 上的 J2
  - 将电池装上, 并将电池盒上的电源接至 Demoboard 上的 J9, 請注意正负
    及的位置, 紅線為正極, 接於 +5V 的字样位置。

◆ 调整 configuration mode
  - SW1-7, SW1-8 調成 OFF, 其餘 don't care
  - SW2-4, SW2-5, SW2-6 調成 ON, SW2-2, SW2-3, SW2-8 調成 OFF, 其餘 don't care

◆ 開啓 Hardware Debugger
  - Click PROGRAMMING icon.
  - 選擇 Hardware Debugger
  - Click OK

Lab 13 - Configuration II

◆ Communication Setup
  - Cable > Communication…
  - 將設定設成右圖
  - Click OK

◆ 下載電路
  - Download > Download Design

◆ 操作電路
  - 將 SW3-1(enable), SW3-2(reset) 設成 HI
  - 按 SPARE/SW5(clk) 即可操作電路

◆ ReadBack
  - Demo