Altera PC

July 2001
Course Outline - 1

- Introduction to PLD
- Altera Device Families
- Design Flow & Altera Tools
- Getting Started
- Graphic Design Entry
- Text Editor Design Entry
- Waveform Design Entry
Course Outline - 2

◆ Design Implementation
◆ Project Verification
  • Functional Simulation
  • Timing Analysis
  • Timing Simulation
◆ Device Programming
◆ Summary & Getting Help
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
- ...

Diagram showing circuit design with annotations.
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 field-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 a computer connected to an FPGA or CPLD device with a download cable]
Rapid Prototyping

◆ Reduce system prototyping time:
  • You can see the “real” things
    - In-circuit design verification
  • Quick delivery instead of IC manufacture
  • No test development, no re-spin potential (i.e. no NRE cost)
  • Satisfied for educational purposes

◆ Fast time-to-market
  0. Design, simulation, & compilation
  1. Downloading configuration bitstream
  2. Entering input data
  3. Obtaining output data
  4. Analysis

FPGA/CPLD is on the board!
Software Environment

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

◆ 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
<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>

Good

Excellent
Altera & CIC

◆ Altera
  • One of the world leaders in high-performance & high-density PLDs & associated CAE tools
  • Supports university program in Taiwan via CIC

◆ From CIC, you can apply:
  • Altera software - *it’s free for educational purpose!*
    - PC : MAX+PLUS II (full design environment)
    - WS : MAX+PLUS II (full design environment)
    Synopsys interface (Cadence & Viewlogic interfaces are optional)
  • Altera hardware -
  • University Program Design Laboratory Package (since 9709):
    • UP1 Education Board
    • ByteBlaster download cable
    • Student Edition Software
  • Of course, CIC is responsible for technical supports
  • WWW: http://www.cic.edu.tw/chip_design/design_intr/altera/
Altera Device Families

MAX
- MAX 7000B
- MAX 7000AE
- MAX 7000S
- MAX 9000A

FLEX
- FLEX 10KE
- FLEX 10KA
- FLEX 10K
- FLEX 6000
- FLEX 8000

APEX
- APEX 20K
- APEX 20KE
Altera Device Families

Altera offers 7 device families

<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>Classic</td>
<td>EPROM</td>
<td>SOP</td>
<td>200 ~ 900</td>
<td>EP610, 910, 1810</td>
</tr>
<tr>
<td>MAX 5000</td>
<td>EPROM</td>
<td>SOP</td>
<td>800 ~ 3,200</td>
<td>EPM5032, 064, 128, 130, 192</td>
</tr>
<tr>
<td>FLEX 6000(1)</td>
<td>SRAM</td>
<td>LUT</td>
<td>10,000 ~ 24,000</td>
<td>EPF6016/A, 024A</td>
</tr>
<tr>
<td>FLEX 8000A</td>
<td>SRAM</td>
<td>LUT</td>
<td>2,500 ~ 16,000</td>
<td>EPF8282A, 452A, 636A, 820A, 1188A, 1500A</td>
</tr>
<tr>
<td>MAX 9000/A(1)</td>
<td>EEPROM</td>
<td>SOP</td>
<td>6,000 ~ 12,000</td>
<td>EPM9320/A, 400/A, 480/A, 560/A</td>
</tr>
<tr>
<td>FLEX 10K/A/B(1)</td>
<td>SRAM</td>
<td>LUT</td>
<td>10,000 ~ 100,000</td>
<td>EPF10K10/A, 20/A, 30/A, 40/A, 50/V/A, EPF10K7/V/A, 100/A, 130/V/A, 250A</td>
</tr>
</tbody>
</table>

Note:
(1) Not all devices are currently available.
(2) Altera plans to ship new MAX7000A family in the near future.
Device Part Numbers

◆ **EPM7128STC100-7**
  - EPM = Family Signature (Erasable Programmable MAX device)
  - 7128S = Device type (128 = number of macrocells)
  - T = Package type (L = PLCC, T = TQFP...)
  - C = Operating temperature (Commercial, Industrial)
  - 100 = Pin count (number of pins on the package)
  - -7 = Speed Grade in nsec
  - Suffix may follow speed grade (for special device features)

◆ **Another Example:**
  - EPM7064SLC44-5
    - EPM7064S in a commercial-temp, 44 pin PLCC package with a 5 ns speed grade
MAX & FLEX Architectures

**MAX architecture**

- Parallel Logic Expanders (from other MCs)
- Product-Term Select Matrix
- Shared Logic Expanders
- 36 Programmable Interconnect Signals
- 16 Expander Product Terms
- Carry-In Cascade-In
- Carry-Out Cascade-Out
- DATA1, DATA2, DATA3, DATA4
- LABCTRL1, LABCTRL2
- LABCTRL3, LABCTRL4
- Clock Select
- Clear/Reset Logic
- Clock/Enable Select
- to I/O Control Block
- to PIA
- LE Out
- Programmable Register
- Register Bypass

**FLEX architecture**

- Look-Up Table (LUT)
- Carry Chain
- Cascade Chain
- Clear/Preset Logic
- Clock Select
- Carry-Out Cascade-Out
- LE Out
- Programmable Register
Choose the appropriate architecture

- Different PLD architectures provide different performance & capacity results for same application

<table>
<thead>
<tr>
<th>Feature</th>
<th>MAX Architecture</th>
<th>FLEX Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Building Block</td>
<td>Course Grain</td>
<td>Fine Grain</td>
</tr>
<tr>
<td>Logic Cell Structure</td>
<td>SOP</td>
<td>LUT</td>
</tr>
<tr>
<td>Technology</td>
<td>EEPROM</td>
<td>SRAM</td>
</tr>
<tr>
<td>Optimization</td>
<td>Combinational-Intensive Logic e.g. Large Decoders, State Machines</td>
<td>Register-Intensive, Arithmetic Functions e.g. Adders, Comparators, Counters, ...</td>
</tr>
</tbody>
</table>
MAX 7000 Families

◆ Today’s MAX 7000 family members
  • Basic version: for low-density members
    – EPM7032/V, 7064, 7096
  • E-version: enhanced architecture, for higher-density members
    – EPM7128E, 7160E, 7192E, 7256E
  • New S-version: enhanced architecture with ISP capability
    – EPM7032S, 7064S, 7096S, 7128S, 7160S, 7192S, 7256S
# MAX 7000 Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>MCs</th>
<th>Gates</th>
<th>Speed Grade</th>
<th>Package Options</th>
<th>I/O Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPM7032</td>
<td>32</td>
<td>600</td>
<td>-5,-6,-7,-10,-12,-15</td>
<td>PLCC44, TQFP44</td>
<td>36</td>
</tr>
<tr>
<td>EPM7032V</td>
<td>32</td>
<td>600</td>
<td>-12,-15,-20</td>
<td>PLCC44, TQFP44</td>
<td>36</td>
</tr>
<tr>
<td>EPM7064</td>
<td>64</td>
<td>1,250</td>
<td>-6,-7,-10,-12,-15</td>
<td>PLCC44/68/84, PQFP100, TQFP44</td>
<td>36,52,68</td>
</tr>
<tr>
<td>EPM7096</td>
<td>96</td>
<td>1,800</td>
<td>-7,-10,-12,-15</td>
<td>PLCC68/84, PQFP100</td>
<td>52,64,76</td>
</tr>
<tr>
<td>EPM7128E</td>
<td>128</td>
<td>2,500</td>
<td>-7,-10,-10P,-12,-15,-20</td>
<td>PLCC84, PQFP100/160</td>
<td>68,84,100</td>
</tr>
<tr>
<td>EPM7160E</td>
<td>160</td>
<td>3,200</td>
<td>-10,-10P,-12,-15,-20</td>
<td>PLCC84, PQFP100/160</td>
<td>64,84,100</td>
</tr>
<tr>
<td>EPM7192E</td>
<td>192</td>
<td>3,750</td>
<td>-12,-12P,-15,-20</td>
<td>PQFP160, PGA160</td>
<td>124</td>
</tr>
<tr>
<td>EPM7256E</td>
<td>256</td>
<td>5,000</td>
<td>-12,-12P,-15,-20</td>
<td>PQFP160, PGA192, RQFP208</td>
<td>132,164</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device</th>
<th>MCs</th>
<th>Gates</th>
<th>Speed Grade</th>
<th>Package Options</th>
<th>I/O Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPM7032S</td>
<td>32</td>
<td>600</td>
<td>-5,-6,-7,-10</td>
<td>PLCC44, TQFP44</td>
<td>36</td>
</tr>
<tr>
<td>EPM7064S</td>
<td>64</td>
<td>1,250</td>
<td>-6,-7,-10</td>
<td>PLCC44/84, PQFP100, TQFP44/100</td>
<td>36,52,68</td>
</tr>
<tr>
<td>EPM7096S</td>
<td>96</td>
<td>1,800</td>
<td>-6,-7,-10</td>
<td>PLCC84, PQFP100, TQFP100</td>
<td>52,64,76</td>
</tr>
<tr>
<td>EPM7128S</td>
<td>128</td>
<td>2,500</td>
<td>-7,-10,-15</td>
<td>PLCC84, PQFP100/160, TQFP100</td>
<td>68,84,100</td>
</tr>
<tr>
<td>EPM7160S</td>
<td>160</td>
<td>3,200</td>
<td>-7,-10,-15</td>
<td>PLCC84, PQFP100/160, TQFP100</td>
<td>64,84,104</td>
</tr>
<tr>
<td>EPM7192S</td>
<td>192</td>
<td>3,750</td>
<td>-7,-10,-15</td>
<td>PQFP160</td>
<td>124</td>
</tr>
<tr>
<td>EPM7256S</td>
<td>256</td>
<td>5,000</td>
<td>-7,-10,-12,-15</td>
<td>PQFP160, RQFP208</td>
<td>132,164</td>
</tr>
</tbody>
</table>
MAX 7000 Features

MAX 7000 main features...

- EEPROM-based devices based on Altera’s MAX architecture
- 32 ~ 256 macrocells
- 600 ~ 5,000 usable gates
- Programmable flip-flops with individual clear, preset & clock enable controls
- Configurable expander allowing up to 32 product terms per macrocell
- Programmable power-saving mode in each macrocell
- Programmable security bit
- PCI-compliant -5, -6, -7, -10P, -12P speed grades
- 3.3-V or 5-V operation
  - Full 3.3-V EPM7032V
  - 3.3-V or 5-V I/O on all devices except 44-pin devices
MAX 7000E/S Features

◆ **MAX 7000E** (128MCs and up) enhanced features...
  • More output enable control signals & more global clocking capabilities
  • Fast input registers
  • Programmable output slew-rate control
  • More interconnect resources

◆ **MAX 7000S enhanced features**
  • Enhanced architecture for all family members
  • Open-drain output option for each I/O pin
  • In-system programmability (ISP) via standard JTAG interface
  • Built-in JTAG boundary-scan test circuitry in EPM7128S or larger devices
  • **ClockBoost** circuitry: a phase-locked loop (PLL) circuit which provides a clock multiplier
  • PCI-compliant -5, -6, -7, -10 speed grades
  • Pin-, function- & programming file-compatible with all MAX 7000/E devices
MAX 7000 Architecture

[Diagram showing MAX 7000 Architecture with details on I/O control blocks, macrocells, and LABs.]
MAX 7000E/S Architecture
MAX 7000E/S Macrocell

- 36 Programmable Interconnect Signals
- 16 Expander Product Terms
- Parallel Logic Expanders (from other MCs)
- Product-Term Select Matrix
- Shared Logic Expanders
- Global Clear Clocks
- Fast Input Select
- Programmed Select Register
- Register Bypass
- Clock/Enable Select
- to I/O Control Block
- to PIA
Shareable Expanders

Macrocell Product-Term Logic

Product-Term Select Matrix

Macrocell Product-Term Logic
Parallel Expanders

- Macrocell
- Product-Term Logic
- Select Matrix

From Previous Macrocell:
- Cannot borrow any parallel expanders

To Next Macrocell:
- Cannot borrow any parallel expanders
- Borrow up to 10 parallel expanders from the first & second MCs
- Borrow up to 15 parallel expanders from the 3 MCs immediately above it

LAB:
- Cannot borrow any parallel expanders
- Borrow up to 5 parallel expanders from the first MC
- Borrow up to 5 parallel expanders from the first MC
- Borrow up to 10 parallel expanders from the first & second MCs
- Borrow up to 15 parallel expanders from the 3 MCs immediately above it
MAX 7000 I/O Control Block
MAX 7000E/S I/O Control Block

- 6 Global Output Enables
- VCC
- GND
- Fast Input to MC Register
- to PIA
- Open-Drain Output (MAX 7000S only)
- Slew-Rate Control
MAX Vertical Migration

<table>
<thead>
<tr>
<th>Device</th>
<th>44-Pin PLCC</th>
<th>44-Pin TQFP</th>
<th>48-Pin 0.8-mm BGA</th>
<th>84-Pin PLCC</th>
<th>100-Pin FineLine BGA™</th>
<th>100-Pin TQFP</th>
<th>144-Pin TQFP</th>
<th>168-Pin 0.8-mm BGA</th>
<th>208-Pin PQFP</th>
<th>256-Pin FineLine BGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPM7032</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPM7064</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPM7128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPM7256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPM7512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MAX 7000/E/S Device Programming

◆ Program the device with external hardware
  • Use Altera hardware programmer
    – MAX 7000/E/S devices can be programmed on PCs with an Altera Logic Programmer card, the Master Programming Unit (MPU), and the appropriate device adapter
    – You can test the programmed device in Altera’s software environment
  • Use the universal programmer
    – Many programming hardware manufacturers provide programming support for Altera MAX 7000/E/S devices

◆ MAX 7000S ISP
What’s ISP?

**ISP: In-System Programming**
- ISP allows devices to be mounted on a PCB before they are programmed
  - Offers quick and efficient design iterations
  - Eliminates package handling

**Mount Unprogrammed**
- Eliminates handling of devices
- Prevents bent leads

**Program In-System**
- Allows generic end-product inventory
- Specific test protocol or algorithm can be programmed during manufacturing or test flow

**Reprogramm in the Field**
- No need to return system for upgrades
- Add enhancements quickly & easily
MAX 7000S ISP

MAX 7000S ISP

- MAX 7000S devices can be programmed through 4-pin JTAG interface
  - By downloading the information via automatic test equipment, embedded processors, or Altera BitBlaster/ByteBlaster download cable
- MAX 7000S internally generates 12.0-V programming voltage
- Refer to Altera’s Application Brief & Application Note for details
  - AB145: Designing for In-System Programmability in MAX 7000S Devices
  - AN039: JTAG Boundary-Scan Testing in Altera Devices
Summary of MAX features

- Advanced I/O Standard Support
- Enhanced ISP Feature Set
- MultiVolt I/O
- PCI
- Jam STAPL
- FineLine BGA Packages
- 0.8mm BGA Packages
- ISP
- MAX 7000S
- MAX 7000A
- MAX 7000B
## FLEX 8000A Family

◆ Today’s FLEX 8000A family members

<table>
<thead>
<tr>
<th>Device</th>
<th>Gates</th>
<th>LEs</th>
<th>FFs</th>
<th>Speed Grade</th>
<th>Package Options</th>
<th>I/O Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPF8282A</td>
<td>2,500</td>
<td>208</td>
<td>282</td>
<td>-2,-3,-4</td>
<td>PLCC84, TQFP100</td>
<td>68,78</td>
</tr>
<tr>
<td>EPF8282AV</td>
<td>2,500</td>
<td>208</td>
<td>282</td>
<td>-4</td>
<td>TQFP100</td>
<td>68,78</td>
</tr>
<tr>
<td>EPF8452A</td>
<td>4,000</td>
<td>336</td>
<td>452</td>
<td>-2,-3,-4</td>
<td>PLCC84, TQFP100, PQFP160, PGA160</td>
<td>68,120</td>
</tr>
<tr>
<td>EPF8636A</td>
<td>6,000</td>
<td>504</td>
<td>636</td>
<td>-2,-3,-4</td>
<td>PLCC84, PQFP160/208, PGA192</td>
<td>68,118,136</td>
</tr>
<tr>
<td>EPF8820A</td>
<td>8,000</td>
<td>672</td>
<td>820</td>
<td>-2,-3,-4</td>
<td>TQFP144, PQFP160/208, PGA192, BGA225</td>
<td>120,152</td>
</tr>
<tr>
<td>EPF81188A</td>
<td>12,000</td>
<td>1,008</td>
<td>1,188</td>
<td>-2,-3,-4</td>
<td>PQFP208/240, PGA232</td>
<td>148,184</td>
</tr>
<tr>
<td>EPF81500A</td>
<td>16,000</td>
<td>1,296</td>
<td>1,500</td>
<td>-2,-3,-4</td>
<td>PQFP240, PGA280, RQFP304</td>
<td>181,208</td>
</tr>
</tbody>
</table>
FLEX 8000A Features

◆ FLEX 8000A main features...

• SRAM-based devices based on Altera’s FLEX architecture
• 282 ~ 1,500 registers
• 2,500 ~ 16,000 usable gates
• Programmable flip-flops with individual clear & preset controls
• Dedicated carry chain & cascade chain
• FastTrack continuous routing structure
• Programmable output slew-rate control
• Supports in-circuit reconfiguration (ICR)
• JTAG boundary-scan test circuitry
• PCI-compliant -2 speed grade
• 3.3-V or 5-V operation
  – Full 3.3-V EPF8282AV
  – 3.3-V or 5-V I/O for EPF8636A and larger devices
FLEX 8000A Logic Element

Diagram showing the connections and components of the FLEX 8000A Logic Element:
- **Carry-In** and **Cascade-In** inputs.
- **Look-Up Table (LUT)**.
- **Carry Chain**.
- **Cascade Chain**.
- **Clear/Preset Logic**.
- **Carry-Out** and **Cascade-Out** outputs.
- **Programmable Register**.
- **Clock Select**.
- **LE Out** output.
- **DATA1**, **DATA2**, **DATA3**, **DATA4** inputs.
- **LABCTRL1**, **LABCTRL2**, **LABCTRL3**, **LABCTRL4** inputs.
Carry Chains

Carry-In

\[ A_1, B_1 \]

\[ \text{LUT} \rightarrow \text{Register} \rightarrow S_1 \]

\[ \text{Carry Chain} \]

\[ \text{LE}_1 \]

\[ A_2, B_2 \]

\[ \text{LUT} \rightarrow \text{Register} \rightarrow S_2 \]

\[ \text{Carry Chain} \]

\[ \text{LE}_2 \]

\[ A_n, B_n \]

\[ \text{LUT} \rightarrow \text{Register} \rightarrow S_n \]

\[ \text{Carry Chain} \]

\[ \text{LE}_n \]

\[ \text{LE}_{n+1} \]

Carry-Out
Cascade Chains

AND Cascade Chain

OR Cascade Chain
FLEX 8000A Logic Array Block

LAB local Interconnect (32 channels)

LAB Control Signals

Carry-In & Cascade-In from LAB on left

Carry-Out & Cascade-Out to LAB on right

Row FastTrack Interconnect

Column-to-Row Interconnect

Column FastTrack Interconnect
FLEX 8000A FastTrack Interconnect

Row FastTrack (168/216 channels)

Column FastTrack (16 channels)

Local FastTrack (32 channels)
FLEX 8000A I/O Element

(OE[4..9]) are for EPF81500A devices only
FLEX 8000A Configuration

Configuration schemes & data source

- Refer to Altera’s Application Notes for details
  - AN033: Configuring FLEX 8000 Devices
  - AN038: Configuring Multiple FLEX 8000 Devices

<table>
<thead>
<tr>
<th>Configuration Scheme</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS (Active Serial)</td>
<td>Serial configuration EPROM</td>
</tr>
<tr>
<td>APU (Active Parallel Up)</td>
<td>Parallel EPROM</td>
</tr>
<tr>
<td>APD (Active Parallel Down)</td>
<td>Parallel EPROM</td>
</tr>
<tr>
<td>PS (Passive Serial)</td>
<td>Serial data path (e.g. serial download cable)</td>
</tr>
<tr>
<td>PPS (Passive Parallel Synchronous)</td>
<td>Intelligent host</td>
</tr>
<tr>
<td>PPA (Passive Parallel Asynchronous)</td>
<td>Intelligent host</td>
</tr>
</tbody>
</table>
MAX 9000A Devices
MAX 9000A KEY FEATURE

◆ MAX 9000A main features…
  • EEPROM-based devices based on Altera’s MAX architecture
  • 320 ~ 560 macrocells
  • 6,000 ~ 12,000 usable gates
  • Configurable expander allowing up to 32 product terms per macrocell
  • FastTrack continuous routing structure
  • I/O registers with clock enable & output slew-rate controls on all I/O pins
  • Programmable security bit
  • 5-V ISP through built-in JTAG interface
  • 3.3-V or 5-V I/O operation on all devices
# MAX 9000A Family

<table>
<thead>
<tr>
<th>Feature</th>
<th>EPM9320A</th>
<th>EPM9400</th>
<th>EPM9560A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macrocells</td>
<td>320</td>
<td>400</td>
<td>560</td>
</tr>
<tr>
<td>Max. # FF</td>
<td>484</td>
<td>580</td>
<td>772</td>
</tr>
<tr>
<td>Packages</td>
<td>84 PLCC 208 RQFP 280 PGA 356 BGA</td>
<td>84 PLCC 208 RQFP 240 RQFP</td>
<td>208 RQFP 240 RQFP 304 RQFP 280 PGA 356 BGA</td>
</tr>
</tbody>
</table>
MAX 9000 Architecture

Up to 10 IOCs per half column

Up to 8 IOCs per half row

Macrocell
This path from P-Term Select Matrix supports register packing.

