Introduction to PLD

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

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

Why programmable? Why reprogrammable?

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

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

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

LUT (Look-Up Table)
Programmable Register

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

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

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

Why filed-programmable?

- You can verify your designs at any time by configuring the FPGA/CPLD devices on board via the download cable or hardware programmer.
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 or CPLD

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
## FPGA/CPLD Benefits

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

- **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

**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>MAX 7000/E/S</td>
<td>EEPROM</td>
<td>SOP</td>
<td>600 ~ 5,000</td>
<td>EPM7032/V/S, 064/S, 096/S,</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,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>EPF10K70/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 - (1)

**MAX architecture**
- Parallel Logic Expanders (from other MCs)
- Product Term Select Matrix
- Shared Logic Expanders
- Programmable Register
- Register Bypass
- Clock Select
- Enable Select

**FLEX architecture**
- Look-Up Table (LUT)
- Carry Chain
- Cascade Chain
- Clear/Preset Logic
- Clock Select
- LE Out
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>
# 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 Architecture
FLEX 8000A Logic Element

![Diagram of FLEX 8000A Logic Element]

- **Carry Chain**: PRn, CLRn
- **Look-Up Table (LUT)**: DATA1, DATA2, DATA3, DATA4
- **Clear/Preset Logic**: LABCTRL1, LABCTRL2
- **Cascade Chain**
- **Cascade Out**: Clock Select
- **Data Out**: LE Out
Carry Chains

Carry-In
A1
B1

LUT
Carry Chain
Register
S1

LE1

A2
B2

LUT
Carry Chain
Register
S2

LE2

An
Bn

LUT
Carry Chain
Register
Sn

LEN

LUT
Carry Chain
Register

Carry-Out
LEN+1
Cascade Chains

AND Cascade Chain

OR Cascade Chain
FLEX 8000A Logic Array Block

LAB local Interconnect (32 channels)
LAB Control Signals

Row FastTrack Interconnect

Column-to-Row Interconnect
Column FastTrack Interconnect

Carry-In & Cascade-In from LAB on left

Carry-Out & Cascade-Out to LAB on right
FLEX 8000A FastTrack Interconnect

Row FastTrack (168/216 channels)

Column FastTrack (16 channels)

Local FastTrack (32 channels)

LAB

LE
FLEX 8000A I/O Element

- Programmable Inversion
- Slew-Rate Control

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 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)
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
# FLEX 10K Families

<table>
<thead>
<tr>
<th>Features</th>
<th>EPF10K10</th>
<th>EPF10K10A</th>
<th>EPF10K20</th>
<th>EPF10K30</th>
<th>EPF10K30A</th>
<th>EPF10K40</th>
<th>EPF10K50</th>
<th>EPF10K50V</th>
<th>EPF10K70</th>
<th>EPF10K100</th>
<th>EPF10K100A</th>
<th>EPF10K130V</th>
<th>EPF10K250A</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>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></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

**Embedded Architecture Evolution**
- Dual-Port RAM
- 4-Kbit EAB with x16 width
- PCI-Compliant I/O

**Advanced Process Technology**
- 0.25-μm CMOS SRAM
- Five-layer metal
- 2.5-V core with MultiVolt™ I/O
- 5.0-V tolerant inputs

**DESIGNED FOR PCI**
100-MHz system speed
150-MHz FIFOs
## 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

[Diagram showing the architecture with IOEs, Logic Arrays, LABs, and Embedded Arrays.]

1. IOE
2. 8 IOE
3. 1 IOE
4. 8 IOE

Logic Element
Logic Array
Embedded Array

...
What is the EAB?

◆ 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
- Write Pulse Circuit
- EAB contains registers for incoming and outgoing signals
10KE EAB

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

RAM/ROM 4,096 Bits
- 256x16
- 512x8
- 1024x4
- 2048x2

Write Enable
Read Enable
Write Address
Read Address

Data Out

EAB contains registers for incoming and outgoing signals
FLEX 10K Logic Element

![Diagram of FLEX 10K Logic Element]

- **Look-Up Table (LUT)**
- **Carry Chain**
- **Cascade Chain**
- **Clear/Preset Logic**
- **Programmable Register**

- **Inputs:**
  - DATA1, DATA2, DATA3, DATA4
  - LABCTRL1, LABCTRL2, LABCTRL3, LABCTRL4
  - Device-Wide Clear

- **Outputs:**
  - Carry-In, Cascade-In, Carry-Out, Cascade-Out

- **Controls:**
  - Clock Select
  - PRn, D/T, ENA, CLRn

- **Interconnections:**
  - To FastTrack Interconnect
  - To LAB Local Interconnect
FLEX 10K Register Packing
FLEX 10K Logic Array Block

- Dedicated Inputs & Global Signals
- Row FastTrack Interconnect
- Column-to-Row Interconnect
- Column FastTrack Interconnect

LAB local Interconnect (30/34 channels)
LAB Control Signals

Carry-In & Cascade-In

Carry-Out & Cascade-Out
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

- **OE[7..0]**
- **VCC**
- **GND**
- **ENA**
- **CLRn**
- **Q**
- **VCC**
- **CLK[2..1]**
- **ENA[5..0]**
- **CLRn[1..0]**
- **Peripheral Control Bus[11..0]**
- **2 Dedicated Clock Inputs**
- **Device-Wide Output Disable**
- **Programmable Inversion**
- **Open-Drain Output Slew-Rate Control**