LAB Local Array

Parallel Expanders from other macrocells

Global Clear Clocks

Shareable Logic Expander

16 local feedbacks

16 shared expander product terms

16 shared expander product terms

Local Array Feedback

To FastTrack Interconnect

33 Row FastTrack Interconnect Inputs

Product-Term Select Matrix

PRN

EN CLRN

Clock/Enable Select

Clear Select

QD EN CLRN

Select

33 Row FastTrack Interconnect Inputs
MAX 9000A Logic Array Block

Each row signal has two paths into the LAB.

To peripheral control bus and other LABs in the device.

Each macrocell can drive both the Row and Column Interconnect at the same time.
MAX 9000 FastTrack Interconnect

- Row FastTrack (96 channels)
- Column FastTrack (48 channels)
- LAB Local Array (114 Channels)
MAX 9000 I/O Cell

Peripheral Control Bus[12..0] : OE/ENA[4..0],OE5,OE6,OE7,CLR1,CLR0/ENA5,CLK[3..0]
MAX 9000 Device Programming

◆ Program the device with external hardware
  • Use Altera hardware programmer
    – MAX 9000 devices can be programmed on PCs with an Altera Logic Programmer card, the Master Programming Unit (MPU), and the appropriate device adapter
    – You can test the programmed device in Altera’s software environment
  • Use the universal programmer
    – Many programming hardware manufacturers provide programming support for Altera MAX 9000 devices

◆ MAX 9000 ISP

![Diagram of MAX 9000 ISP with CPLD and programmer & adapter](image)
MAX 9000 ISP

◆ MAX 9000 ISP
  • MAX 9000 devices can be programmed through 4-pin JTAG interface
    – By downloading the information via automatic test equipment, embedded processors, or Altera BitBlaster/ByteBlaster download cable
  • MAX 9000 internally generates 12.0-V programming voltage
  • Refer to Altera’s Application Brief & Application Note for details
    – AB141: In-System Programmability in MAX 9000 Devices
    – AN039: JTAG Boundary-Scan Testing in Altera Devices
MAX Supports Jam STAPL for ISP

- JEDEC-Approved Open Standard
- Small File Size
- Faster Programming Times
- Vendor-Independent
- Platform-Independent
- Supports Existing and Future Products

http://www.jamisp.com
Embedded Programming using Jam Player

Generating Programming Signals from the Jam File

Jam Player

I/O Functions

Main Program

Parser

Extract Data

Compare & Export

Messages & Export

TCK

TMS

TDI

TDO

Jam (Programming Data & Algorithm)
MAX Family Slew Rate Control

◆ For designs without the above elements (or during prototype stages), the board may not be able to support the fast switching outputs of Altera devices
  • Faster switching outputs cause higher transient currents in outputs as they discharge load capacitance
  • These higher currents can cause ground bounce (ringing)
The magnitude of this ringing is $V = L \frac{di}{dt}$

- where $L$ is the board inductance and $\frac{di}{dt}$ is the rate of current
- For more information about ground bounce, see AN 75: High-Speed Board Designs (Data Book)

Ground Bounce

$L \frac{di}{dt}$

$V_{th}$
Programmable Speed/ Power Control

◆ MAX devices offer low-power OR high speed operation through the Turbo Bit logic option
◆ Power dissipation can be reduced by 50% or more
◆ This is controllable for the entire device OR on a macrocell by macrocell basis

• The user can have a section of the design operating in high performance (Turbo Bit = on) and in the same device, other sections may be operating in low power (Turbo Bit = off)
• MAX 7000 devices: macrocells running at low power (Turbo Bit = off) incur a delay $t_{LPA}$ (8 ns for -5 speed grade) for the $t_{LAD}$, $t_{LAC}$, $t_{IC}$, $t_{ACL}$, $t_{EN}$, $t_{SEXP}$ parameters
• MAX 9000 devices: macrocells running at low power (Turbo Bit = off) incur a delay $t_{LPA}$ for the LAB local array delay ($t_{local}$)
FLEX 10K Devices
<table>
<thead>
<tr>
<th>Features</th>
<th>EPF10K10</th>
<th>EFP10K20</th>
<th>EFP10K30</th>
<th>EFP10K40</th>
<th>EFP10K50</th>
<th>EFP10K70</th>
<th>EFP10K100</th>
<th>EFP10K130V</th>
<th>EFP10K250A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Gates</td>
<td>10,000</td>
<td>20,000</td>
<td>30,000</td>
<td>40,000</td>
<td>50,000</td>
<td>70,000</td>
<td>100,000</td>
<td>130,000</td>
<td>250,000</td>
</tr>
<tr>
<td>Logic Elements</td>
<td>576</td>
<td>1,152</td>
<td>1,728</td>
<td>2,304</td>
<td>2,880</td>
<td>3,744</td>
<td>4,992</td>
<td>6,656</td>
<td>12,160</td>
</tr>
<tr>
<td>RAM Bits</td>
<td>6.144</td>
<td>12,288</td>
<td>12,288</td>
<td>16,384</td>
<td>20,480</td>
<td>18,432</td>
<td>24,576</td>
<td>32,768</td>
<td>40,960</td>
</tr>
<tr>
<td>Registers</td>
<td>720</td>
<td>1,344</td>
<td>1,968</td>
<td>2,576</td>
<td>3,184</td>
<td>4,096</td>
<td>5,392</td>
<td>7,120</td>
<td>12,624</td>
</tr>
<tr>
<td>Max. User I/O</td>
<td>134</td>
<td>189</td>
<td>246</td>
<td>189</td>
<td>310</td>
<td>358</td>
<td>406</td>
<td>470</td>
<td>470</td>
</tr>
</tbody>
</table>
FLEX 10K Features

◆ FLEX 10K/A main features...

- SRAM-based devices based on Altera’s FLEX architecture
- Embedded programmable logic family
  - Embedded array for implementing RAMs & specialized logic functions
  - Logic array for general logic functions
- High density
  - 10,000 ~ 100,000 typical gates (logic & RAMs)
  - 720 ~ 5,392 registers
  - 6,144 ~ 24,576 RAM bits
- Flexible interconnect
  - FastTrack continuous routing structure
  - Dedicated carry chain & cascade chain
  - Up to 6 global clock & 4 global clear signals
FLEX 10K Features - (2)

◆ FLEX 10K main features... (continued)
  • Powerful I/O pins
    – Individual tri-state control for each pin
    – Programmable output slew-rate control
    – Open-drain option on each I/O pin
    – Peripheral register
  • System-level features
    – Supports in-circuit reconfiguration (ICR)
    – JTAG boundary-scan test circuitry
    – PCI-compliant -3 speed grade
    – 3.3-V or 5-V I/O pins on devices in PGA, BGA & 208-pin QFP packages
    – ClockLock & ClockBoost option (for EPF10K100GC503-3DX device only)
  • Flexible package options
    – Pin-compatibility with other FLEX 10K devices in the same packages
Altera 10KE Device

**BREAKTHROUGH PERFORMANCE**
- DESIGNED FOR PCI
- 100-MHz SYSTEM SPEED
- 150-MHz FIFOs

**Next-Generation Packaging**
- 1.0-mm FineLine BGA™ Packages
- Requires Half the Board Area
- Minimizes Cost

**Advanced Process Technology**
- 0.25-µm CMOS SRAM
- Five-Layer Metal
- 2.5-V Core with MultiVolt™ I/O
- 5.0-V Tolerant Inputs

**Embedded Architecture Evolution**
- Dual-Port RAM
- 4-Kbit EAB with x16 Width
- PCI-Compliant I/O
Flex10KE Family Member

<table>
<thead>
<tr>
<th>Features</th>
<th>EPF10K30E</th>
<th>EPF10K50E</th>
<th>EPF10K100E</th>
<th>EPF10K130E</th>
<th>EPF10K200E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Gates</td>
<td>30,000</td>
<td>50,000</td>
<td>100,000</td>
<td>130,000</td>
<td>200,000</td>
</tr>
<tr>
<td>Logic Elements</td>
<td>1,728</td>
<td>2,880</td>
<td>4,992</td>
<td>6,656</td>
<td>9,984</td>
</tr>
<tr>
<td>RAM Bits</td>
<td>24,576</td>
<td>40,960</td>
<td>49,152</td>
<td>65,536</td>
<td>98,304</td>
</tr>
<tr>
<td>Registers</td>
<td>1,968</td>
<td>3,184</td>
<td>5,392</td>
<td>7,120</td>
<td>10,448</td>
</tr>
<tr>
<td>Max. User I/O</td>
<td>246</td>
<td>310</td>
<td>406</td>
<td>470</td>
<td>470</td>
</tr>
</tbody>
</table>
FLEX 10K Architecture

IOE

Logic Element

EAB

Embedded Array

LAB

Logic Array

Logic Array
What is the EAB?

- Larger block of RAM embedded into the PLD
- Can be preloaded with a pattern
- EAB size is flexible - 256x8 / 512x4 / 1024x2 / 2048x1
- You can combine EABs to create larger blocks
- Using RAM does not impact logic capacity

EAB as logic

- EAB is preloadable at configuration time
- You can use EAB to create a large lookup table or ROM
- EAB is the same die size of 16 LEs, however, one EAB can perform complex functions requiring more than 16 LEs
  - Example: 4x4 Multiplier (40 LEs, 43MHz) vs. (1 EAB, 73MHz)
FLEX 10K/V/A EAB

Data In 1, 2, 4, 8
Address 11, 10, 9, 8
Write Enable
In Clock
Out Clock

RAM/ROM
2,048 Bits
256 x 8
512 x 4
1,024 x 2
2,048 x 1

EAB contains registers for incoming and outgoing signals.
10KE EAB

RAM/ROM 4,096 Bits

256x16
512x8
1024x4
2048x2

Data In
Write Address
Write Enable
Read Address
Read Enable
Clock 1
Clock 1 Enable
Clock 2
Clock 2 Enable

Read Enable
Write Enable
Data Out
Write Address
Write Enable
Read Address
Read Enable
Clock 1
Clock 1 Enable
Clock 2
Clock 2 Enable

EAB contains registers for incoming and outgoing signals
FLEX 10K Register Packing

Carry-In Cascade-In

Carry-Out Cascade-Out

Data1 Data2 Data3 Data4

LABCTRL1 LABCTRL2 Device-Wide Clear

LABCTRL3 LABCTRL4

Clear/ Preset Logic

Clock Select

Carry-Out Cascade-Out

Programmable Register

to FastTrack Interconnect
to LAB Local Interconnect
FLEX 10K Logic Array Block

- **Dedicated Inputs & Global Signals**
- **LAB local Interconnect (30/34 channels)**
- **LAB Control Signals**
- **Carry-In & Cascade-In**
- **Carry-Out & Cascade-Out**
- **Column-to-Row Interconnect**
- **Column FastTrack Interconnect**
FLEX 10K FastTrack Interconnect

Row FastTrack
(144/216/312 channels)

Column FastTrack
(24 channels)

Local FastTrack(30/34 channels)

LAB
LE
FLEX 10K I/O Element

- from One Row or Column Channel
- to Row or Column Interconnect
- from Row or Column Interconnect
- from One Row or Column Channel
- 2 Dedicated Clock Inputs
- Peripheral Control Bus[11..0]
- OE[7..0]
- Device-Wide Output Disable
- Programmable Inversion
- Open-Drain Output Slew-Rate Control
- Clock Inputs
- OE[7..0]
- ENA[5..0]
- CLRn[1..0]
- Clock Inputs
- 12
- OE[7..0]
ClockLock Feature

◆ ClockLock: faster system performance
  • ClockLock feature incorporates a phase-locked loop (PLL) with a balanced clock tree to minimize on-device clock delay & skew

```
Clock at Pin    Clock at Register

ClockLock Clock

Clock at Pin
ClockLock Clock
Clock at Register

Effective clock delay is small.
```
ClockBoost Feature

◆ ClockBoost: increased system bandwidth & reduced area
  - ClockBoost feature provides clock multiplication, which increases clock frequencies by as much as 4 times the incoming clock rate
  - You can distribute a low-speed clock on the PCB with ClockBoost
  - ClockBoost allows designers to implement time-domain multiplexed applications. The same functionality is accomplished with fewer logic resources.

- Note:
  1. Up to now, only EPF10K100-3DX devices support ClockLock & ClockBoost features.
  2. All new FLEX 10KA devices will support ClockBoost option.
# FLEX 10K Configuration

## Configuration schemes & data source

- Refer to Altera’s *Application Notes* for details
  - AN059: *Configuring FLEX 10K Devices*
  - AN039: *JTAG Boundary-Scan Testing in Altera Devices*

<table>
<thead>
<tr>
<th>Configuration Scheme</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PS</strong> (Passive Serial)</td>
<td>Altera’s EPC1 configuration EPROM, BitBlaster or ByteBlaster download cable, serial data source</td>
</tr>
<tr>
<td><strong>PPS</strong> (Passive Parallel Synchronous)</td>
<td>Intelligent host, parallel data source</td>
</tr>
<tr>
<td><strong>PPA</strong> (Passive Parallel Asynchronous)</td>
<td>Intelligent host, parallel data source</td>
</tr>
<tr>
<td><strong>JTAG</strong></td>
<td>JTAG controller</td>
</tr>
</tbody>
</table>
Configuration Application Notes, Data Sheets

◆ Application Notes
  • AN 33: Configuring FLEX 8000 Devices
  • AN 38: Configuring Multiple FLEX 8000 Devices
  • AN 87: Configuring FLEX 6000 Devices

◆ Data Sheets
  • BitBlaster Serial Download Cable
  • ByteBlasterMV Parallel Port Download Cable
  • Configuration Devices for FLEX Devices
  • Altera Programming Hardware
Altera Architecture Evolution

- Classic MAX 5000 MAX 7000/E/S
- FLEX 10K/A FLEX 8000A
- FLEX 6000 MAX 9000/A

Global Interconnect

PIA: Programmable Interconnect Array

Enhanced PIA

FastTrack Interconnect
FLEX 6000 Device Family

◆ FLEX 6000 main features
  • OptiFLEX™ Architecture
  • Gate Count from 10,000 to 24,000 Gates
  • 5.0 V, 0.5 m, TLM, SRAM Process (FLEX6000A 3.3 V, 0.35 m)
  • 125-MHz Performance (16-Bit Counter)
  • PCI-Compliant
  • Pin Migration
  • One Output Enable per Pin
  • MultiVolt™ I/O
  • High-Pin-Count TQFP, PQFP & BGA Packages
Pricing vs. Gate Arrays

- Competitive with Gate Array Unit Cost
- Benefits of Programmable Logic
  - Faster to Market
  - Low Risk
  - No NRE
  - No Re-Spin Cost
  - Short Lead Times
  - Low Inventory Cost

FLEX 6000 Provides Low-Cost Flexibility

Source: Dataquest/Altera
Mid-1999 Price Projections
Appendix: FLEX 6000 Architecture

FastTrack™ Interconnect

Interleaved LABs

FastFLEX™ I/O

μPitch™ Technology

3.2 mil (81 µm)
Bond Pads
What is $\mu$Pitch Technology?

- OptiFLEX Leverages Most Advanced Bond Pad Pitch in the PLD Industry
  - FLEX 6000 81 mm
  - FLEX 6000A 75 mm
- Maximum Die Size Reduction for Lowest Possible Cost
### FLEX 6000 Family

<table>
<thead>
<tr>
<th>Feature</th>
<th>EPF6010A</th>
<th>EPF6016</th>
<th>EPF6016A</th>
<th>EPF6024A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Geometry</td>
<td>0.35 µ</td>
<td>0.5 µ</td>
<td>0.35 µ</td>
<td>0.35 µ</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>3.3 V</td>
<td>5.0 V</td>
<td>3.3 V</td>
<td>3.3 V</td>
</tr>
<tr>
<td>Gate Count</td>
<td>5,000 - 10,000</td>
<td>8,000 - 16,000</td>
<td>8,000 - 16,000</td>
<td>12,000 - 24,000</td>
</tr>
<tr>
<td>Logic Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User I/O Pins (Max.)</td>
<td>880</td>
<td>1,320</td>
<td>1,320</td>
<td>1,960</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>204</td>
<td>171</td>
<td>218</td>
</tr>
<tr>
<td>Package Options*</td>
<td>100-Pin TQFP</td>
<td>144-Pin TQFP</td>
<td>100-Pin BGA*</td>
<td>144-Pin TQFP</td>
</tr>
<tr>
<td></td>
<td>100-Pin TQFP</td>
<td>208-Pin PQFP</td>
<td>100-Pin TQFP</td>
<td>208-Pin PQFP</td>
</tr>
<tr>
<td></td>
<td>144-Pin TQFP</td>
<td>240-Pin PQFP</td>
<td>144-Pin TQFP</td>
<td>240-Pin PQFP</td>
</tr>
<tr>
<td>Availability</td>
<td>256-Pin BGA*</td>
<td>256-Pin BGA</td>
<td>208-Pin PQFP</td>
<td>256-Pin BGA*</td>
</tr>
<tr>
<td></td>
<td>256-Pin BGA</td>
<td>256-Pin BGA</td>
<td>256-Pin BGA</td>
<td>256-Pin BGA</td>
</tr>
</tbody>
</table>

*Package options may vary depending on availability and region.
FLEX 6000 Logic Element

Look-Up Table (LUT)

Carry Chain

Cascade Chain

DATA 1
DATA 2
DATA 3
DATA 4
LAB Clear/Preset 1
LAB Clear/Preset 2
Chip Wide Clear

Clear/Preset Logic

Clock Select

Clock 1
Clock 2

Carry-In
Cascade-In

Carry-Out
Cascade-Out

PRN
CLRN

D Q

LE Out
Low Power/
MultiVolt™ Design
Low-Power/MultiVolt Design

- Providing 2.5-V Power Supply for FLEX 10KE
- Interfacing with Multi-Voltage Systems
2.5-V Power Advantage

◆ 0.25-μm Process Reduces Power by 54%

◆ Example
  • 50-MHz Design Uses 821 mW in EPF10K30A Device
  • Uses 379 mW in EPF10K30E Device

◆ Benefits
  • Smaller Power Supply
  • Simpler Cooling System
  • Less Heat Buildup
Designing for 2.5-V Power Supply

- **2.5-V Devices Becoming Common**
  - Memory, Microprocessors

- **What if FLEX 10KE Device Is Only 2.5-V Device?**
  - Generate 2.5-V Supply from 3.3-V or 5.0-V Supply

Source: Altera
Voltage Regulator Options

- Use Voltage Regulator to Generate 2.5-V Supply
  - Linear
  - Switching
- Selecting a Voltage Regulator
- Example: EPF10K200E Design
Linear Regulators

◆ Advantages
  • Component Count
  • Cost
  • Board Area
  • No EMI Radiation
  • Tight Voltage Regulation

◆ Disadvantages
  • Efficiency
  • Power Dissipation
Linear Regulator

Vin 2.5 V
Switching Regulators

♦ Advantages
  - Efficiency
  - Power Dissipation
  - Wide Input-Voltage Range
  - High-Current Capability

♦ Disadvantages
  - EMI
  - More Complex
  - Component Count
  - Board Area
  - Cost
Calculated \( I_{OUT} = 6.3 \text{ A} \)

- **Linear Solution (LT1580CT)**
  - Efficiency = 75%

- **Switching Solution (LTC1649)**
  - Efficiency \( \geq 90 \% \)

### Design Requirement

<table>
<thead>
<tr>
<th>Design Requirement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCIO Supply Level</td>
<td>3.3 V</td>
</tr>
<tr>
<td>VCCINT Supply Level</td>
<td>2.5 V</td>
</tr>
<tr>
<td>( f_{MAX} )</td>
<td>100 MHz</td>
</tr>
<tr>
<td>Output Pins</td>
<td>350</td>
</tr>
<tr>
<td>Utilization</td>
<td>100%</td>
</tr>
<tr>
<td>Supply Voltages Available on the Board</td>
<td>3.3 V, 5.0 V</td>
</tr>
</tbody>
</table>
Interfacing 2.5-V PLD to System

- Most Systems Today Incorporate 5.0-V & 3.3-V Devices
- 2.5-V FLEX 10KE Device Must Interface to System
- 2.5-V I/O Standards Incompatible with LVTTL/LVCMOS

What’s The Solution?
FLEX 10KE & Multi-Voltage Boards

- FLEX 10KE Interfaces with Multiple Voltage Levels
  - MultiVolt™ I/O Feature
  - 2.5-V, 3.3-V, 5.0-V I/O
  - 3.3-V PCI
3.3-V I/O with 2.5-V Logic

- 3.3-V Outputs Can Drive 3.3-V or 5.0-V Devices
- Altera Min. \( V_{OH} = V_{CC} - 0.2 \) V
  Exceeds 5.0-V TTL or 3.3-V CMOS/TTL Specifications

2.5-V & 5.0-V Tolerant Input Buffers

- 3.3-V Outputs Can Drive 3.3-V or 5.0-V Devices
- Altera Min. \( V_{OH} = V_{CC} - 0.2 \) V
  Exceeds 5.0-V TTL or 3.3-V CMOS/TTL Specifications
2.5-V I/O with 2.5-V Logic

- **FLEX 10KE Device**
- **Logic**
- **2.5-V Input**
- **3.3-V Input**
- **5.0-V Input**
- **GND**
- **2.5-V Output Can Drive 2.5-V Devices**
- **Min. \( V_{OH} = 2.1 \) V

**2.5-V & 5.0-V Tolerant Input Buffers**
Simulating Timing of MultiVolt I/O

- Increasing $V_{CCIO}$ Reduces Output Delay
- MAX+PLUS® II Accurately Models Timing Effect

Turn on MultiVolt™ I/O Setting when $V_{CCIO}$ Is Not Equal to $V_{CCINT}$
FLEX 10KE MultiVolt I/O
Summary

- Separate VCC Pins for Logic & I/O Pins
  - Logic Driven by VCCINT
  - I/O Pins Driven by VCCIO

- Connect VCCINT to 2.5-V Supply
- Connect VCCIO to 2.5-V or 3.3-V Supply

<table>
<thead>
<tr>
<th>VCCINT</th>
<th>VCCIO</th>
<th>Drives</th>
<th>Driven by</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2.5 V</td>
<td>3.3 V</td>
</tr>
<tr>
<td>2.5 V</td>
<td>3.3 V</td>
<td>Ok</td>
<td>Ok</td>
</tr>
<tr>
<td>2.5 V</td>
<td>2.5 V</td>
<td>Ok</td>
<td></td>
</tr>
</tbody>
</table>
# Altera’s Multivolt Offering

<table>
<thead>
<tr>
<th>Device</th>
<th>Technology</th>
<th>VccINT</th>
<th>VccIO</th>
<th>Drives(TTL)</th>
<th>Driven by</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX6000</td>
<td>0.5</td>
<td>5</td>
<td>5</td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td>FLEX6000A</td>
<td>0.35</td>
<td>3.3</td>
<td>3.3</td>
<td>Y Y Y Y Y Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td>FLEX8000A</td>
<td>.6,.5</td>
<td>5</td>
<td>5</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>.6,.5</td>
<td>5</td>
<td>5</td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y Y Y Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td>EPF8282AV</td>
<td>.6,.5</td>
<td>3.3</td>
<td>3.3</td>
<td>Y Y Y Y</td>
<td>Y</td>
</tr>
<tr>
<td>FLEX10K</td>
<td>0.5</td>
<td>5</td>
<td></td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y Y Y Y Y Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td>FLEX10A</td>
<td>0.35</td>
<td>3.3</td>
<td>3.3</td>
<td>Y Y Y Y Y Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td>EPF10KE</td>
<td>0.22</td>
<td>2.5</td>
<td>3.3</td>
<td>Y Y Y Y Y Y</td>
<td>Y Y Y</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y Y Y</td>
</tr>
</tbody>
</table>
# Device feature summary

<table>
<thead>
<tr>
<th></th>
<th>3.3V</th>
<th>ISP</th>
<th>ICR</th>
<th>EAB</th>
<th>Open Drain</th>
<th>GCLK</th>
<th>Dedicated Input</th>
<th>OE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX10K(A)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>2(+4)</td>
<td>4</td>
<td>ALL</td>
</tr>
<tr>
<td>FLEX8K</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>(+4)</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>FLEX6K(A)</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>(+4)</td>
<td>4</td>
<td>ALL</td>
</tr>
<tr>
<td>MAX9000</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MAX7000S</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Notes:
- Multi-Volt; PCI; Slew slow rate; JTAG-BST ==> All Devices
- PLL = Phase Locked Loop (ClockBoost)
- ISP = In System Programmability
- ICR = In Circuit Reconfiguration
- OE = Output Enable
MAX+PLUS II Can ...(An Introduction)

◆ Operate in a self-contained environment

**Design Entry**
- Standard EDA Design Entry
  - Cadence
  - Mentor Graphics
  - OrCAD
  - Synopsys
  - Viewlogic
  - Synplicity
  - Exemplar
  - Others

- Graphic Design Entry

- Text Design Entry (AHDL, VHDL, Verilog HDL)

- High-Level Design Entry
  - MegaCore Functions
  - LPM Functions
  - AMPP Megafuntions

- Waveform Design Entry

- Hierarchical Design Entry

- Floorplan Editing

**Design Compilation**
- MAX+PLUS II Compiler

- Design-Rule Checking

- Logic Synthesis & Fitting

- Multi-Device Partitioning

- Automatic Error Location

- Timing-DrivenCompilation

- OpenCore Evaluation

**Verification & Programming**
- Standard EDA Verification
  - Cadence
  - Mentor Graphics
  - OrCAD
  - Synopsys
  - Viewlogic
  - Others

- Timing Simulation

- Functional Simulation

- Multi-Device Simulation

- Timing Analysis

- Device Programming

- OpenCore Evaluation
MAX+PLUS II Software Products

◆ Fixed-Node Subscription Products
  • Windows 95/98 and Windows NT Operating System, require hardware protection key for node identification
  • FIXEDPC full featured MAX+PLUS II software with VHDL/Verilog

◆ Floating-Node Subscription Products
  • Licensed Using Windows NT and UNIX Servers
  • FLOATPC for Windows 95/98 and Windows NT clients only.
  • FLOATNET for Windows 95/98/NT and UNIX clients.

◆ MAX+PLUS II BASELINE Software
  • entry-level version of the MAX+PLUS II software which is free of charge.
MAX+plus II

**Supported Platforms**
- PC
- UNIX Platform
  - Sun SPARCstation
  - HP 9000 Series 700/800 workstation
  - IBM RISC System /6000 workstation

*Please read the READ.ME file with every release of MAX+plus II*

**Network licensing supported on both PC and Unix**
Design Flow & Altera Tools

◆ FPGA/CPLD Design Flow
  • Design Ideas
  • Detailed Design
  • Functional Simulation
  • Synthesis & Implementation
  • Timing Simulation
  • Device Programming

◆ Altera MAX+PLUS II Development Software
  • Design Entry
  • Project Processing
  • Project Verification
  • Device Programming
FPGA/CPLD Design Flow

1. **Design Ideas**
2. **Detailed Design**
3. **Functional Simulation**
4. **Implementation (P&R)**
5. **Timing Simulation**
6. **Device Programming**

- **FPGA/CPLD**
  - $t_{pd}=22\text{.1}\text{ ns}$
  - $f_{max}=47.1\text{ MHz}$
Design Ideas

◆ What are the main design considerations?
  • Design feasibility?
  • Design spec?
  • Cost?
  • FPGA/CPLD or ASIC?
  • Which FPGA/CPLD vendor?
  • Which device family?
  • Development time?
Detailed Design

◆ Choose the design entry method
  • Schematic
    – Gate level design
    – Intuitive & easy to debug
  • HDL (Hardware Description Language), e.g. Verilog & VHDL
    – Descriptive & portable
    – Easy to modify
  • Mixed HDL & schematic

◆ Manage the design hierarchy
  • Design partitioning
    – Chip partitioning
    – Logic partitioning
  • Use vendor-supplied libraries or parameterized libraries to reduce design time
  • Create & manage user-created libraries (circuits)
Functional Simulation

- **Preparation for simulation**
  - Generate simulation patterns
    - Waveform entry
    - HDL testbench
  - Generate simulation netlist

- **Functional simulation**
  - To verify the functionality of your design only

- **Simulation results**
  - Waveform display
  - Text output

- **Challenge**
  - Sufficient & efficient test patterns
Design Implementation

◆ Implementation flow
  • Netlist merging, flattening, data base building
  • Design rule checking
  • Logic optimization
  • Block mapping & placement
  • Net routing
  • Configuration bitstream generation

◆ Implementation results
  • Design error or warnings
  • Device utilization
  • Timing reports

◆ Challenge
  • How to reach high performance & high utilization implementation?
Timing Analysis & Simulation

◆ Timing analysis
  • Timing analysis is static, i.e., independent of input & output patterns
  • To examine the timing constraints
  • To show the detailed timing paths
  • Can find the critical path

◆ Timing simulation
  • To verify both the functionality & timing of the design
Device Programming

◆ Choose the appropriate configuration scheme
  • SRAM-based FPGA/CPLD devices
    – Downloading the bitstream via a download cable
    – Programming onto a non-volatile memory device & attaching it on the circuit board
  • OTP, EPROM, EEPROM or Flash-based FPGA/CPLD devices
    – Using hardware programmer
    – ISP

◆ Finish the board design

◆ Program the device

◆ Challenge
  • Board design
  • System considerations
Altera Design Flow

 Operate seamlessly with other EDA tools
MAX+PLUS II
Altera’s Fully-Integrated Development System

Design Entry
- MAX+PLUS II Text Editor
- MAX+PLUS II Waveform Editor
- MAX+PLUS II Floorplan Editor
- MAX+PLUS II Symbol Editor

Project Verification
- MAX+PLUS II Simulator
- MAX+PLUS II Timing Analyzer

Device Programming
- MAX+PLUS II Programmer

Project Processing
- MAX+PLUS II Compiler
- CNF Extractor
- Database Builder
- Logic Synthesizer
- SNF Extractor
- Partitioner
- Fitter
- Netlist Writer
- Design Doctor
- Assembler

Message Processor & Hierarchy Display
Design Entry

◆ MAX+PLUS II design entry tools
  • Graphic Editor & Symbol Editor
    – For schematic designs
  • Text Editor
    – For AHDL and VHDL designs
    – However, VHDL is not covered by this course
  • Waveform Editor
  • Floorplan Editor
  • Hierarchy Display
MAX+PLUS II Design Entry

### MAX+PLUS II Design Entry

**Graphic Editor**

- **-**
  - **Name:**
  - **Value:**
  - **Time:** 185.0 ns

**Waveform Editor**

- **Name:**
- **Value:**
- **Time:** 600.0 ns

**Symbol Editor**

- **chiptrip**

```plaintext
SUBDESIGN time_cnt
{
  enable, clk : INPUT;
  time[7..0]  : OUTPUT;
}

VARIABLE
  count[7..0] : DFF;
BEGIN
  count[7].clk = clk;
  time[] = count[];
```

**Symbol Editor**

- **chiptrip**
- **-**
- **-**

**Text Editor**

- **chiptrip**
- **-**
- **-**

**Symbol Editor**

- **chiptrip**
- **-**
- **-**
Project Processing

MAX+PLUS II tools for project processing (implementation)

- MAX+PLUS II Compiler
- MAX+PLUS II Floorplan Editor
  - For pin, logic cell location assignments
- Message Processor
  - For error detection & location
MAX+PLUS II Project Processing
Project Verification

◆ MAX+PLUS II tools for project verification
  • MAX+PLUS II Simulator
  • MAX+PLUS II Waveform Editor
  • MAX+PLUS II Timing Analyzer
MAX+PLUS II Project Verification
Device Programming

- MAX+PLUS tool for device programming
  - MAX+PLUS II Programmer
MAX+PLUS II Features

◆ MAX+PLUS II, Altera’s fully integrated design environment
  • Schematic, text (AHDL), waveform design entry & hierarchy display
  • Floorplan editing
  • DRC, logic synthesis & fitting, timing-driven compilation
  • Multi-device partitioning
  • Automatic error location
  • Functional simulation, timing simulation, and multi-device simulation
  • Timing analysis
  • Programming file generation & device programming
  • EDA interface: industry-standard library support, EDA design entry & output formats (EDIF, Verilog & VHDL)
  • On-line help
Getting Started

- System Requirements
- Installing MAX+PLUS II
- Starting MAX+PLUS II
- Entering Authorization Codes
- MAX+PLUS II Manager Window
- MAX+PLUS II Project
- Hierarchy Display
System Requirements

◆ The minimum system requirements
  • Pentium- or 486-based PC
  • Microsoft Windows NT 3.51 or 4.0, Windows 95, or Windows version 3.1x with Win32s support
  • Microsoft Windows-compatible graphics card & monitor
  • Microsoft Window-compatible 2- or 3-button mouse
  • CD-ROM drive
  • Parallel port

◆ Memory & disk space requirement
  • Go to the read.me file for specific information about disk space & memory requirements in the current version of MAX+PLUS II
    – At least 64MB physical RAM is recommended
    – Memory requirement depends on the selected device and the design complexity
Installing MAX+PLUS II

◆ To install MAX+PLUS II from CD-ROM
  • Insert MAX+PLUS II CD-ROM into the CD-ROM drive. The installation program is located at:
    `<CD-ROM drive>:\pc\maxplus2\install.exe`
  • Follow the directions provided on-screen
  • Window 3.1x users:
    – Installation program will install Win32s files if they are not already present

◆ Additional Windows NT installation steps
  • You must install *Sentinel driver* after running the install program
    – To detect the key-pro
  • (Optional) ByteBlaster and Altera LP6 Logic Programmer Card drivers
    – Required only for ByteBlaster or LP6 users
Starting MAX+PLUS II

◆ To start MAX+PLUS II...
  • Double click on the MAX+PLUS II icon
Entering the Authorization Code

- When starting MAX+PLUS II for the first time
- Options -> license setup
  - You must enter an authorization code obtained from CIC
  - You can use all most MAX+PLUS II features after enter the correct auth-code
MAX+PLUS II Operating Environment

◆ MAX+PLUS II Manager
  • Start-up window

Project Directory and Project name

Toolbar provides shortcuts for commonly used functions

MAX+PLUS II menu gives you access to all MAX+PLUS II functions

Help menu gives you access to on-line help

Status bar provides a brief description of selected menu command and toolbar button

Opens an untitled Message Processor window or brings an open window to the foreground
MAX+PLUS II Menu

To invoke MAX+PLUS II applications
File Menu

- New...
- Open...
- Delete File...
- Hierarchy Project Top
- Exit MAX+plus II

Options Menu:
- Close
- Close All
- Save
- Save As...
Assign Menu

To specify project assignments & options
Options Menu

To setup user preferences
Help Menu
MAX+PLUS II Help Contents

◆ On-line help
  • All of the information necessary to enter, compile, and verify a design and to program an Altera device is available in MAX+PLUS II Help
  • Help also provides introductions to all MAX+PLUS II applications, design guidelines, pin and logic cell numbers for each Altera device package
MAX+PLUS II
Design Methodology
Design Entry Process

◆ Project Setup/Management
◆ Multiple design entry methods
  • MAX+PLUS II
    – Graphic design entry
    – Text design entry
      • AHDL, VHDL, Verilog
    – Waveform design entry
  • 3rd party EDA tools
    – EDIF, OrCAD schematics
  • Add flexibility and optimization to the Design entry process by:
    – mixing and matching design files
    – using LPM and Megafonctions to accelerate design entry
Project Setup/Management

◆ What is a Project?
  • A design file
  • A project is:
    – checked for design entry errors
    – compiled
    – simulated (functional or with timing)
    – analyzed for timing
    – used to generate programming file

◆ Projects can be archived

◆ To specify a project
  Menu: File -> Project -> Name... (To specify an existing or new design file)
  Menu: File -> Project -> Set Project to Current File (To specify the current design file)
Set Up A New Project

- Every design must have a project name
- Project name must match design file name
Design Entry Files

Top-level design files can be .gdf, .tdf, .vhd, .v, .sch, or .edf

Generated within MAX+PLUS II

Imported from other EDA tools

MAX+PLUS II Symbol Editor
MAX+PLUS II Graphic Editor
MAX+PLUS II Floorplan Editor
MAX+PLUS II Text Editor

Top-Level File

Verilog
VHDL
AHDL
Waveform
Schematic

OrCAD

Synopsys, Synplicity, Mentor Graphics, etc...
Hierarch Display

◆ MAX+PLUS II Hierarchy Display

- The Hierarchy Display shows a hierarchy tree that represents the current hierarchy and allows you to open and close files in the hierarchy.
- The hierarchy tree branches show a filename and file icon for each subdesign in the hierarchy, and it also shows ancillary files associated with the current hierarchy.
- To get a better perspective on your project, you can zoom in and out to different display scales or switch between vertical or horizontal orientation.
- To invoke Hierarchy Display

Menu: MAX+PLUS II -> Hierarchy Display
Hierarchy Display Window

File Icons
Graphic Design Entry

◆ MAX+PLUS II Graphic Editor & Symbol Editor
◆ Basic Knowledge
  • Naming Rules
  • User Libraries & System Libraries
◆ Creating Graphic Design Files
◆ Examples
Graphic Design Entry Process

◆ Add resource libraries to search list as needed

◆ Draw schematic
  • Enter design components (symbols)
  • Connect components with net (wires)
  • Add labels to key nets signal
    – Must label all busses, primary inputs, outputs, bidir

Note: MAX+PLUS II DOES NOT AUTO SAVE

◆ Save and check the design
  • The file extension is .gdf
  • Correct any errors with the aid of Message Processor

◆ Create symbol or include file for sub-design
Resource Libraries

- **prim (Altera primitives)**
  - Basic logic building blocks

- **mf (Macrofunction)**
  - 7400 family logic

- **mega_lpm (LPMs)**
  - Library of Parameterized Modules (LPMs)
  - Megafunctions are high level function module
    - busmux, ram elements, fifo’s, etc...
Value added Libraries

◆ MegaCores IP models you can try before purchase (download from www.altera.com)
  – UARTs, FFT, PCI etc…

◆ AMPP ( Altera Megafucntion Partners Program )
  • Partners providing PCI, DSP, μControllers, etc…

Note: For the latest information on MegaCores or Megafuctions, refer to Altera’s web site www.altera.com
Add User Resource Libraries

Access user created libraries
- Add user library directories
- Set priorities

Select the library directory then click on Add.

Library search priority can be changed.

The Project directory has the highest priority, followed by the User Libraries, then by the Altera Libraries.
Open New File & Enter Symbols

- Open a new .gdf file in Graphic Editor
- Double click in Graphic file to enter symbol
Graphic Editor Window

Selection tool
Text tool
Orthogonal line tool
Diagonal line tool
Arc tool
Circle tool

Zoom functions
Rubberbanding functions

Node
Symbol
Bus

INPUT symbols
OUTPUT symbol
Making Connections

◆ Wire
  • Single bit line

◆ Bus
  • Multi-bit line

◆ Signal name
  • Matching name
  • Attached to wire

Drawing tool shortcuts
Wire to Bus Connection

Bus - Bus signal names required for LPM module buses

Wire
Graphic Editor Options

◆ Font, Text Size
  • Text Control

◆ Line Style
  • Select Wire or Bus

◆ Display Assignments
  • Turns display on or off

◆ Guideline Control
  • Controls grid lines

◆ Rubber-banding
  • Wires move with symbols
Generate Symbols and Include Files

- Create symbol for higher-level schematic capture
- Create include file for AHDL or Verilog function prototype
Symbol Editor

◆ Symbols can be modified with the Symbol Editor
Pin/Node Naming

◆ Pin/node name
  • A pin name is enclosed within a pin primitive symbol; a node name is a text block that is associated with a node line (wire).

◆ Pin/node naming rules
  • It can contain up to 32 name characters
  • It may not contain blank spaces. Leading or trailing spaces are ignored.
  • It must be unique, i.e., no two pins may have the same name in the same design file at the same hierarchy level.
  • Any node that is connected to a bus line must be named
  • Node names that are bits of a dual-range bus must be expressed in the format <name>[<width>][<size>] or <name><width>_<size>. If you name a single node in this format, it will be interpreted as part of a dual-range bus if another single-range or dual-range bus in the file uses the same <name>.
Bus Naming

◆ **Single-range bus name**
  
  - Example: D[3..0] = D3,D2,D1,D0
  
  - The bus identifier can contain up to 32 name characters; the bus width can contain a maximum of 256 bits. The bus width is a string that defines the number of bits (i.e., nodes) in a bus and uses the form \(<\text{MSB}>..<\text{LSB}>\). The name of a single node within the bus can be specified with the identifier followed by the bit number, either with or without brackets.

◆ **Dual-range bus name**
  
  - Example: D[3..0][1..0] = D3_1,D3_0,D2_1,D2_0,D1_1,D1_0
  
  - A dual-range bus name uses two bracket-enclosed ranges [ ]: the bus width and the bus size. Bus widths and sizes can together define a maximum of 256 bits.

◆ **Sequential bus name**
  
  - Example: A[31..0],B,C[3..0]
  
  - A sequential bus name consists of a series of node names and/or bus names, separated by commas (,). The first node or bus bit in the series is the MSB, the last node in the series is the LSB.
Using Buffer Primitives - (1)

◆ Buffer primitives
  • Including: CARRY, CASCADE, EXP, GLOBAL, LCELL, OPNDRN, SOFT, TRI
  • All buffer primitives except TRI and OPNDRN allow you to control the logic synthesis process. In most circumstances, you do not need to use these buffers.

◆ GLOBAL primitive
  • To indicate that a signal must use a global clock, clear, preset or output enable signal, instead of signals generated with internal logic or driven by ordinary I/O pins
  • A NOT gate may be inserted between the input pin and GLOBAL

◆ TRI primitive
  • A active-high tri-state buffer

◆ OPNDRN primitive
  • An open-drain buffer, equivalent to a TRI primitive whose output enable input is fed by an signal, but whose primary input is fed by a GND primitive
  • Only supported for the FLEX 10K and MAX 7000S device families
Using Buffer Primitives - (2)

◆ **LCELL primitive**
  - The **LCELL** buffer allocates a logic cell for the project. An LCELL buffer always consumes one logic cell. It’s not removed from a project during logic synthesis.
  - Although **LCELL** primitives can be used to create an intentional delay or asynchronous pulse
    - However, race conditions can occur and create an unreliable circuit because the delay of these elements varies with temperature, power supply voltage and device fabrication process

◆ **SOFT primitive**
  - The **SOFT** buffer specifies that a logic cell may be needed in the project.
  - During project processing, MAX+PLUS II Compiler examines the logic feeding the primitive and determines whether a logic cell is needed. If it’s needed, the **SOFT** buffer is converted into an **LCELL**; if not, the **SOFT** buffer is removed.
More on LPM Libraries

◆ **Library of Parameterized Modules**
  - Standard Library of basic and functional elements
  - Based on EDIF standard

◆ **Advantage of LPMs**
  - Portability of design
  - Architecture independence

◆ **MAX+PLUS II and LPMs**
  - LPM can be used in graphical design and HDL designs
  - LPM can be customized via the Megawizard feature
Standard LPM without Megawizard
Using MegaWizard Plug-In Manager

◆ Click on the MegaWizard Plug-In Manager Button

Double click in Graphic Editor

Click on the MegaWizard Plug-In Manager
Accessing the MegaWizard

Select MegaWizard Plug-In Manager

The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafonctions.

Which action do you want to perform?
- [ ] Create a new custom megafonction variation
- [x] Edit an existing custom megafonction variation

Copyright (c) Altera Corporation 1988-1998. All rights reserved.
New vs Existing Megafunction

*Choose between a new custom megafunction variation or an existing megafunction variation*
Available Megafunctions & Output File

Select a function from the available megafunction

Select a type of output file

Select a directory and a output file name
Customizing the Megafunction

MegaWizard Plug-In Manager - LPM_MUX [page 3 of 4]

- How many 'data' inputs do you want? 4
- How wide should the 'data' input and the 'result' output buses be? 8 bits
- Do you want to pipeline the multiplexer? No
- Create an asynchronous Clear input

Options: Cancel, < Back, Next >, Finish
Files generated by the MegaWizard

Design file implemented in the language you selected (.tdf, .vhd, or .v)
INC an AHDL include file
CMP a VHDL component declaration file
SYM a Graphic design symbol file
Entering Customized Megafuntion

Double click in Graphic Editor

Customized megafunction appears the same way as other symbols in the Enter symbol window
Make Changes to Customized Megafunction

Double Click symbol will bring you back to the MegaWizard Plug-in Manager.

After the changes, MegaWizard will over-write the source file (tdf, vhd, v), inc file and cmp file for you.

Remember to update the symbol in your graphic editor.
How to Use System Functions?

To get help...

- You can find the detailed description for each primitive, macrofunction, and megafunction in MAX+PLUS II on-line help.

<table>
<thead>
<tr>
<th>Gates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1pm_and</td>
<td>1pm_inv</td>
</tr>
<tr>
<td>1pm_bustr</td>
<td>1pm_nux</td>
</tr>
<tr>
<td>1pm_cshift</td>
<td>1pm_or</td>
</tr>
<tr>
<td>1pm_constant</td>
<td>1pm_nor</td>
</tr>
<tr>
<td>1pm_decode</td>
<td>mux</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1pm_abs</td>
<td>1pm_counter</td>
</tr>
<tr>
<td>1pm_add_sub</td>
<td>1pm_mult</td>
</tr>
<tr>
<td>1pm_compare</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>configo</td>
<td>1pm_rom_dg</td>
</tr>
<tr>
<td>configram</td>
<td>1pm_rom_io</td>
</tr>
<tr>
<td>1pm_ff</td>
<td>1pm_rom</td>
</tr>
<tr>
<td>1pm_latch</td>
<td>1pm_dfl</td>
</tr>
<tr>
<td>1pm_shiftreg</td>
<td>1pm_dfl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>clksource</td>
<td>pull</td>
</tr>
<tr>
<td>ntkc</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MegaCore Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a16450</td>
<td>a8255</td>
</tr>
<tr>
<td>a6450</td>
<td>fft</td>
</tr>
<tr>
<td>a850</td>
<td>rgb2ycrc8</td>
</tr>
<tr>
<td>a837</td>
<td>ycrcb2rgb</td>
</tr>
<tr>
<td>a8251</td>
<td></td>
</tr>
</tbody>
</table>
Entering Symbols

◆ Enter a symbol
Menu: Symbol -> Enter Symbol...
(or by double clicking on the empty workspace)

◆ Move/cut/copy/paste symbols
- You can move, cut, copy or paste symbols in the same way as you did in another Windows-based software
  - Move: click & drag (mouse)
  - Cut: Ctrl-X
  - Copy: Ctrl-C or Ctrl-Click & drag
  - Paste: Ctrl-V
  - Undo: Ctrl-Z

◆ Commands regarding the symbol
- Just click the right mouse button on the symbol
Entering I/O Symbol

◆ I/O symbols
  • Input pin/port: enter a **INPUT** symbol
  • Output pin/port: enter a **OUTPUT** symbol
  • Bidirectional pin/port: enter a **BIDIR** symbol

◆ Name the I/O pins/ports
  • Double click on the “**PIN_NAME**” field of the I/O symbol

◆ Pin default value
  • The values assigned to unconnected INPUT and BIDIR primitives when the symbol that represents the current GDF file is used in a higher-level design file
  • Default is VCC
  • Double click on the “**VCC**” field to set the default value
Save & Check the Design

- Save & check the design file with .gdf extension
- Correct any errors with the aid of Message Processor
Message Processor

- Lists all Info, Warning and Error messages
  - Info messages are general information
  - Warning messages are possible problems
  - Error messages indicate Compiler is unable to complete compilation process

- Provides help on the messages
- Locates source of message in design file
Example: Multiplier

◆ **Design a multiplier with LPM_MULT**
  - The easiest way to create a multiplier is to use the LPM_MULT function
    - Can be unsigned or signed
    - Can be pipelined
    - Also can create a MAC(Multiplier-Accumulator) circuit
Example: Multiplexer

- **Design a multiplexer with** `LPM_MUX`
  - Use `WIRE` primitive to rename a bus or node
  - `LPM_MUX` data input is a dual range bus
Example: RAM

◆ Design RAM circuit with LPM
  • Use `LPM_RAM_IO` to design RAM with a single input & output port
  • Use `LPM_RAM_DQ` to design RAM with separate input & output ports
Example: Sequencer

Design a sequencer with \texttt{LPM\_COUNTER} \& \texttt{LPM\_ROM}

- ROM data is specified in a Memory Initialization File (.mif) or a Intel-Hex File (.hex)
- This example only sequences through 19 states so the modulus of lpm\_counter is set to 19. It uses a small section of an EAB (19 out of 256-address locations)
Example: Bidirectional Pin

- **Use TRI & BIDIR pin symbol**
  - If the TRI symbol feeds to an output or bidirectional pin, it will be implemented as tri-state buffer in the I/O cell
Example: Tri-State Buses

◆ Tri-state emulation
  • Altera devices do not have internal tri-state buses
  • MAX+PLUS II can *emulate* tri-state buses by using multiplexers and by routing the bidirectional line outside of the device and then back in through another pin

*MAX+PLUS II will automatically convert it into a multiplexer. If the tri-state buffers feed a pin, a tri-state buffer will be available after the multiplexer.*
Example: Tri-State Buses  - (2)

◆ Tri-state buses for bidirectional communication
  
  • When tri-state buses are used to multiplex signals, MAX+PLUS II will convert the logic to a combinatorial multiplexer
  
  • When tri-state buses are used for bidirectional communication, you can rout this bidirectional line outside of the device, which uses the tri-states present at the I/O pins, or you can convert the tri-state bus into a multiplexer
Example: Tri-State Buses

- Rout this bidirectional line outside of the device
- Tri-state emulation
Text Design Entry

◆ Set up a new project
  • Same as Graphic Design Entry

◆ Enter text description
  • AHDL
  • VHDL
  • Verilog

◆ Save & check the design
  • Similar to Graphic Design Entry
  • The file extension is .tdf, .vhd, .v
AHDL

- Altera Hardware Description Language
- High-level hardware behavior description language
- Uses Boolean equations, arithmetic operators, truth tables, conditional statements, etc.
- Especially well-suited for large or complex state machines
- Text Editor has AHDL Template and Syntax Color
- Refer to the Appendix for more info on AHDL
VHDL

◆ VHSIC Hardware Description Language
◆ 1987 and 1993 IEEE 1074 standard
◆ High-level hardware behavior description language
◆ Especially well-suited for large or complex designs
◆ Text Editor has VHDL Template and Syntax Color
Verilog

- Hardware Description Language
- 1993 Verilog IEEE 1364 standard
- High-level hardware behavior description language
- Especially well-suited for large or complex designs
- Text Editor has Verilog Template and Syntax Color
AHDL Design Entry

- What’s AHDL
- AHDL Structure
- AHDL Syntax
- MAX+PLUS II Text Editor
- Creating AHDL design files
- Examples
What’s AHDL?

◆ AHDL: Altera Hardware Description Language
  • To create MAX+PLUS II text design file (*.tdf)
  • High-level, modular hardware description language
  • Completely integrated into the MAX+PLUS II system
  • Especially well suited for...
    – Complex combinational logic
    – Group operations
    – State machines
    – Truth tables
    – Parameterized logic
  • Easy to learn & debug under MAX+PLUS II system
AHDL Example - (1)

SUBDESIGN 7segment
(i[3..0] : INPUT;
a, b, c, d, e, f, g : OUTPUT;
)
% -a- 0 1 2 3 %
% f| b 4 5 6 7 %
% -g- 8 9 A b %
% e| c C d E F %
% -d- %
BEGIN
TABLE
  i[3..0] => a, b, c, d, e, f, g;
  H"0" => 1, 1, 1, 1, 1, 1, 0;
  H"1" => 0, 1, 1, 0, 0, 0, 0;
  H"2" => 1, 1, 0, 1, 1, 0, 1;
  H"3" => 1, 1, 1, 1, 0, 0, 1;
  H"4" => 0, 1, 1, 0, 0, 1, 1;
  H"5" => 1, 0, 1, 1, 0, 1, 1;
  H"6" => 1, 0, 1, 1, 1, 1, 1;
  H"7" => 1, 1, 1, 1, 1, 1, 1;
  H"8" => 1, 1, 1, 1, 0, 1, 1;
  H"9" => 1, 1, 1, 0, 1, 1, 1;
  H"A" => 1, 1, 1, 0, 1, 1, 1;
  H"B" => 0, 0, 1, 1, 1, 1, 1;
  H"C" => 1, 0, 0, 1, 1, 1, 0;
  H"D" => 0, 1, 1, 1, 1, 0, 1;
  H"E" => 1, 0, 0, 1, 1, 1, 1;
  H"F" => 1, 0, 0, 0, 1, 1, 1;
END TABLE;
END;
AHDL Example - (2)

SUBDESIGN stepper
( clk, reset : INPUT;
  ccw, cw   : INPUT;
  phase[3..0] : OUTPUT;)

VARIABLE
  ss: MACHINE OF BITS (phase[3..0])
    WITH STATES ( s0 = B"0001",
                 s1 = B"0010",
                 s2 = B"0100",
                 s3 = B"1000");

BEGIN
  ss.clk = clk;
  ss.reset = reset;

  TABLE
    ss, ccw, cw => ss;
    s0, 1, x   => s3;
    s0, x, 1   => s1;
    s1, 1, x   => s0;
    s1, x, 1   => s2;
    s2, 1, x   => s1;
    s2, x, 1   => s3;
    s3, 1, x   => s2;
    s3, x, 1   => s0;

END TABLE;

END;
AHDL Structure - (1)

◆ **Title statement (optional)**
  • Comments for the report file generated by MAX+PLUS II Compiler

◆ **Include statement (optional)**
  • To specify an include file

◆ **Constant statement (optional)**
  • To specify a symbolic name that can be substituted for a constant

◆ **Define statement (optional)**
  • To define an evaluated function, which is a mathematical function that returns a value that is based on optional arguments

◆ **Parameters statement (optional)**
  • To declare one or more parameters that control the implementation of a parameterized megafunction or macrofunction
AHDL Structure - (2)

◆ Function prototype statement (optional)
  • To declare the ports of a logic function and the order in which those ports must be declared in an in-line reference
  • In parameterized functions, it also declares the parameters of the function

◆ Options statement (optional)
  • To set the default bit-ordering for the file

◆ Assert statement (optional)
  • To allow you to test the validity of an arbitrary expression

◆ Subdesign section (required)
  • To declare the input, output, and bidirectional ports of the design file
AHDL Structure - (3)

◆ **Variable statement (optional)**
  
  • To declare variables that represent and hold internal information
    
    – Instance declaration
    – Node declaration
    – Register declaration
    – State machine declaration
    – Machine alias declaration
    – If-Generate statement
AHDL Structure - (4)

逻辑部分（必需）
- 用于定义文件的逻辑操作
  - 默认声明
  - 断言声明
  - 布尔方程式
  - 布尔控制方程式
  - 案例声明
  - If-Generate 声明
  - If-Then 声明
  - 真值表声明
  - 在线逻辑函数引用
AHDL Basic Elements - (1)

◆ Numbers in AHDL
  - Default is to use decimal numbers
  - Binary number syntax: \texttt{B"<series of 0's, 1's, X's>"} (where \texttt{X} = "don't care")
  - Octal number syntax: \texttt{O"<series of digits 0 to 7>"} or \texttt{Q"<series of digits 0 to 7>"}
  - Hex number syntax: \texttt{H"<series of digits 0 to F>"} or \texttt{X"<series of digits 0 to F>"}

◆ Group
  - Sequential group: \texttt{(a, b, c)}
  - Single-range group: \texttt{a[4..1] = (a4, a3, a2, a1)}
  - Dual-range group: \texttt{d[2..0][1..0] = (d2_1, d2_0, d1_1, d1_0, d0_1, d0_0)}
  - Entire range group: \texttt{a[], d[]]}
AHDL Basic Elements - (2)

- Arithmetic/Boolean operators & comparators
  - +, -, ^(exponent), MOD, DIV, *, LOG2
  - ! (not), & (and), ! & (nand), # (or), ! # (nor), $ (xor), ! $ (xnor)
  - ==, !=, >, >=, <, <=
  - ? (ternary operator, e.g, (a<b) ? 3 : 4)
AHDL Basic Elements - (3)

Primitives

- I/O primitives (ports)
  - INPUT, OUTPUT, BIDIR
- Logic primitives
  - AND, NAND, OR, NOR, XNOR, XOR, NOT
- Buffer primitives
  - CARRY, CASCADE, EXP, GLOBAL, LCELL, OPNDRN, TRI
- Flip-flop & latch primitives
  - LATCH, DFF, DFFE, JKFF, JKFFE, SRFF, SRFFE, TFF, TFFE
- VCC & GND primitives
AHDL Basic Elements - (4)

◆ Ports
  • Ports of the current file
    – port types: INPUT, OUTPUT, BIDIR, MACHINE INPUT, MACHINE OUTPUT
  • Ports of instances:
    – Commonly used primitive ports names:
      .clk = clock input; .ena = latch/clock enable input;
      .reset = reset input to a state machine (active-high)
      .clrn = clear input (active-low); .prn = preset input (active-low);
      .d, .j, .k, .s, .r, .t = data input of D-, JK-, SR, and T-type flip-flop;
      .q = output of a flip-flop or latch

◆ MegafUNCTIONS/LPMs
◆ Old-style macrofunctions
◆ Parameters
AHDL Syntax  (1)

◆ **Title statement**
  - Documentary comments for the report file generated by the Compiler
  - Example:

    ```
    TITLE "Display Controller";
    ```
Parameters statement

- To declare one or more parameters that control the implementation of a parameterized megafunction of macrofunction
- Example:

```
PARAMETERS
(
    FILENAME = "myfile.mif",
    WIDTH,
    AD_WIDTH = 8,
    NUMWORDS = 2^AD_WIDTH
);
```
AHDL Syntax - (3)

◆ Include statement

• To import text from an include file (*.inc) into the current file
• Include files contains function prototype, define, parameters, or constant statements
• Compiler will search directories for include files in the following order:
  – Project directory
  – User libraries
  – Megafunctions/LPMs: \maxplus2\max2lib\mega_lpm
  – Macrofunctions: \maxplus2\max2inc
• Example:

  INCLUDE "8fadd.inc";

** The content of “8fadd.inc” file is:

  FUNCTION 8fadd (cin, a[8..1], b[8..1]) RETURNS (cout, sum[8..1]);

AHDL Syntax - (4)

◆ Constant statement
  • To substitute a meaningful symbolic name for a number or an arithmetic expression
  • Example:

    ```
    CONSTANT UPPER_LIMIT = 130;
    CONSTANT BAR = 1 + 2 DIV 3 + LOG2(256);
    ```
Define statement

- To define an evaluated function, which is a mathematical function that returns a value that is based on optional arguments
- Example:

```ahdl
DEFINE MAX(a,b) = (a > b) ? a : b;

SUBDESIGN
(
    dataa[MAX(WIDTH,0)..0]: INPUT;
    datab[MAX(WIDTH,0)..0]: OUTPUT;
)
BEGIN
    datab[] = dataa[];
END;
```
Function prototype statement

- To provide a shorthand description of a logic function, listing its name and ports
- Example:

% unparameterized function example %
FUNCTION compare (a[3..0], b[3..0])
    RETURNS (less, equal, greater);

% parameterized function example %
FUNCTION lpm_add_sub (cin, dataa[LPM_WIDTH-1..0],
    datab[LPM_WIDTH-1..0], add_sub)
    WITH (LPM_WIDTH, LPM_REPRESENTATION, LPM_DIRECTION,
        ADDERTYPE, ONE_INPUT_IS_CONSTANT)
    RETURNS (result[LPM_WIDTH-1..0], cout, overflow);
AHDL Syntax - (7)

◆ Options statement

• To specify whether the lowest numbered bit of a group will be the MSB, LSB or either, depending on its location
• Example:

    OPTIONS BIT0 = MSB;
**Assert statement**

- To test the validity of any arbitrary expression that uses parameters, numbers, evaluated functions, or the used or unused status of a port
- Severity level: **ERROR**, **WARNING** or **INFO**
- Example:

```vhdl
ASSERT (WIDTH > 0)
REPORT "Width (%) must be a positive integer" WIDTH
SEVERITY ERROR
HELP_ID INTVALUE; -- for internal Altera use only
```
AHDL Syntax - (9)

◆ Subdesign section

- To declare the input, output, and bidirectional ports of the TDF
- The port type may be \texttt{INPUT}, \texttt{OUTPUT}, \texttt{BIDIR}, \texttt{MACHINE} INPUT, or \texttt{MACHINE} OUTPUT
  - \texttt{MACHINE} INPUT \& \texttt{MACHINE} OUTPUT keywords are used to import and export state machines between TDFs and other design files. However, they cannot be used in a top-level TDF.
- Example:

  
  \begin{verbatim}
  SUBDESIGN top
  (  
    foo, bar, clk1, clk2 : INPUT = VCC;
    a0, a1, a2, a3, a4   : OUTPUT;
    b[7..0]              : BIDIR;
  )
  \end{verbatim}

AHDL Syntax - (10)

◆ Variable section
  - To declare and/or generate any variables used in the logic section
  - Example:

```vhdl
VARIABLE
  a, b, c : NODE;
  temp    : halfadd;
  ts_node : TRI_STATE_NODE;

IF DEVICE_FAMILY == "FLEX8000" GENERATE
  8kadder    : flex_adder;
  d, e       : NODE;
ELSE GENERATE
  7kadder    : pterm_adder;
  f, g       : NODE;
END GENERATE;
```
AHDL Syntax - (11)

◆ Variable section - node declaration
  • AHDL supports two types of nodes: NODE & TRI_STATE_NODE
  • Example:

    SUBDESIGN node_ex
    ( a, oe : INPUT;
      out : OUTPUT;
      c  : BIDIR;
    )

    VARIABLE
    b  : NODE;
    t  : TRI_STATE_NODE;

    BEGIN
    b  = a;
    out = b;  % therefore out = a  %
    t  = TRI(a, oe);
    t  = c;  % t is bus of c and tri_stated a  %

    END;
Variable section - instance declaration

- Each instance of a particular logic function can be declared as a variable
- Example:

FUNCTION compare (a[3..0], b[3..0])
  RETURNS (less, equal, greater);

FUNCTION lpm_add_sub (cin, dataa[LPM_WIDTH-1..0],
  datab[LPM_WIDTH-1..0], add_sub)
  WITH (LPM_WIDTH, LPM_REPRESENTATION, LPM_DIRECTION,
    ADDERTYPE, ONE_INPUT_IS_CONSTANT)
  RETURNS (result[LPM_WIDTH-1..0], cout, overflow);

VARIABLE
  comp : compare;
  adder : lpm_add_sub with (LPM_WIDTH = 8)
AHDL Syntax - (13)

◆ Variable section - register declaration
  • You can declare registers including D, T, JK, SR flip-flops & latches
    – DFF, DFFE, TFF, TFFE, JKFF, JKFFE, SRFF, SRFFE, LATCH
  • Example:

    ```
    VARIABLE
    ff : TFF;
    a, b : DFF;
    ```

    *ff is a T flip-flop and have the following ports:
    ff.t, ff.clk, ff.clrn, ff.prn, ff.q*

    *Since all primitives have only one output, you can use the name of an instance of a primitive without appending a port name. For example, “a = b” is equivalent to “a.d = b.q”*
AHDL Syntax - (14)

◆ Variable section - state machine declaration
  • To create a state machine by declaring its name, states, and, optionally its bits
  • Example:

```
VARIABLE
  ss : MACHINE OF BITS (q1, q2, q3)
  WITH STATES
  ( s1 = B"000",
    s2 = B"010",
    s3 = B"111"
  );
```

Reset state for the state machine
Variable section - machine alias declaration

- To rename a state machine with a temporary name
- Example:

```
FUNCTION ss_def (clock, reset, count)
    RETURNS (MACHINE ss_out);

VARIABLE
    ss : MACHINE;

BEGIN
    ss = ss_def (sys_clk, reset, !hold);
    IF ss == s0 THEN
        :
    ELSIF ss == s1 THEN
        :
    END;
```

`ss_out` is a state machine output
AHDL Syntax - (16)

◆ Logic section
  • To specify the logical operations of the TDF
  • The `BEGIN` and `END` keywords enclose the logic section

◆ Boolean equations
  • To represent the connection of nodes and the dataflow of inputs and outputs
  • Example:
    ```
    a[] = ((c[] & -B"001101") + e[6..1]) # (p, q, r, s, t, v);
    chip_enable = (address[15..0] == H"0370");
    ```

◆ Boolean control equations
  • Set up the state machine clock, reset, and clock enable signals
  • Example:
    ```
    ss.clk = clk1;
    ss.reset = a & b;
    ss.ena = clk1ena;
    ```
AHDL Syntax - (17)

◆ Case statement
  • Example:

```
CASE f[].q IS
  WHEN H"00" =>
    addr[] = 0;
    s = a & b;
  WHEN H"01" =>
    count[].d = count[].q + 1;
  WHEN H"02", H"03", H"04" =>
    f[3..0].d = addr[4..1];
  WHEN OTHERS =>
    f[].d = f[].q;
END CASE;
```

To define the default behavior
AHDL Syntax - (18)

**Defaults statement**

- To specify default values for variables used in truth table, if-then and case statements
- Example:

```ahdl
BEGIN
  DEFAULTS
  a = VCC;
END_DEFAULTS;

IF y & z THEN
  a = GND;  % a is active low %
END IF;
END;
```

*Only one Defaults statement is allowed in the Logic Section, and it must be the first statement after the BEGIN keyword. Defaults statement can't be used to set a default value of X (don't care) to a variable.*

*Active-low variables that are assigned more than once should be given a default value of VCC*
AHDL Syntax - (19)

◆ If-Then statement

   • Example:

```vhdl
IF a[] == b[] THEN
  c[8..1] = H "77";
  addr[3..1] = f[3..1].q;
  f[].d = addr[] + 1;
ELSIF g3 $ g4 THEN
  f[].d = addr[];
ELSE
  d = VCC;
END IF;
```

AhDL Syntax - (20)

**If-Generate statement**

- To list a series of behavioral statements that are activated after the positive evaluation of an arithmetic expression
  - Can be used in the logic section or in the variable section
  - If-Then statement is evaluated in hardware, whereas If-Generate statement is evaluated when the design is compiled
- The predefined parameter and evaluated function
  - `DEVICE_FAMILY`: to test the current device family for the project
  - `USED`: to test whether an optional port has been used in the current instance.
- Example:

```ahdl
IF DEVICE_FAMILY == "FLEX8K" GENERATE
  c[] = 8kadder(a[], b[], cin);
ELSE
  c[] = otheradder(a[], b[], cin);
END GENERATE;
```
AHDL Syntax  - (21)

◆ For-Generate statement
  • Example:

  CONSTANT NUM_OF_ADDERS = 8;
  SUBDESIGN 4gentst
  ( a[NUM_OF_ADDERS..1], b[NUM_OF_ADDERS..1], cin : INPUT;
    c[NUM_OF_ADDERS..1], cout : OUTPUT;
  )
  VARIABLE
    carry_out[(NUM_OF_ADDERS+1)..1] : NODE;
  BEGIN
    carry_out[1] = cin;
    FOR i IN 1 TO NUM_OF_ADDERS GENERATE
      c[i] = a[i] $ b[i] $ carry_out[i];
      carry_out[i+1] = a[i] & b[i] # carry_out[i] & (a[i] $ b[i]);
    END GENERATE;
    cout = carry_out[NUM_OF_ADDERS+1];
  END;
**In-line logic function reference**

- A Boolean equation that implements a logic function
- Example:

```vhdl
FUNCTION compare (a[3..0], b[3..0])
    RETURNS (less, equal, greater);

FUNCTION lpm_add_sub (cin, dataa[LPM_WIDTH-1..0],
                        datab[LPM_WIDTH-1..0], add_sub)
    WITH (LPM_WIDTH, LPM_REPRESENTATION)
    RETURNS (result[LPM_WIDTH-1..0], cout, overflow);

(cw, , ccw) = compare(position[], target[]);

sum[] = lpm_add_sub (.datab[] = b[], .dataa[] = a[])
    WITH (LPM_WIDTH = 8)
    RETURNS (.result[]);
```
**AHDL Syntax** - (23)

◆ **Truth table statement**
  - To specify combinational logic or state machine behavior
  - Use defaults statement to assign output values in cases when the actual inputs do not match the input values of the table
  - When using X (don't care) characters to specify a bit pattern, you must ensure that the pattern cannot assume the value of another bit pattern in the truth table. AHDL assumes that only one condition in a truth table is true at a time.
  - Example:

```plaintext
TABLE
  a0, f[4..1].q => f[4..1].d, control;

  0, B"0000"   => B"0001",   1;
  0, B"0100"   => B"0010",   0;
  1, B"0XXX"   => B"0100",   0;
  X, B"1111"   => B"0101",   1;

END TABLE;
```
AHDL Details

◆ To know more about AHDL
  • Refer to Altera’s AHDL manual
  • Search relative topics in MAX+PLUS II on-line help

◆ More AHDL examples
  • Find AHDL examples under \max2work\ahdl directory

◆ To make writing AHDL code easy
  • Use MAX+PLUS II Text Editor to edit your TDFs
  • Using LPM
MAX+PLUS II Text Editor

Features of MAX+PLUS II Text Editor

- AHDL templates & examples
- AHDL context-sensitive help
- Syntax coloring
- Error location
- Resource & device assignments
AHDL Templates

AHDL templates make design easier

- You can insert AHDL template into your TDF, then replace placeholder variables in the templates with your own identifiers and expressions

Menu: Templates -> AHDL Template...
Inserting AHDL Template
Using Syntax Coloring

◆ Syntax Coloring command
  • To improve TDF readability & accuracy
    Menu: Options -> Syntax Coloring

◆ To customize the color palette
  Menu: Options -> Color Palette...
  • The AHDL-relative options:
    – Comments
    – Illegal Characters
    – Megafucntions/Macrofunctions
    – Reserved Identifiers
    – Reserved Keywords
    – Strings
    – Text
CONSTANT NORTH = B"00"; % Create descriptive name for numbers
CONSTANT EAST = B"01"; % for use elsewhere in file
CONSTANT WEST = B"10";
CONSTANT SOUTH = B"11";
SUBDESIGN auto_max
{
    dir[1..0], accel, clk, reset : INPUT; % File inputs
    speed_too_fast, at_altera, get_ticket : OUTPUT; % File output
}

VARIABLE

street_map : MACHINE % Create state machine with

    OF BITS (q2,q1,q0) % q1 & q0 as outputs of register

WITH STATES {
    yc, % Your company
    mplt, % Marigold Park Lane Drive
    epld, % East Pacific Lane Drive
    gd, % Great Delta Freeway
    cnf, % Capitol North First
    rtp, % Regal Park Terrace
    epm, % East Pacific Main
}

Line 4 Col 24 INS
Creating Text Design Files

- **Open a new design file**
  
  Menu: `File -> New... -> Text Editor file (.tdf)`

- **Save as a TDF file**
  
  Menu: `File -> Save As...`

- **Set project to the current TDF file**
  
  Menu: `File -> Project... -> Set Project to Current File`

- **Edit the TDF**
  
  - Turn on syntax coloring option
  - Use AHDL Template & on-line help if necessary
  - Follow the AHDL style guide mentioned in MAX+PLUS II Help

- **Save the file & check for basic errors**
  
  Menu: `File -> Project -> Project Save & Check`
Example: Decoder

Design a decoder with...

- If-Then statements
- Case statements
- Table statements
- LPM function: LPM_DECODE

```plaintext
SUBDESIGN decoder
(
  code[1..0] : INPUT;
  out[3..0]  : OUTPUT;
)
BEGIN
  CASE code[] IS
    WHEN 0 => out[] = B"0001";
    WHEN 1 => out[] = B"0010";
    WHEN 2 => out[] = B"0100";
    WHEN 3 => out[] = B"1000";
  END CASE;
END;
```

```plaintext
SUBDESIGN priority
(
  low, middle, high : INPUT;
  highest_level[1..0] : OUTPUT;
)
BEGIN
  IF high THEN
    highest_level[] = 3;
  ELSIF middle THEN
    highest_level[] = 2;
  ELSIF low THEN
    highest_level[] = 1;
  ELSE
    highest_level[] = 0;
  END IF;
END;
```
Example: Counter

Create a counter with **DFF/DFFE or LPM_COUNTER**

```vhdl
SUBDESIGN ahdicnt
(
    clk, load, ena, clr, d[15..0] : INPUT;
    q[15..0] : OUTPUT;
)
VARIABLE
    count[15..0] : DFF;
BEGIN
    count[].clk = clk;
    count[].clrn = !clr;
    IF load THEN
        count[].d = d[];
    ELSIF ena THEN
        count[].d = count[].q + 1;
    ELSE
        count[].d = count[].q;
    END IF;
    q[] = count[];
END;
```
Example: Multiplier

◆ **Design a multiplier with LPM_MULT**

```verbatim
CONSTANT WIDTH = 4;
INCLUDE "lpm_mult.inc";

SUBDESIGN tmul3t
(
 a[WIDTH-1..0]      : INPUT;
 b[WIDTH-1..0]      : INPUT;
 out[2*WIDTH-1..0]  : OUTPUT;
)

VARIABLE
  mult : lpm_mult WITH (LPM_REPRESENTATION="SIGNED",
                        LPM_WIDTHA=WIDTH,  LPM_WIDTHB=WIDTH,
                        LPM_WIDTHS=WIDTH,  LPM_WIDTHP=WIDTH*2);

BEGIN
  mult.dataa[] = a[];
  mult.datab[] = b[];
  out[] = mult.result[];
END;
```
Example: Multiplexer

◆ Design a multiplexer with \texttt{LPM_MUX}

\begin{verbatim}
FUNCTION lpm_mux (data[LPM_SIZE-1..0][LPM_WIDTH-1..0], sel[LPM_WIDTHS-1..0])
  WITH (LPM_WIDTH, LPM_SIZE, LPM_WIDTHS, CASCADE_CHAIN)
RETURNS (result[LPM_WIDTH-1..0]);

SUBDESIGN mux
  (a[3..0], b[3..0], c[3..0], d[3..0]  : INPUT;
   select[1..0]                        : INPUT;
   result[3..0]                        : OUTPUT;
  )
BEGIN
  result[3..0] = lpm_mux (a[3..0], b[3..0], c[3..0], d[3..0], select[1..0])
  WITH (LPM_WIDTH=4, LPM_SIZE=4, LPM_WIDTHS=2);
END;
\end{verbatim}
Example: RAM

◆ Design RAM circuit with LPM

```
INCLUDE "lpm_ram_dq.inc";

SUBDESIGN ram_dq
(
    clk : INPUT;
    we : INPUT;
    ram_data[31..0] : INPUT;
    ram_add[7..0] : INPUT;
    data_out[31..0] : OUTPUT;
)
BEGIN
    data_out[31..0] = lpm_ram_dq (ram_data[31..0], ram_add[7..0], we, clk, clk)
        WITH (LPM_WIDTH=32, LPM_WIDTHAD=8);
END;
```
Example: Tri-State Buses

◆ Design tri-state buses with TRI

SUBDESIGN tribus
(
    ina[7..0], inb[7..0], inc[7..0], oe_a, oe_b, oe_c, clock : INPUT;
    out[7..0] : OUTPUT;
)

VARIABLE
    flip[7..0] : DFF;
    tri_a[7..0], tri_b[7..0], tri_c[7..0] : TRI;
    mid[7..0] : TRI_STATE_NODE;

BEGIN
    -- Declare the data inputs to the tri-state buses
    tri_a[] = ina[];  tri_b[] = inb[];  tri_c[] = inc[];
    -- Declare the output enable inputs to the tri-state buses
    tri_a[].oe = oe_a;  tri_b[].oe = oe_b;  tri_c[].oe = oe_c;
    -- Connect the outputs of the tri-state buses together
    mid[] = tri_a[];  mid[] = tri_b[];  mid[] = tri_c[];
    -- Feed the output pins
    flip[].d = mid[];  flip[].clk = clock;  out[] = flip[].q;
END;
Example: Moore State Machine

Moore state machine

- The outputs of a state machine depend only on the state

```vhdl
SUBDESIGN moore1
(
  clk : INPUT;
  reset : INPUT;
  y : INPUT;
  z : OUTPUT;
)
VARIABLE
ss: MACHINE OF BITS (z)
  WITH STATES (s0 = 0, s1 = 1, s2 = 1, s3 = 0);
% current_state =
current_output%
BEGIN
  ss.clk = clk;
  ss.reset = reset;
  TABLE
    ss, y => ss;
    s0, 0 => s0;
    s0, 1 => s2;
    s1, 0 => s0;
    s1, 1 => s2;
    s2, 0 => s2;
    s2, 1 => s3;
    s3, 0 => s3;
    s3, 1 => s1;
END TABLE;
END;
```
Mealy state machine

- A state machine with asynchronous output(s)

```vhdl
SUBDESIGN mealy
(
    clk  : INPUT;
    reset : INPUT;
    y    : INPUT;
    z    : OUTPUT;
)
VARIABLE
    ss: MACHINE WITH STATES (s0, s1, s2, s3);
BEGIN
    ss.clk = clk;
    ss.reset = reset;
    TABLE
        ss, y => z, ss;
        s0, 0 => 0, s0;
        s0, 1 => 1, s1;
        s1, 0 => 1, s1;
        s1, 1 => 0, s2;
        s2, 0 => 0, s2;
        s2, 1 => 1, s3;
        s3, 0 => 0, s3;
        s3, 1 => 1, s0;
    END TABLE;
END;
```
Waveform Design Entry

- MAX+PLUS II Waveform Editor
- Creating Waveform Files
- Examples
- Design Entry Summary
MAX+PLUS II Waveform Editor

◆ Features of MAX+PLUS II Waveform Editor
  • To serve 2 roles:
    – As a design entry tool: to create Altera waveform design files (*.wdf)
    – As a tool for entering test vectors & viewing simulation results: simulation channel files (*.scf)

◆ For design entry
  • Waveform design entry is best suited for circuits with well-defined sequential inputs & outputs, such as state machines, counters, and registers

◆ For design verification
  • Waveform Editor is a simulation pattern editor/viewer
  • Waveform Editor is fully integrated with MAX+PLUS II Simulator & Programmer to provide full project verification flow
MAX+PLUS II
Waveform Design Environment
File Menu
Node Menu

To enter the node or group information
Edit Menu

To edit the waveform value
Creating a New Waveform File

◆ Open a new design file
  Menu: File -> New... -> Waveform Editor file (.wdf or .scf)

◆ Save as a WDF / SCF file
  Menu: File -> Save As... ->

◆ Set project to current file (for WDF file only)
  Menu: File -> Project... -> Set Project to Current File
Setting Waveform Editor Options

◆ Set the grid size & show the grid
  Menu: Options -> Grid Size...
  Menu: Options -> Show Grid
  • Setting appropriate grid size is helpful for waveform repeating & overwriting count value operations

◆ Specify the end time
  Menu: File -> End Time...

◆ Regarding the grid size & interval...
  • In a WDF, the grid size & interval are arbitrary. The time scale indicates only a sequential order of operations, not a specific response time.
  • In a SCF, the grid size & interval are important for timing simulation. MAX+PLUS II Simulator reflects the real-world timing according to your SCF and the specific device. Setup & Hold time violation will occur if you enter impractical simulation patterns.
Entering Nodes

◆ Insert the node or group for WDF file

Menu: Node -> Insert Node... (or double click on the node name field)

- You can specify the node name, I/O type, node type & default value
  - Registered & machine node type must specify a clock signal and optionally specify reset or preset signal (active high)
  - You can specify machine values with the state names instead of logic values
Entering Nodes from SNF

◆ Enter the node or group for SCF file

Menu: Node -> Enter Nodes from SNF...

- SNF: Simulation Netlist File
  - Generated by MAX+PLUS II Compiler (discussed later)
  - After compilation, you can list the nodes and help you to create the SCF file
◆ **Edit the waveforms**

- First select the interval to edit
  - Sometimes you may specify new grid size for easy selection
- To create clock-like waveform
  
  **Menu:** *Edit -> Overwrite -> Clock...*

- To edit the state machine node values
  
  **Menu:** *Edit -> Overwrite State Name...*
Edit the waveforms

- To edit the node values
  
  **Menu:** Edit -> Overwrite -> 0 / 1 / X / Z / Invert / Count Value / Group Value

- To stretch / compress the selected signal
  
  **Menu:** Edit -> Grow or Shrink...

- To align node values or state names to grid if necessary

  **Menu:** Edit -> Align to Grid
Saving & Checking the Design

- Save the WDF/SCF file
  Menu: File -> Save

- Check basic errors for the WDF file
  Menu: File -> Project -> Project Save & Check
Waveform File Formats

◆ MAX+PLUS II file formats
  • Binary format: WDF & SCF files
  • ASCII format (Altera vector file format): TBL & VEC files
    – TBL: an ASCII-format table file that records all logic level transitions for nodes and groups in the current SCF or WDF
    – VEC: an ASCII text file used as the input for simulation, functional testing, or waveform design entry
    – Refer to MAX+PLUS II Help for detailed information about vector file format

◆ To create a table file (*.tbl)
  Menu: File -> Create Table File...

◆ To import a vector file (*.vec)
  Menu: File -> Import Vector File...
WDF Design Guidelines

◆ When design a WDF file...
  • WDFs cannot be at intermediate levels of a hierarchy
  • Include all possible combinations of input values
  • Align all logic level and state name transition
  • Assume a 0ns propagation delay for all logic
  • Assume a 0.1ns setup time and 0ns hold time for state machine node
  • For clarity, Altera recommends that you draw inputs that affect registers only on falling clock edges
  • If a function is cyclical, show the last set of conditions looping back to the first by repeating the first time-slice at the end of the cycle
Example: Decoder

◆ When design a decoder...
  • Use “Overwrite Count Value” to help create all possible combinations of decoder input values, and then manually edit the output waveforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>50 0ns</th>
<th>100 0ns</th>
<th>150 0ns</th>
<th>200 0ns</th>
<th>250 0ns</th>
<th>300 0ns</th>
<th>350 0ns</th>
<th>400 0ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>in[3..0]</td>
<td>INPUT</td>
<td>F</td>
<td>E</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>decout[15..0]</td>
<td>COMB</td>
<td>8000</td>
<td>4000</td>
<td>2000</td>
<td>1000</td>
<td>0800</td>
<td>0400</td>
<td>0200</td>
<td>0100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0080</td>
<td>0040</td>
<td>0020</td>
<td>0010</td>
<td>0008</td>
<td>0004</td>
<td>0002</td>
<td>0001</td>
</tr>
</tbody>
</table>
Example: Counter

◆ When design a counter
  ● Use “Overwrite Count Value” command to create a regular counter waveform
Example: State Machine

◆ When design a state machine
  • Use “Overwrite State Name” to help create a state machine output
    – You can specify machine values with the state names instead of logic values
  • Make sure all possible combinations of inputs and states are included
Design Entry Summary

MAX+PLUS II

3rd Party EDA Tools

Design Files
- .gdf
- .tdf
- .wdf
- .vhd
- .sch
- .edf

Support Files
- .sym
- .v
- .inc
- .lmf
Design Implementation

- MAX+PLUS II Compiler
- Preparing for Compilation
- Compiling the Project
- Analyzing the Compilation Results
- Floorplan Editor
- Appendix: Interfacing with 3rd-Party Tools
MAX+PLUS II Compiler Window

To invoke MAX+PLUS II Compiler
Menu: MAX+PLUS II -> Compiler
MAX+PLUS II Compiler

◆ Process all design files associated with the project
  • Files can be created with MAX+PLUS II or 3rd party EDA Tools

◆ Checks for syntax errors and common design pitfalls

◆ Performs logic synthesis and place & route
  • According to assignments in .acf file

◆ Generates files for simulation and timing analysis
  • Files can be used by MAX+PLUS II or 3rd party EDA Tools

◆ Generates files for programming targeted devices
Compiler Input and Output Files

MAX+PLUS II

- Design Files (.gdf, .tdf, .vhd, .v, .wdf)
- Assignments (.acf)

MAX+PLUS II Compiler

- Compiler Netlist
  - Extractor (includes all netlist readers)
- Database Builder
- Logic Synthesizer
- Functional, Timing, or Linked SNF Extractor
- EDIF, VHDL & Verilog Netlist Writers
- Partitioner
- Fitter
- Design Doctor
- Assembler

Mapping Files (.imf)

- Functional SNF Files (.snf)
- Timing SNF Files (.snf)
- Programming Files (.pof, .sof, .jed)

3rd Party EDA Design Files (.edf, .sch)

3rd Party EDA Simulation/Timing Files (.edo, vo, vho, sdo)
Compiler Input Files

**Design files**
- MAX+PLUS II
  - Graphics file (.gdf), AHDL file (.tdf), VHDL file (.vhd), Verilog (.v), Waveform file (.wdf)
- 3rd Party EDA Tools
  - EDIF file (.edf)
    - Select Vendor in EDIF Netlist Reader Settings
    - Library Mapping File (.lmf) required for vendors not listed
  - OrCAD file (.sch)

**Assignment and Configuration File (.acf)**
- Controls the Compiler’s synthesis and place & route operations
- Automatically generated when user enter assignments
- Automatically updated when user changes assignments or back-annotates project
Compiler Output Files

◆ Design verification files
  • MAX+PLUS II
    – Simulation Netlist File (.snf)
  • 3rd Party EDA Tools
    – VHDL netlist file (.vho)
    – EDIF netlist file (.edo)
    – Verilog netlist file (.vo)
    – Standard Delay Format SDF file (.sdo)

◆ Programming files
  • Programmer Object file (.pof)
  • SRAM Object file (.sof)
  • JEDEC file (.jed)
For EDIF Netlist Input

For EDIF input, the EDIF Reader Settings need to be selected.
VHDL Netlist Reader Settings

Select VHDL version and Library names
For EDIF Netlist Output

Need to select vendor in EDIF Netlist Writer settings.

For EDIF output, the EDIF Netlist Writer needs to be selected.
Verilog Netlist Writer & Writer Settings

Select Verilog Netlist Writer Settings to adjust Verilog outputs

Select Verilog Netlist Writer to output Verilog file for Verilog simulator
Select VHDL Netlist Writer Settings to adjust the VHDL output.

Select VHDL Netlist Writer to output VHDL file for VHDL simulator.
Imported Design

◆ Top-level Design: can be read in directly
  – EDIF Netlist files
  – OrCAD schematics
    • Refer to MAX+PLUS II Read Me file for the version of 3rd Parties tools it interface with

◆ Lower-level modules
  • EDIF, OrCAD schematics files
    – Create symbols or files to instantiate component
  • Other proprietary files
    – JEDEC, ABEL, PALASM
    – Conversion utilities exist in Altera ftp site
Compiler Modules - (1)

Compiler Netlist Extractor

- The Compiler module that converts each design file in a project (or each cell of an EDIF input file) into a separate binary CNF (Compiler Netlist File)
- The Compiler Netlist Extractor also creates a single HIF that documents the hierarchical connections between design files
- This module contains a built-in EDIF Netlist Reader, VHDL Netlist Reader, and XNF Netlist Reader for use with MAX+PLUS II.
- During netlist extraction, this module checks each design file for problems such as duplicate node names, missing inputs and outputs, and outputs that are tied together.
- If the project has been compiled before, the Compiler Netlist Extractor creates new CNFs and a HIF only for those files that have changed since the last compilation, unless Total Recompile (File menu) is turned on.
Compiler Modules - (2)

- Database Builder
  - The Compiler module that builds a single, fully flattened project database that integrates all the design files in a project hierarchy.
  - As it creates the database, the Database Builder examines the logical completeness and consistency of the project, and checks for boundary connectivity and syntactical errors (e.g., a node without a source or destination).
Compiler Modules - (3)

◆ Logic Synthesizer
  • The Compiler module that synthesizes the logic in a project's design files.
  • The Logic Synthesizer calculates Boolean equations for each input to a primitive and minimizes the logic according to your specifications.
  • The Logic Synthesizer also synthesizes equations for flip-flops to implement state registers of state machines.
  • As part of the logic minimization and optimization process, logic and nodes in the project may be changed or removed.
  • Throughout logic synthesis, the Logic Synthesizer detects and reports errors such as illegal combinatorial feedback and tri-state buffer outputs wired together ("wired ORs").

◆ Design Doctor Utility
  • The Compiler utility that checks each design file in a project for poor design practices that may cause reliability problems when the project is implemented in one or more devices.
Compiler Modules - (4)

Partitioner

- The Compiler module that partitions the logic in a project among multiple devices from the same device family.
- Partitioning occurs if you have created two or more chips in the project's design files or if the project cannot fit into a single device.
- This module splits the database updated by the Logic Synthesizer into different parts that correspond to each device.
- A project is partitioned along logic cell boundaries, with a minimum number of pins used for inter-device communication.
Compiler Modules - (5)

Fitter

- The Compiler module that fits the logic of a project into one or more devices
- Using the database updated by the Partitioner, the Fitter matches the logic requirements of the project with the available resources of one or more devices
- It assigns each logic function to the best logic cell location and selects appropriate interconnection paths and pin assignments
- The Fitter module generates a “fit file” (*.fit) that documents pin, buried logic cell, chip, clique, and device assignments made by the Fitter module in the last successful compilation
- Regardless of whether a fit is achieved, the Fitter generates a report file (*.rpt) that shows how the project is implemented in one or more devices
Compiler Modules - (6)

SNF (Simulation Netlist File) Extractor

- Functional SNF Extractor
  - The Compiler module that creates a functional SNF containing the logic information required for functional simulation.
  - Since the functional SNF is created before logic synthesis, partitioning, and fitting are performed, it includes all nodes in the original design files for the project.

- Timing SNF Extractor
  - The Compiler module that creates a timing SNF containing the logic and timing information required for timing simulation, delay prediction, and timing analysis.
  - The timing SNF describes a project as a whole. Neither timing simulation nor functional testing is available for individual devices in a multi-device project.

- Linked SNF Extractor
  - The Compiler module that creates a linked SNF containing timing and/or functional information for several projects.
  - A linked SNF of a super-project combines the timing and/or functional information for each project, allowing you to perform a board-level simulation.
**Netlist Writer**

- **EDIF Netlist Writer**
  - The Compiler module that creates one or more EDIF output files (*.edo). It can also generate one or more optional SDF output files (*.sdo).
  - EDIF output Files contain the logic and timing information for the optimized project and can be used with industry-standard simulators. An EDIF Output File is generated for each device in a project.

- **Verilog Netlist Writer**
  - The Compiler module that creates one or more Verilog output files (*.vo). It can also generate one or more optional SDF output files.

- **VHDL Netlist Writer**
  - The Compiler module that creates one or more VHDL output files (*.vho). It can also generate one or more optional VITAL-compliant SDF output files.
Assembler

- The Compiler module that creates one or more programming files for programming or configuring the device(s) for a project
- The assembler generates one or more device programming files
  - POFs and JEDEC Files are always generated
  - SOFs, Hex Files, and TTFs are also generated if the project uses FLEX devices
  - You can generate additional device programming files for use in other programming environment. For example, you can create SBF and RBF to configure FLEX devices.
- File format:
  - POF: Programming Object File
  - SOF: SRAM Object File
  - TTF: Tabular Text File
  - HEX: Intel-format Hexadecimal File
  - SBF: Serial Bitstream File
  - RBF: Raw Binary File
Compiling a Project

- Select functional compilation or timing compilation
- Assignments
- Run the compilation
- Consult the report file (.rpt) or the Floorplan Editor for device utilization summaries and synthesis and place & route results
The Functional Compilation Process

- Compiler Netlist Extractor builds the .cnf netlist file and checks for syntax errors
- Database Builder constructs the node name database
- Functional SNF Extractor builds .snf file for functional simulation
The Timing Compilation Process

- Compiler Netlist Extractor and Database Builder build netlist database and check for syntax errors.
- Logic Synthesizer performs logic synthesis/minimization.
- Design Doctor checks for design violations.
- Partitioner and Fitter execute place & route algorithm and builds the .rpt file on device implementation.
- Timing SNF Extractor builds .snf file for simulation and timing analysis.
- Assembler builds files for programming the device.
Compiler Processing Options

◆ Functional
  • Compilation generates file for Functional Simulation
    – Functional SNF file (.snf)

◆ Timing
  • Compilation generates user selectable files for
    – Timing Simulation and Timing Analysis
      • Timing SNF file (.snf)
    – 3rd party EDA Simulation
      • Verilog file (.vo)
      • VHDL file (.vho)
      • SDF file (.sdo)
    – Device Programming
      • Altera Programmer file (e.g. .pof, .sof)
Compilation Process Settings - (6)

◆ Customize the report file settings

Menu: Processing -> Report File Settings...
Compilation Process Settings - (7)

◆ “Smart Recompile” & “Total Recompile”
  - The first time the Compiler processes a project, all design files of that project are compiled
  - Use “Smart Recompile” feature to create an expanded project database that helps to accelerate subsequent compilations
    - Allow you to change physical device resource assignments without rebuilding the database & resynthesizing the project
  - Use “Total Recompile” feature to force the Compile to regenerate database & resynthesize the project

  Menu: Processing -> Smart Recompile
  Menu: Processing -> Total Recompile
Assign Menu

You can specify optional sections to be included in the report file (*.rpt), which is created by the Filter when a project is compiled.

All sections are included by default.
Assignments Control

◆ **Device FIT**
  - MAX+PLUS II default settings are designed for maximum fit-ability
  - Almost all assignments affect fitting

◆ **Device Utilization**
  - Circuit design
  - Logic assignment

◆ **Performance**
  - Circuit design
  - Logic assignments
  - Logic placements
Assignments

◆ Most common Assignments
  - Device assignments
  - Pin assignments

◆ Other assignments
  - Logic options
  - architectural features
  - Location assignments
    - Lab, Row, Column, LC
  - Clique
  - timing assignments
  - Device Option assignments
Making Device Assignment

◆ Select Device
  - Specific device
  - Auto
    - MAX+PLUS II chooses smallest and fastest device the design fits into
Making Pin Assignment

◆ Highlight node in graphic or text source file
  - Assign > Pin/Location/Chip
◆ Floorplan Editor can also be used (discussed later)

Highlight node and choose Assign Pin/Location/Chip
Node name automatically entered in the Node Name field
Choose pin or LCELL location then click on Add to enter assignment
(Note: You must choose a specific device prior to this step)
Logic Synthesis Style

◆ The most common way toward adjusting these assignments is to apply the predefined Logic Synthesis Style toward the different portion of your design:

• Normal
• Fast
• WYSIWYG

◆ Each of the Logic Synthesis Styles is a collection of both logic synthesis options and individual architectural settings
Global Project Logic Synthesis Style

◆ Choose Assign then Global Project Logic Synthesis

◆ Select from predefined synthesis style
  • NORMAL (default), FAST or WYSIWYG

◆ Or create user tailored settings
Assign Logic Synthesis Style Locally

Select Logic Synthesis Style
Individual Logic Option Assignment

- Provides controls to turn individual architectural features and synthesis algorithms on or off

- Gray or Default (default): set by higher level or global setting
- Check or Auto: enable feature
- Blank or Ignore: disable feature
Location assignments

Select Pin/Location/Chip ...

Select Location
Clique Assignments

Clique assignments tell the compiler to place the nodes with the same clique assignment close together inside the device.
Timing Requirements Assignments

*FLEX devices only*

- Specifies desired speed performance
- Use after performing timing analysis to improve specific timing path
- Localized control
  - Highlight node, pin or logic block
  - Choose Assign then Timing Requirements
  - Assign desired tpd, tco, tsu, fmax values
- Global control
  - Choose Assign then Global Project Timing Requirements
  - Assign desired tpd, tco, tsu, fmax values
Assignment Recommendation

- Start with device and pin assignments. Beware, your pin assignments might affect performance. Ideally, you should let MAX+PLUS II choose the pin assignments. If you have pin assignments, you might want to compile your design once without your pin assignments to see if they affect your performance.

- Compile design. Check device utilization and performance.

- If you need to adjust device utilization or performance try the other assignments. Try the synthesis style assignments first.

- Assignments can only be made to “hard” nodes or lower-level designs that contains hard nodes. Hard nodes are objects that translate directly into objects in silicon e.g. Flip-flops, LCELLs and I/O pins.
Ignore or Clear Assignments

Ignore Project Assignments
Clear Project Assignments
Global Project Device Options Window contains options related to the operation of the device rather than options that affect the logic synthesis and place & route of the design.

For example,

**FLEX Device**
- configuration scheme
- multi-volt I/O

**MAX Device**
- Enable JTAG support
- security bit
More Compiler Processing Options

◆ Design Doctor
  • Checks for common design errors

◆ Fitter Settings
  • Set place & route options

◆ Smart Recompile
  • Faster compilation time

◆ Total Recompile
  • Recompile every file
Compile the Design

- Start Button starts compilation process
- Messages are displayed by the Message Processor
  - Info
  - Warning
  - Error

Start Compilation

Messages
The Report File

- Project summary
  - Device assignments
  - Error summary
  - Device pin-out diagram (useful for PCB layout)
- Resource utilization
  - Pin
  - LCELL
  - Equations
- Compiler resources
  - Compilation time
  - Memory usage

Open report file by double clicking on the rapt icon
Checking the Messages

◆ Check the messages in Message Processor
  • In Message Processor window, choose the message and click the HELP on Message to understand the meaning of the message, its cause and the possible solutions (suggested actions)

◆ Error location
  • In Message Processor window, choose the message and click the Locate button to locate the source of the message in the original design files
  • You can turn on Locate in Floorplan Editor and click Local All button to find the corresponding nodes in the Floorplan Editor
Help on Message

Node <name> has assignments but doesn't exist or is a primitive array

CAUSE: You assigned resources to a node that does not exist. The node either never existed in the project's design files, or was removed during logic synthesis and minimization. Or, you may have entered a resource assignment on a primitive that is connected to one or more buses to form a primitive_array. This message may also occur if you assigned resources to a macro_function that does not contain any hard_logic_functions when you used pre-version 5.0 synthesis.

ACTION: Check the design files for accuracy. Remove unused resource_assignments and make sure you have not entered assignments on symbols in a primitive array. Refer to Guidelines For Working with Assignments for more information on how to enter assignments for a primitive array.

See also:
- Deleting an Assignment
- MAX+PLUS II Messages
- Message Format
- Partitioning a Project
Checking the Reports

◆ Check the report file
  • Use Text Editor or double click the Report File icon
  • Device summary, project compilation messages, file hierarchy, resource usage, routing resources, logic cell interconnections, ...
**DEVICE SUMMARY**

<table>
<thead>
<tr>
<th>Chip/Device</th>
<th>Input</th>
<th>Output</th>
<th>Ebid</th>
<th>Lcs</th>
<th>% Utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td>chiptrip EPF8282A/LC04-2</td>
<td>6</td>
<td>13</td>
<td>0</td>
<td>80</td>
<td>38%</td>
</tr>
<tr>
<td>User Pins:</td>
<td>6</td>
<td>13</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PROJECT COMPILATION MESSAGES**

Warning: Ignored all pin assignments as requested in the Ignore Project Assignments...
Pin-out file (.pin)

An ASCII file that contains the pin out of your device. It is created as a pin-out file for a board layout tool.

| N.C.  = Not Connected.         |
| VCCINT = Dedicated power pin, which MUST be connected to VCC (5.0 volts). |
| VCCIO = Dedicated power pin, which MUST be connected to VCC (5.0 volts). |
| GNDINT = Dedicated ground pin or unused dedicated input, which MUST be connected to GND. |
| GNDIO = Dedicated ground pin, which MUST be connected to GND. |
| RESERVED = Unused I/O pin, which MUST be left unconnected. |

CHIP "filter" ASSIGNED TO AN EPF10K10QC208-3

<table>
<thead>
<tr>
<th>PIN</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCK</td>
<td>1</td>
</tr>
<tr>
<td>CONF_DONE</td>
<td>2</td>
</tr>
<tr>
<td>nCEO</td>
<td>3</td>
</tr>
<tr>
<td>TDO</td>
<td>4</td>
</tr>
<tr>
<td>VCCIO</td>
<td>5</td>
</tr>
<tr>
<td>VCCINT</td>
<td>6</td>
</tr>
<tr>
<td>N.C.</td>
<td>7</td>
</tr>
<tr>
<td>N.C.</td>
<td>8</td>
</tr>
<tr>
<td>N.C.</td>
<td>9</td>
</tr>
<tr>
<td>x7</td>
<td>10</td>
</tr>
</tbody>
</table>
Floorplan Editor

◆ Graphical user interface for viewing/creating resource assignments
  • Pins
  • Logic cells
  • Cliques
  • Logic options

◆ Drag-and-drop capability for assigning pins/logic cells

◆ Graphical view of current assignments as well as last compilation results

◆ LAB view or external chip view
Floorplan Views

Assignments to Back-Annotate Project

- Chip, Logic Cell, Pin & Device
- Chip, LAB, Pin & Device
- Chip, Row, Pin & Device
- Chip, Pin & Device
- Chip
- Device

Fitter Results

<design>.fit
<design>.acf

User Assignments

Current Assignment Floorplan (Editable View)

Last Compilation Floorplan (Read-Only)
Floorplan Editor (Read Only)

◆ Last Compilation Floorplan Full Screen LAB View with Report File Equation Viewer
Floorplan Editor (Read Only)

- Last Compilation Floorplan Device View

- Color Legend definition
- Pin name
- Pin number
Floorplan Editor (Editable)

- Current Assignment view has drag and drop capability (Note: Auto Device can not be used)

Click on Node, hold left mouse, drag to location
Floorplan Editor (Editable)
Project Compilation
Recommendations

- Use assignments after design analysis to improve fitting or performance
- Use the Report File to find specific information on the design
- Use the Floorplan Editor to see results of Assignments
Report File Equation Viewer
Routing Statistics

<table>
<thead>
<tr>
<th>Information on Selected Node/Pin/LAB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name is:</strong> LC1_B7 <strong>LAB is:</strong> B7</td>
<td></td>
</tr>
<tr>
<td><strong>Row is:</strong> B <strong>Column is:</strong> 7</td>
<td></td>
</tr>
<tr>
<td><strong>Logic Cell Fan-In:</strong> 5 <strong>Logic Cell Carry-Out:</strong> Yes</td>
<td></td>
</tr>
<tr>
<td><strong>Logic Cell Fan-Out:</strong> 9 <strong>Logic Cell Cascade-Out:</strong> No</td>
<td></td>
</tr>
</tbody>
</table>

- **Cell Total Shared Expanders Used:**
  - Embedded Cell Depth (Bits):
  - LAB Total Shared Expanders Used:
  - LAB External Interconnect Used: 6/24 (25%)
  - Column Interconnect Channels Used: 3/16 (18%)
  - Full Row Interconnect Channels Used: 128/168 (76%)
  - Half Row Interconnect Channels Used:
  - Logic Cell Inputs Borrowed from LC1:

**Most Congested Areas in Current Chip**

- **Most Congested LAB (or EAB) is:** A20
  - LAB (or EAB) External Interconnect Used: 15/24 (62%)
- **Most Congested Row is:** B
  - Row Interconnect Channels Used: 128/168 (76%)
- **Most Congested Column is:** 6
  - Column Interconnect Channels Used: 10/16 (62%)
Floorplan Editor Utilities Menu

◆ To find text, node, ...
  • “Find Text” command: to search the current chip for the first occurrence of the specified text
  • “Find Node” command: to find one or more nodes or other logic function(s) in the design file or in the floorplan

◆ To help running timing analysis
  • You can specify source and destination nodes in the floorplan to run timing analysis
Assigning Logic to Physical Resources

◆ **Use Floorplan Editor to assign logic to physical resources**
  
  • You can assign logic to a device, to any row or column within a device, or to a specific LAB, pin, logic cell, or I/O cell in Floorplan Editor very easily
  
  • To toggle between current assignment & last compilation floorplan
    
    Menu: **Layout -> Current Assignments Floorplan**
    
    Menu: **Layout -> Last Compilation Floorplan**

◆ **Back-annotate the floorplan for subsequent compilation**
  
  • If necessary, you can back-annotate the floorplan to ACF(Assignment & Configuration File) and it is useful for retaining the current resource and device assignments for future compilations
    
    Menu: **Assign -> Back-Annnotate Project...**
Current Pin Assignment Floorplan
Current LAB Assignment
Floorplan

Anywhere on Device
Anywhere on this Column
Anywhere on this Row
### Project Compilation Summary

**Design Files**
- .wdf
- .v
- .vhd
- .sch
- .edf

**Simulation/Timing Files**
- .snf

**Programming Files**
- .pof

**Report Files**
- .rpt

**EDA Files**
- .sdo
- .edo
- .vo
- .vho

**MAX+PLUS II Compiler**
- Compiler Netlist Extractor (includes all netlist readers)
- Database Builder
- Logic Synthesizer
- Functional, Timing, or Linked SNF Extractor
- Partitioner
- Fitter
- EDIF, VHDL & Verilog Netlist Writers
- Design Doctor
- Assembler

**3rd Party EDA Files**
Project Verification

- Project Verification Methodology
- MAX+PLUS II Simulator
- Functional Simulation
- Timing Simulation
- Timing Analysis
Project Verification Methodology

MAX+PLUS II Timing Analyzer

MAX+PLUS II Simulator

MAX+PLUS II Waveform Editor

Files:
- .snf
- .mif
- .hex
- .cm
- .tao
- .hst
- .sif
- .log
- .tbl
- .vec
- .scf
- .tbl
MAX+PLUS II Simulator

- MAX+PLUS II Compiler
- MAX+PLUS II Waveform Editor
- MAX+PLUS II Text Editor
- .snf
- .scf
- .vec
- .scf
- MAX+PLUS II Waveform Editor
MAX+PLUS II Simulation

◆ Create Simulation Stimulus
  • Waveform
  • Vector

◆ Run Functional Simulation
  • Fast compilation
  • Logical model only, no logic synthesis
  • All nodes are retained and can be simulated
  • Outputs are updated without delay

◆ Run Timing Simulation
  • Slower compilation
  • Timing model: logical & delay model
  • Nodes may be synthesized away
  • Outputs are updated after delay
Simulation Waveform

◆ **Stimulus Waveform**
  - Waveform Editor File (.scf)
  - Control
    - Clock: Use built-in clock generator
    - Others: Hand drawn with overwrite/copy/paste/repeat
  - Data
    - Counting patterns: Use built-in binary or gray code generator
    - Others: Enter with overwrite/copy/paste/repeat

◆ **Reference Compare waveform**
  - Waveform Editor File (.scf)
  - Draw or save previous simulation result as reference waveform
  - Use with Compare after new simulation run to verify output
Create Waveform Simulation Stimulus

- Open **Waveform Editor**
- Select **Enter Nodes from SNF...** from Node menu
- Enter Nodes into **Selected Nodes & Groups** field
Simulator Environment
**Grid Control**

◆ **Snap to Grid**
  - On: waveforms drawn increments of grid size
  - Off: waveforms can be drawn to any size
Draw Stimulus Waveform

- Highlight portion of waveform to change
- Overwrite with desired value (Group value or single bit)
Create Clock Waveform

- Snap to Grid On: Clock Period is twice the grid size
- Snap to Grid Off: Clock Period can be any value

Highlight waveform
Clock shortcut
Specify clock period
Create Counting Pattern

- Make sure your counting frequency matches your clock frequency
Grouping Signals and Set Radix

- Highlight waveforms to be grouped
  - MSB must be the top waveform
- Enter Group Name and set Radix
Save the Waveform Stimulus File

- Save the waveform stimulus file with .scf extension
- MAX+PLUS II will use Project name as default file name
Create Vector Simulation Stimulus

- Open **Text Editor**
- Type in vector stimulus
  - **Clock**
    - % units default to ns%
    - START 0;
    - STOP 1000;
    - INTERVAL 100;
    - INPUTS CLOCK;
    - PATTERN
    - 0 1; % CLOCK ticks every 100 ns%
  - **Pattern**
    - INPUTS A B;
    - PATTERN
    - 0> 0 0
    - 220> 1 0
    - 320> 1 1
    - 570> 0 1
    - 720> 1 1;
  - **Output**
    - OUTPUTS Y1 Y0;
    - PATTERN % check output at every Clock pulse%
    - = X X
    - = 0 0
    - = 0 1
    - = 1 0
    - = 1 1;
Save the Vector Stimulus File

- Save the vector stimulus file with .vec extension
  - You must change the .vec extension since MAX+PLUS II defaults to .tdf extension for text files

Change the extension to .vec
Select Simulation Stimulus File

- Defaults to .scf file
- For vector input stimulus, set Vector Files Input to .vec file
Specify Length of Simulation

- Specify maximum length of simulation time with End Time
Run Functional Simulation

- Click on Start then Open SCF to see result

- Click on Start Button

- Open SCF file

- Output change on clock edge
MAX+PLUS II Functional Simulation

◆ Use to verify operation of design

◆ Advantage over Timing Simulation
  • Fast compilation
  • All nodes are retained and can be simulated
  • Outputs are updated without delay
    – Most of the time, this makes figuring out cause and effect much easier

◆ Disadvantages
  • Logical model only, no logic synthesis
  • No delays in simulation
    – Oscillations, glitches and other timing related errors do not show up
Run Timing Simulation

- Click on Start then Open SCF to see result

Output change after timing delay
MAX+PLUS II Timing Simulation

- Used to debug timing related errors

**Advantages over Functional Simulation**
- Simulation of full synthesis result
- Outputs change after timing delay
  - Detection of oscillations, glitches and other timing related errors are possible

**Disadvantages**
- Longer compilation time
- Combinatorial logic nodes cannot be simulated
  - Node may be transformed or removed
- Only “Hard” nodes can be simulated
- Timing delays make debugging more difficult because cause and effect relationships are harder to locate
Comparing Different Simulations

Compare Two Simulation Files

- Open first channel file
- Choose Compare under File menu
- Select the name of the second channel file with the Compare dialog box
- Waveforms from the first channel file are drawn in black. Waveforms from the second channel file are drawn in red on top of the black waveforms. Deviations of second channel file can easily be spotted.
Project Simulation Recommendations

- Use built-in clock generator to create clock
- Use built-in count generator to create test pattern
- Use Functional Simulation to verify proper operation
- Use Timing Simulation to examine signal delay effects
- Use Compare function to verify output
- Use the dynamic link (Find Node in Design File) to go to source file to make any necessary corrections
Simulation Input & Output Files

◆ Specify simulation input and output files

You can specify SCF or VEC file as the source of simulation input vectors

Menu: File -> Inputs/Outputs...
- VEC file will be converted into SCF file by Simulator
- You can specify a history(*.hst) or log(*.log) file to record simulation commands and outputs

During and after simulation, the simulation results are written to the SCF file, you can create another ASCII-format table file

Menu: File -> Create Table File...
- TBL file format is a subset of VEC file format
- A TBL file can be specified as a vector input file for another simulation
Memory Initialization

◆ Give memory initialization values for functional simulation

  • To generate memory initialization values in Simulator
    
    Menu: Initialize -> Initialize Memory...

  • You can save the data in the Initialize Memory dialog box to a Hexadecimal File (*.hex) or Memory Initialization File (*.mif) for future use
    
    Menu: Initialize -> Initialize Memory... -> Export File...
    
    – An MIF is used as an input file for memory initialization in the Compiler and Simulator. You can also use a Hexadecimal File (.hex) to provide memory initialization data.

  • You can load the memory initialization data for a memory block that is saved in a HEX or MIF file
    
    Menu: Initialize -> Initialize Memory... -> Import File...
Initialize Memory Window
Memory Initialization File Formats

WIDTH = 16;
DEPTH = 256;
ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;
CONTENT BEGIN
  0 : 0000;
  1 : 0000;
  2 : 0000;
  3 : 0000;
  4 : 0000;
  5 : 0000;
  6 : 0000;
  7 : ffff;
  8 : ffff;
  9 : ffff;
 a : ffff;
b : ffff;
c : ffff;
d : ffff;
e : ffff;
f : ffff;
ff : 0000;
END;

MIF file example

HEX file example

020000000000fe
020001000000fd
020002000000fc
020003000000fb
020004000000fa
020005000000f9
020006000000f8
020007000000f7
020008000000f6
020009000000f5
02000a000000f4
02000b000000f3
02000c000000f2
02000d000000f1
02000e000000ff
02000f000000ff
00000001ff
...
MIF File Format

◆ To edit a MIF file...

- MIF file is an ASCII text file that specifies the initial content of a memory block
  - You can create an MIF in the MAX+PLUS II Text Editor or any ASCII text editor
  - You can also very easily generate an MIF by exporting data from the Simulator's Initialize Memory dialog box

- Example:

```plaintext
DEPTH = 32; % Memory depth and width are required %
WIDTH = 14; % Enter a decimal number %
ADDRESS_RADIX = HEX; % Address and value radices are optional %
DATA_RADIX = HEX; % Enter BIN, DEC, OCT or HEX(default) %

-- Specify values for addresses, which can be single address or range
CONTENT
BEGIN
[0..F] : 3FFF; % Range--Every address from 0 to F = 3FFF %
6 : F; % Single address--Address 6 = F %
8 : FE 5; % Range starting from specific address %
```
Remember: MAX+PLUS II Compiler uses MIF or HEX file(s) to create ROM or RAM initialization circuit in FLEX 10K EAB

- Specify the LPM_FILE parameter to a MIF or HEX file for each RAM and ROM block
  - Memory initialization file is optional for RAM
  - Using MIF files is recommended because its file format is simple

If the memory initial file does not exist when MAX+PLUS II Compiler is generating functional SNF file, you must initialize the memory by using Initialize Memory command before starting the functional simulation

- MAX+PLUS II Compiler reports an warning when it can’t read the memory initialization file when processing Functional SNF Extractor
- However, the memory initialization file must exist when MAX+PLUS II processes Timing SNF Extractor
If you do not have MIF or HEX files, do the following:

- Run MAX+PLUS II Compiler to generate a functional SNF file first
- Then invoke MAX+PLUS II Simulator, use Memory Initialization command to create memory content for each ROM or RAM block
- Export memory content to a MIF or HEX file
  - And now, you can perform functional simulation for your project
- Invoke MAX+PLUS II Compiler again, turn on “Timing SNF Extractor” and start complete compilation for FLEX 10K devices
Node/Group Initialization

◆ Specify initial logic levels for nodes/groups
  
  • You can change the initial logic levels of registered nodes/groups in the SNF file for the project before you begin simulation

  Menu: Initialize -> Initialize Nodes/Groups...
  
  – You can also specify an initial state name for a group that represents a state machine.
  
  • By default, all register outputs are initialized to 0 and pin inputs are initialized to the first logic level provided in the current SCF
Saving Initialization Values

◆ Save the initialization values to SIF file
  • You can save current initialized node and group logic levels and memory values to a Simulation Initialization File (*.sif)
    Menu: Initialize -> Save Initialization As...
  • To retrieve initialized node, group, and memory values stored in a SIF file
    Menu: Initialize -> Restore Initialization...
  • To reset initial node, group, and memory values to the values stored in the SNF file
    Menu: Initialize -> Reset to Initial SNF Values
      – All register outputs are initialized to 0, and pin inputs are initialized to the first logic level provided in the current SCF file
Creating Breakpoints

◆ Specify simulation breakpoints
  • You can create one or more breakpoints, each of which consists of one or more node value, group value, and time conditions
  Menu: Options -> Breakpoints...
  • Specify breakpoint conditions
    – .TIME variable in Node/Group list represents the simulation time
    – Operator:=, !, >, <, >=, <=, >-> (transition)
    – A breakpoint can consist one or more conditions and must be given a unique name
Monitoring Options

◆ **Setup time & hold time**
  - You can instruct the Simulator to monitor all simulated nodes and groups for setup time and hold time violations
    - It’s not available in functional simulation mode
    - In timing simulation linked simulation mode, setup and hold time violations are determined by the architecture of the device(s) being simulated

◆ **Glitch**
  - You can instruct the Simulator to monitor the logic levels of all simulated nodes and groups for glitches or spikes, i.e., two or more logic level changes that occur within a period less than or equal to the specified time
    - It’s not available in functional simulation mode

◆ **Oscillation**
  - The Simulator can monitor all simulated nodes and groups for logic levels that do not stabilize within the specified time period after the most recent input vector has been applied
    - In functional simulation mode, oscillation option is always on and check only for nil-period oscillation
Project Simulation Summary

◆ Two types of simulation
  • Functional simulation
    – No logic synthesis
    – No delay model
    – All nodes can be simulated
  • Timing simulation
    – Logic synthesis
    – Delay model
    – Only hard nodes can be simulated

◆ Two types of stimulus file
  • Waveform
  • Vector

◆ Simulation result is stored in .scf file
Project Timing Analysis

◆ **Timing Analyzer is a static timing analyzer**

◆ **Three forms of timing analysis**
  - *Registered Performance* calculates fastest possible internal clock frequency
  - *Delay Matrix* calculates combinatorial delays
  - *Setup/Hold Matrix* calculates setup & hold times for device flip-flops

◆ **Source of delay path can be located in**
  - Design file
  - Floorplan Editor
Specify source/destination nodes for timing analysis

- The Timing Analyzer provides default timing tagging for source and destination nodes for each analysis mode.

Menu: Node -> Timing Analysis Source...
Menu: Node -> Timing Analysis Destination...

- Besides, you can specify timing analysis source & destination nodes in the Graphic, Waveform or Floorplan Editor (under Utilities Menu).
Registered Performance Analysis

- Calculates maximum internal register frequency
- Used to determine if design meets clock specification

Clock period = tco + delay + tsu + tskew

**Note:** tskew is added to the clock period if destination clock edge is earlier than source clock edge
Run Registered Performance Analysis

- Click on Start
- Source/Destination, Clock period and Frequency of the longest path are displayed
- Click on List Paths to trace delay path
Tracing Delay Path In Floorplan Editor

- Highlight Path of interest
- Check Locate in Floorplan Editor
- Click on Locate All
- Click on show path to display path
Application of Registered Performance

- Use Registered Performance Analysis to see if design meets clock frequency requirement

- What to do if frequency is less than desired
  - Use List Path to display the worst case delays
  - Use Floorplan Editor to view the entire path
    - Are Logic Cells and pins scattered among different rows?
    - Can the Logic Cells benefit from carry/cascade chains (FLEX) or parallel expanders (MAX)?
  - Use Assignments (Clique, Logic Options, etc…) on the critical path to improve performance
  - If still less than desired, consider pipelining technique or different design implementations where appropriate
Delay Matrix Analysis

- Calculates combinatorial logic delays
- Typically used to evaluate input pin to output pin delay
- Internal point to point delay analysis is possible by setting node source and destination for analysis

Combination Logic
Delay Matrix Source and Destination

- Set Source and Destination to be analyzed
Useful Analysis Options

- **Time Restrictions**
  - Show All Path
  - Show Only Longest Paths
  - Show Only Shortest Paths

- **Cell Width**
  - Control matrix display

- **Cut Off I/O Pin Feedback**
  - See next page

- **Cut Off Clear & Preset Paths**
  - No clear or preset delay analysis

- **List Only Longest Path**
  - List Path lists only longest path between two points
Cut Off I/O Pin Feedback

- Used to break bi-directional pin from the analysis
- When on, paths A and B true C false
- When off, path A, B and C are true
Run Delay Matrix Analysis

- Select Delay Matrix Analysis and click on Start button
- Matrix shows all paths, longest path, or shortest path depending on Time Restrictions option selected
- Use List Paths to analyze the path of delays
Setup/Hold Matrix Analysis

Setup/Hold Matrix calculates setup & hold times for device flip-flops

- **Setup**
  - $t_{setup} = t_{data} - t_{clock} + t_{setup}$

- **Hold**
  - $t_{hold} = t_{clock} - t_{data} + t_{hold}$
Run Setup/Hold Matrix Analysis

- Click on Start button
- Setup/Hold times are displayed with respect to the clocks
Saving Timing Analysis Results

◆ Save the current Timing Analyzer results to a TAO File
  - Timing Analyzer can save the information in the current timing analysis display to an ASCII-format Timing Analyzer Output file (*.tao)
  
  **Menu: File -> Save Analysis As...**

<table>
<thead>
<tr>
<th>Destination</th>
<th>y3</th>
<th>y4</th>
<th>y5</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclr</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>clk</td>
<td>10.8ns</td>
<td>12.7ns</td>
<td>11.7ns</td>
</tr>
<tr>
<td>xin1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin2</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin3</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin4</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin5</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin6</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin7</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>xin8</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
Listing & Locating Delay Paths

❖ To trace delay paths or clock paths in the design file
  • After you run a timing analysis, you can list selected signal paths and locate them in the original design file(s) for the project
  • Select the matrix cell or clock, click **List Paths**
  • Select one of the delay paths shown in Message Processor, and click **Locate** to trace the path in the source file(s)
Listing & Locating Paths
Recommended Verification Flow

- **Functional simulation**
  - Perform functional simulation to verify the design functionality

- **Timing Analysis**
  - Perform static timing analysis to check overall performance
  - Find the delay paths

- **Timing simulation**
  - Perform timing simulation to verify real-world design timing & functionality

- **On-board test**
  - Program FPGA/CPLD device(s) and test the function & timing in system
Timing Analysis Recommendations

- Use Timing Analyzer to locate performance bottleneck
- Use Registered Performance Analysis to determine internal clock frequency performance of the design
- Use Show Only Longest Path Time Restrictions in Delay Matrix to get the longest delay time from input pin to output pin
- Use List Path and Locate in Floorplan Editor to view worst case paths
- Use List Path and Locate to trace through path in design file
- Use assignments and recompile to fine-tune performance
Project Timing Analysis Summary

◆ **Timing Analyzer is a static timing analyzer**

◆ **Three modes of Timing Analysis**
  - Registered Performance
  - Delay Matrix
  - Setup/Hold Matrix

◆ **Provides ability to trace path through Floorplan Editor or design file**
Device Programming

- Programming Methods
- Altera Configuration EPROM Family
- Altera Programming Hardware
  - PL-ASAP2 Stand-Alone Programmer
  - BitBlaster Download Cable
  - ByteBlaster Download Cable
- FLEX Device Configuration Schemes
- MAX+PLUS II Programmer
Altera Provide Method

- **Altera provide different methods for**
  - Program Device
    - MAX family
  - Configure Device
    - FLEX family
MAX Device

◆ Use Altera Stand Alone Programmer (ASAP2)

◆ Through JTAG port with ByteBlaster
  • JTAG for Single Device (MAX or FLEX)
  • JTAG Chain for Multiple Device (MAX & FLEX)
  • JAM for Single/Multiple Device (MAX & FLEX)

◆ 3rd Programmer
  • Data I/O
  • BP MicroSystem
    – http://www.bpmicro.com
FLEX Device

◆ Through JTAG port with ByteBlaster
  • JTAG for Single Device (FLEX or MAX)
  • JTAG Chain for Multiple Device (FLEX & MAX)

◆ Through PS port with ByteBlaster
  • FLEX for Single Device
  • FLEX Chain for Multiple Device

◆ Serial PROM
  • EPC1 (1Mbits, good for 6K/8K/10K)
  • EPC1441 (441Kbits, good for 6K/8K/10K10, 10K20, 10K30)
  • EPC1213 (213Kbits, only for 8K)
  • EPC1064 (64Kbits, only for 8K)
Configuration File Sizes

◆ FLEX device configuration file sizes

• Each FLEX device has a different size requirement for its configuration data, based on the number of SRAM cells in the device
• The following table summarizes the configuration file size required for each FLEX device
  - To calculate the amount of data storage space for multi-device configurations, simply add the file sizes for each FLEX device in the design

<table>
<thead>
<tr>
<th>Device</th>
<th>Data Size(bits)</th>
<th>Device</th>
<th>Data Size(bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPF8282A/V</td>
<td>40,000</td>
<td>EPF10K10</td>
<td>115,000</td>
</tr>
<tr>
<td>EPF8452A</td>
<td>64,000</td>
<td>EPF10K20</td>
<td>225,000</td>
</tr>
<tr>
<td>EPF8636A</td>
<td>96,000</td>
<td>EPF10K30</td>
<td>368,000</td>
</tr>
<tr>
<td>EPF8820A</td>
<td>128,000</td>
<td>EPF10K40</td>
<td>488,000</td>
</tr>
<tr>
<td>EPF81188A</td>
<td>192,000</td>
<td>EPF10K50</td>
<td>609,000</td>
</tr>
<tr>
<td>EPF81500A</td>
<td>250,000</td>
<td>EPF10K70</td>
<td>881,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EPF10K100</td>
<td>1,172,000</td>
</tr>
</tbody>
</table>
Altera Configuration EPROM Family

◆ Altera’s serial configuration EPROMs for FLEX devices
  • Simple, easy-to-use 4-pin interface to FLEX devices
  • Available in OTP packages: 8-pin PDIP, 20-pin PLCC and 32-pin TQFP
  • Family member
    – EPC1064: 65,536 bit device with 5.0-V operation
    – EPC1064V: 65,536 bit device with 3.3-V operation
    – EPC1213: 212,942 bit device with 5.0-V operation
    – EPC1: 1,046,496 bit device with 5.0-V or 3.3-V operation
Configuration EPROM Block Diagram - (1)

◆ EPC1064, EPC1213, or EPC1 in FLEX 8000A mode
Configuration EPROM
Block Diagram - (2)

◆ EPC1 in FLEX 10K mode
Altera Programming Hardware

Hardware to program and configure Altera devices

- For MAX 7000/E/S, MAX 9000 and Altera configuration EPROM(EPC-series) devices
  - Altera stand-alone programmer: PL-ASAP2 (PC platform)
  - 3rd-party universal programmer (PC platform)
- For MAX 7000S and MAX 9000 ISP, FLEX devices downloading
  - Altera BitBlaster download cable (RS-232 port)
  - Altera ByteBlaster download cable (parallel port of PC)

- Of course, you can use another 3rd-party universal programmer or download cable to program or configure Altera devices. In this chapter, we discuss Altera programming hardware only.
Altera Stand-Alone Programmer

**PL-ASAP2: Altera stand-alone hardware programmer**

- The Altera stand-alone programmer, PL-ASAP2, together with the appropriate programming adapters, supports device configuration and programming for Altera devices
  - All MAX devices
  - Altera serial configuration EPROM: EPC1/V, EPC1064/V, EPC1213
- PL-ASAP2 includes an LP6 Logic Programmer card, an MPU and software
  - LP6 card generates programming waveforms and voltages for the MPU
  - MPU (Master Programming Unit) connects to LP6 card via a 25-pin ribbon cable and is used together with an appropriate adapter to program Altera devices
  - Optional FLEX download cable for configuring FLEX devices
Installing LP6 Card, MPU & Adapter

Connect the 25-pin flat ribbon cable to the LP6 card.
BitBlaster Download Cable

◆ Altera BitBlaster serial download cable
  • BitBlaster serial download cable allows PC and workstation users to
    – Program MAX 9000, MAX 7000S in-system via a standard RS-232 port
    – Configure FLEX devices in circuit via a standard RS-232 port
  • BitBlaster provides two download modes
    – Passive Serial (PS) mode: used for configuring all FLEX devices
    – JTAG mode: industry-standard JTAG implementation for programming or configuring FLEX 10K, MAX 9000, and MAX 7000S devices
  • BitBlaster status lights:
    – POWER: indicates a connection to the target system’s power supply
    – DONE: indicates device configuration or programming is complete
    – BUSY: indicates device configuration or programming is in process
    – ERROR: indicates error detection during configuration or programming
Installing BitBlaster

- Status lights
- Receptacle for pin 1
- 25-pin female port
- Baud rate dip switches
- to 10-pin male header on circuit board
ByteBlaster Download Cable

◆ Altera ByteBlaster parallel port download cable
  • ByteBlaster serial download cable allows PC users to
    – Program MAX 9000, MAX 7000S in-system via a standard parallel port
    – Configure FLEX devices in circuit via a standard parallel port
  • ByteBlaster provides two download modes
    – Passive Serial(PS) mode: used for configuring all FLEX devices
    – JTAG mode: industry-standard JTAG implementation for programming or configuring FLEX 10K, MAX 9000, and MAX 7000S devices
  • ByteBlaster download cable provides a fast and low-cost method for ISP and FLEX device configuration
  • ByteBlaster download cable uses identical 10-pin circuit board connector as the BitBlaster serial download cable
Installing ByteBlaster
BitBlaster & ByteBlaster Plug Connections

Dimensions are shown in inches.

<table>
<thead>
<tr>
<th>Pin</th>
<th>PS Mode</th>
<th>JTAG Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DCLK</td>
<td>TCK</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>3</td>
<td>CONF_DONE</td>
<td>TDO</td>
</tr>
<tr>
<td>4</td>
<td>VCC</td>
<td>VCC</td>
</tr>
<tr>
<td>5</td>
<td>nCONFIG</td>
<td>TMS</td>
</tr>
<tr>
<td>6</td>
<td>N.C.</td>
<td>N.C.</td>
</tr>
<tr>
<td>7</td>
<td>nSTATUS</td>
<td>N.C.</td>
</tr>
<tr>
<td>8</td>
<td>N.C.</td>
<td>N.C.</td>
</tr>
<tr>
<td>9</td>
<td>DATA0</td>
<td>TDI</td>
</tr>
<tr>
<td>10</td>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>
FLEX Device Configuration Schemes

- Passive Serial (PS) configuration with the download cable
  - Single-device configuration
  - Multiple-devices configuration

- JTAG configuration with the download cable
  - Available for FLEX 10K and ISP devices only

- Serial configuration EPROM configuration
  - FLEX 8000A Active Serial (AS) configuration with serial configuration EPROM
  - FLEX 10K Passive Serial (PS) configuration with EPC1 configuration EPROM

- Parallel EPROM configuration
  - FLEX 8000A Active Parallel Up (APU) or Active Parallel Down (APD) configuration
  - Not available for FLEX 10K devices
FLEX 8000A Single-Device PS Configuration with the Download Cable
FLEX 8000A Multiple-Device PS Configuration with the Download Cable
FLEX 10K Single-Device PS Configuration with the Download Cable

PS. nCE pin of FLEX 10K device must connect to GND.
FLEX 10K Multiple-Device PS Configuration with the Download Cable

PS. nCE pin of the lead FLEX 10K device must connect to GND.
JTAG Configuration with the Download Cable
FLEX 8000A Configuration EPROM Configuration (AS Mode)

AS configuration with automatic reconfiguration on error (nStatus pin is connected to OE pin of the configuration EPROM and when “Auto-Restart Configuration on Frame Error” option bit is turned on)
FLEX 8000A Multiple Configuration EPROMs

Configuration
FLEX 8000A Multi-Device Configuration
EPROM Configuration
FLEX 10K Configuration EPROM Configuration (PS Mode)

Altera-supplied serial EPC1 configuration EPROM
(EPC1: 1-Mbit EPROM)
FLEX 10K Multi-Device Configuration EPROM
FLEX 8000A Multi-Device APU & APD Configuration
To program or configure Altera devices

- After the MAX+PLUS II Compiler has processed a project, it generates one or more programming files, which the Programmer uses to program or configure one or more devices.

- The MAX+PLUS II Programmer allows you to program, verify, examine, blank-check, configure, and test Altera all MAX and FLEX devices and configuration EPROM.

- With the Programmer and programming hardware—the Altera MPU, add-on cards, programming adapters, the FLEX download cable, the BitBlaster, or the ByteBlaster—you can easily create a working device in minutes.
Device Programming Methodology
Programmer Operations

◆ 6 operations

• **Program**: programs data onto a blank MAX device or configuration EPROM
• **Verify**: verifies contents of a device against current programming data
• **Examine**: examines a device & stores the data in a temporary buffer
• **Blank-Check**: examines a device to ensure it is blank
• **Test**: functionally tests a programmed device
• **Configure**: downloads configuration data into the SRAM of one or more FLEX devices
Starting Programming

- Program or configure the Altera device
  - Setup the hardware
    Menu: Options -> Hardware Setup... -> Auto-Setup
  - Specify the programming file
    Menu: File -> Select Programming File...
  - Program or configure the device: just click on the Program or Configure button
Functional Test on Device

◆ Functionally test the Altera device
  • You can use an SCF or VEC file, or test vectors stored in the current programming file, to functionally test actual device outputs against simulation outputs
    – Functional testing is not available for SRAM-based FLEX devices
    – You can only test devices for single-device projects
    – You also cannot test projects that contain bidirectional buses
  • After the device is programmed, select simulation input file
    Menu: File -> Inputs/Outputs
    – You can specify an output Programmer Log File (*.plf) to record the Programmer's activities
  • Test the device: click on Test button
Converting or Combining Programming Files

To convert or combine programming files

- You can combine and convert one or more SRAM Object Files (*.sof) into one of the following file formats, which support different FLEX device configuration schemes
  - Programmer Object File (*.pof)
  - Raw Binary File (*.rbf)
  - Tabular Text File (*.ttf)
  - Serial Bitstream File (*.sbf)
  - Hexadecimal (Intel-format) File (*.hex)

Menu: File -> Combine Programming Files
Configuring Multiple FLEX Devices

◆ Configure multiple FLEX device with the download cable
  • You can configure multiple FLEX devices in a FLEX chain with the download cable
    – By typing a command at a DOS command prompt to download configuration data from an SBF file through the BitBlaster
      DOS Prompt: copy <design>.sbf COM1: (or COM2:)
        • The SBF file can be created by using “Combine Programming File” command (under File Menu)
    – By creating “multi-device FLEX chain” (under FLEX Menu) and using the Programmer to download configuration data from SOFs through the BitBlaster, ByteBlaster, or FLEX download cable
      • Multi-device FLEX chain: a series of FLEX devices through which configuration data is passed from device to device using the sequential Passive Serial configuration scheme

FLEX Menu
Creating Multi-Device FLEX Chain

To configure multiple FLEX devices in a FLEX chain

- You can specify the order and names of SOFs for multiple FLEX devices in a chain
  Menu: FLEX -> Multi-Device FLEX Chain Setup...
- You can save the FLEX chain settings to an Flex Chain File(*.fcf) or restore the settings from a existing FCF file
  Menu: FLEX -> Save FCF...
  Menu: FLEX -> Restore FCF...
- To turn on or off multi-device FLEX chain configuration mode
  Menu: FLEX -> Multi-Device FLEX Chain...
- Click Configure button on Programmer to start configuration
Programming Multiple JTAG Devices

◆ Program or configure multiple JTAG devices with the download cable

• You can program or configure one or more MAX 9000, MAX 7000S, FLEX 10K devices, and other devices that support JTAG programming in a JTAG chain using the BitBlaster or ByteBlaster
  – The JTAG chain can contain any combination of Altera and non-Altera devices that comply with the IEEE 1149.1 JTAG specification, including some FLEX 8000 devices
  – By creating “multi-device JTAG chain” (under JTAG Menu) and using the Programmer to download configuration data from SOFs through the BitBlaster or ByteBlaster cable
Creating Multi-Device JTAG Chain

To program multiple devices in a JTAG chain

- You can select the names and sequence of devices in the JTAG chain, and optional associated programming files
  
  **Menu:** JTAG -> Multi-Device JTAG Chain Setup...

- You can save the JTAG chain settings to an JTAG Chain File (*.jcf) or restore the settings from a JCF file
  
  **Menu:** JTAG -> Save JCF...
  **Menu:** JTAG -> Restore JCF...

- To turn on or off multi-device JTAG chain programming mode
  
  **Menu:** JTAG -> Multi-Device JTAG Chain

- Click **Configure** or **Program** button on Programmer to start programming

![Image of Multi-Device JTAG Chain Setup window]
Details about Device Programming

 Panasonic

◆ Please refer to Altera document for details
  • Altera Data Book
  • Altera Data Sheet
    – dsconf_06.pdf: Configuration EPROMs for FLEX Devices
    – dsbit03.pdf: BitBlaster Serial Download Cable
    – dsbyte01.pdf: ByteBlaster Parallel Port Download Cable
  • Altera Application Note & Application Brief
    – an033_03.pdf: Configuring FLEX 8000 Devices
    – an038_03.pdf: Configuring FLEX 8000 Devices
    – an059_01.pdf: Configuring FLEX 10K Devices
    – ab141_01.pdf: In-System Programmability in MAX 9000 Devices
    – ab145_01.pdf: Designing for In-System Programmability in MAX 7000S Devices
    – an039_03.pdf: JTAG Boundary Scan Testing in Altera Devices
Getting Help

◆ CIC technical support: 周育德
  • Phone: (03)5773693 ext. 148
  • Email: steven@cic.edu.tw
  • News: nsc.cic
  • WWW: http://www.cic.edu.tw/chip_design/design_intr/altera/

◆ To buy Altera chips, hardware or demo boards:
  • Contact Galaxy Far East Corp. 茂綸公司楊樂麗小姐 (03)578-6766 ext. 220

◆ Altera technical support on Internet
  • WWW: http://www.altera.com
  • FTP: ftp://ftp.altera.com (however, the international access may be slow)
MAX+plus II Lab

Fibonacci Generator
Lab 1 - myor8 I

◆ 用基本的邏輯閘兜電路
  • File > New (Graphic Editor File - .gdf)
  • File > Save As… (myor8.gdf)
  • File > Project… > Set Project to Current File
  • 繪出下圖
Lab 1 - myor8 II

🔹 Save and Check
  • File > Project … > Save & Check (Cltr + K)

🔹 Generate Symbol
  • File > Create Default Symbol

🔹 View Symbol
  • File > Open (myor8.sym)
Lab 2 - Disbounce I

◆用Schematic設計防止彈跳電路
  • File > New (Graphic Editor File - .gdf)
  • File > Save As… (disbounce.gdf)
  • File > Project… > Set Project to Current File
  • 繪出下頁電路，請注意LPM_COUNTER的I/O與parameters的設定
  • File > Project … > Save & Check (Ctrl + K)
  • File > Create Default Symbol
Lab 3 - 7segment

◆ 用AHDL來設計七段顯示器解碼電路
  - File > New
    - (Text Editor File -.tdf)
  - File > Save as
    - (7segment.tdf)
  - 輸入AHDL
  - File > Project > Save & Check (Ctrl + K)
  - File > Create Default Symbol

TITLE "SEVEN SEGMENT BCD CODE DECODE";
SUBDESIGN 7SEGMENT
(
    DATAIN[3..0] :INPUT;
    DISPLAY[6..0] :OUTPUT;
)
BEGIN
TABLE
    DATAIN[] => DISPLAY[6..0];
    0 => B"1000000";
    1 => B"1111001";
    2 => B"0100100";
    3 => B"0110000";
    4 => B"0011001";
    5 => B"0010010";
    6 => B"0000010";
    7 => B"1111000";
    8 => B"0000000";
    9 => B"0010000";
   10 => B"0001000";
   11 => B"0000011";
   12 => B"1000110";
   13 => B"0100001";
   14 => B"0000110";
   15 => B"0001110";
END TABLE;
END;
Lab 4 - fib_top I

◆ File > New (Graphic Editor File - .gdf)
◆ 依據下列數圖完成電路
◆ File > Project > Save & Check (fib_top.v)
  • File > New (Graphic Editor File - .gdf)
  • File > Save As… (fib_top.gdf)
  • File > Project… > Set Project to Current File
  • 繪出下頁電路，請注意LPM的I/O與parameters的設定
  • File > Project … > Save & Check (Clnr + K)
  • File > Create Default Symbol
Lab 4 - fib_top III

利用 LPM_ROM 去產生七段顯示器的解碼電路

WIDTH = 7;
DEPTH = 16;
  • LPM_FILE - 7seg.mif

ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;

CONTENT BEGIN
  0: 1000000;
  1: 1111001;
  2: 0100100;
  3: 0110000;
  4: 0011001;
  5: 0010010;
  6: 0000010;
  7: 1011000;
  8: 0000000;
  9: 0010000;
  A: 0001000;
  B: 0000011;
  C: 1000110;
  D: 0100001;
  E: 0000110;
  F: 0001110;
END;
Lab 4 - fib_top IV

LPM_REG

LPM_DFF

data[]

enable

q[]

PREV[7..0]
Lab 4 - fib_top V

◆ LPM_ADD_SUB

LPM_DIRECTION="ADD"
LPM_PIPELINE=
LPM_REPRESENTATION=
LPM_WIDTH=8
ONE_INPUT_IS_CONSTANT=
Lab 5 - Compile I

◆ MAX+plus II > Compiler
◆ Assign > Device
  • EPF10K20RC204-4
◆ Assign > Pin/Location/Chip
Lab 5 - Compile II

◆ Assign > Global Project Logic Synthesis
  - Global Project Synthesis Style - FAST
  - Optimize - 10 (Speed)
◆ Press “Start”
Lab 6 - Check Report File

◆ Double click the Report File icon

◆ 觀察並記錄報告內容
  - Total dedicated input pins used:
  - Total I/O pins used:
  - Total logic cells used:
  - Total embedded cells used:
  - Total EABs used:
  - Memory Bits:
  - Average fan-in:
  - Total fan-in:
Lab 7 - Check Floorplan

- MAX+plus II > Floorplan Editor
- 與 report file 做比對
Lab 8 - Timing Simulation I

◆ Create a SCF file
  • File > New (Waveform Editor File - .scf)

◆ Insert nodes in SCF
  • Double click on “name”
  • Fill Node Name
    – TESTCLK, SEL, RESET, ENABLE, PREV[7..0], FIB[7..0], NEXT[7..0], DISPA[6..0], DISP[6..0]
Lab 8 - Timing Simulation II

◆ Change Grid Size
  • Options > Grid Size (20.0ns)

◆ Set End Time
  • File > End Time (5us)

◆ Draw Waveforms
  • TESTCLK (clock, period 40ns)
  • SEL (0)
  • RESET (0ns 0, 120ns 1, 1.0us 0, 1.12ns 1)
  • ENABLE (0ns 1, 1.6us 0, 2.0us 1)

◆ Save SCF File
  • File > Save as (fib_top.scf)

◆ Run Simulator
  • MAX+plus II > Simulator
  • Press “Start”
  • Press “Open SCF”
Lab 8 - Timing Simulation III
Lab 9 - Timing Analysis I

- MAX+plus II > Timing Analyzer
- Analysis > Delay Matrix
  - Press “Start”
Lab 9 - Timing Analysis II

- Analysis > Setup/Hold Matrix
  - Press “Start”

- Analysis > Registered Performance
  - Press “Start”
Lab 10 - Programmer I

◆連接硬體
  • 適當連接 Byteblaster, UP1, Adapter, Parallel Port

◆開啓 Programmer
  • MAX+PLUS II > Programmer

◆設定 Byteblaster
  • Options > Hardware Setup

◆設定 Multi-Device JTAG Chain
  • JTAG > Multi-Device JTAG Chain
  • 選定之後會在該選向前出現打勾的符號

◆設定 Multi-Device JTAG Chain Setup
  • JTAG > Multi-Device JTAG Chain Setup…
  • Device Name - 10K20
  • Programming File Name - fib_top.sof
    - 記得要按 Add
Lab 10 - Programmer II

- Click “Detect JTAG Chain Info”, if there are no problems, the following message will appear:

  ![Image of MAX+plusII Programmer window with JTAG chain information confirmed by hardware check message]

  Download Circuit
  - Click “Configure” in Programmer Window

  Operate Circuit
  - Click “Detect JTAG Chain Info”, if there are no problems, will show below message:

  ![Image of Multi-Device JTAG Chain Setup window]

  **Download Circuit**
  - Click “Configure” in Programmer Window

  **Operate Circuit**
Pin Assignment

SEL : PIN = 39;
CLK : INPUT_PIN = 91;
KEYIN : PIN = 28;
DISPB6 : OUTPUT_PIN = 24;
DISPB5 : OUTPUT_PIN = 23;
DISPB4 : OUTPUT_PIN = 21;
DISPB2 : OUTPUT_PIN = 19;
DISPB3 : OUTPUT_PIN = 20;
DISPB1 : OUTPUT_PIN = 18;
DISPB0 : OUTPUT_PIN = 17;
DISPA6 : OUTPUT_PIN = 13;
DISPA5 : OUTPUT_PIN = 12;
DISPA4 : OUTPUT_PIN = 11;
DISPA3 : OUTPUT_PIN = 9;
DISPA2 : OUTPUT_PIN = 8;
DISPA1 : OUTPUT_PIN = 7;
DISPA0 : OUTPUT_PIN = 6;
ENABLE : INPUT_PIN = 40;
RESET : INPUT_PIN = 41;