Connections:
- From One Row or Column Channel
- To Row or Column Interconnect
- From Row or Column Interconnect
- From One Row or Column Channel

Slew-Rate Control
Programmable Inversion
Peripheral Control Bus[11..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

![ClockLock Diagram]

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>PS (Passive Serial)</td>
<td>Altera’s EPC1 configuration EPROM, BitBlaster or ByteBlaster download cable, serial data source</td>
</tr>
<tr>
<td>PPS (Passive Parallel Synchronous)</td>
<td>Intelligent host, parallel data source</td>
</tr>
<tr>
<td>PPA (Passive Parallel Asynchronous)</td>
<td>Intelligent host, parallel data source</td>
</tr>
<tr>
<td>JTAG</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
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

![Graph showing the percentage of design starts for different voltage levels from 1992 to 2001. The graph indicates a trend where 5.0 V is decreasing, 3.3 V and 2.5 V are increasing, and 1.8 V remains low. Source: Altera.](image)
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

- 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

2.5-V Input
3.3-V Input
5.0-V Input

Logic

Min. $V_{OH} = 2.1$ V

2.5-V Output Can Drive 2.5-V Devices

2.5-V & 5.0-V Tolerant Input Buffers

2.5-V & 5.0-V Tolerant Input Buffers

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 VCCIO Is Not Equal to VCCINT
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>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>Ok</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</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>FLEX6000A</td>
<td>0.35</td>
<td>3.3</td>
<td>3.3</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>FLEX8000A</td>
<td>.6,.5</td>
<td>5</td>
<td>5</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.6,.5</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>EPF8282AV</td>
<td>.6,.5</td>
<td>3.3</td>
<td>3.3</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>FLEX10K</td>
<td>0.5</td>
<td>5</td>
<td>5</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>FLEX10A</td>
<td>0.35</td>
<td>3.3</td>
<td>3.3</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>EPF10KE</td>
<td>0.22</td>
<td>2.5</td>
<td>3.3</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
# Device feature summary

<table>
<thead>
<tr>
<th>Device</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>(+4)</td>
<td></td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>FLEX6K(A)</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td>(+4)</td>
<td></td>
<td>4</td>
<td>ALL</td>
</tr>
<tr>
<td>MAX9000</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>MAX7000S</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>Y</td>
<td>2</td>
<td></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
FPGA/CPLD Design Flow

- Design Ideas
- Detailed Design
- Functional Simulation
- Implementation (P&R)
- Device Programming
- Timing Simulation

FPGA/CPLD

\[ t_{pd} = 22.1 \text{ ns} \]
\[ f_{max} = 47.1 \text{ MHz} \]
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

- Verilog HDL & VHDL Design Files
- Standard EDA HDL Files
- Standard EDA Schematics
- Standard EDA Simulator
- EDIF
- MAX+PLUS II
- FLEX MAX Classic
- Verilog HDL VHDL EDIF SDF
MAX+PLUS II
Altera’s Fully-Integrated Development System

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

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

**Project Verification**
- MAX+PLUS II Simulator
- MAX+PLUS II Waveform Editor
- MAX+PLUS II Timing Analyzer

**Device Programming**
- MAX+PLUS II Programmer
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 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
Design Entry Files

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

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

- Generated within MAX+PLUS II
- Imported from other EDA tools

Verilog
VHDL
AHDL
Waveform
Schematic

OrCAD
Synopsys, Synplicity, Mentor Graphics, etc...
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
New vs Existing Megafunction

❖ Choose between a new custom megafunction variation or an existing megafunction variation

New Custom Megafunction

Edit Existing Custom Megafunction
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? 83 bits

Do you want to pipeline the multiplexer?
- No
- Yes, I want an output latency of ___ Clock cycles

Create an asynchronous Clear input

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 Megafunction

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
Example: Multiplier

◆ Design a multiplier with \texttt{LPM\_MULT}
  • The easiest way to create a multiplier is to use the \texttt{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 $\text{LPM\_MUX}$
  - Use $\text{WIRE}$ primitive to rename a bus or node
  - $\text{LPM\_MUX}$ 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

咣 Design a sequencer with `LPM_COUNTER & 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 a 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

- 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 route 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
Example: Decoder

◆ Design a decoder with...

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

```vhdl
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;
```

```vhdl
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 \texttt{DFF/DFFE} or \texttt{LPM_COUNTER}

```vhdl
SUBDESIGN ahdlcnt
(
    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;
```

```vhdl
INCLUDE "lpm_counter.inc"
SUBDESIGN lpm_cnt
(
    clk, load, ena, clr, d[15..0] : INPUT;
    q[15..0] : OUTPUT;
)
VARIABLE
    my_cntr: lpm_counter WITH (LPM_WIDTH=16);
BEGIN
    my_cntr.clock = clk;
    my_cntr.aload = load;
    my_cntr.cnt_en = ena;
    my_cntr.aclr = clr;
    my_cntr.data[] = d[];
    q[] = my_cntr.q[];
END;
```
Example: Multiplier

◆ Design a multiplier with LPM_MULT

```vhdl
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

```vhdl
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;
```
Example: Mealy State Machine

- **Mealy state machine**
  - A state machine with asynchronous output(s)

```plaintext
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;
```
Compiler Input and Output Files

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

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

Mapping Files (.imf)

MAX+PLUS II Design Files (.gdf, .tdf, .vhd, .v, .wdf)

Assignments (.acf)

Programming Files (.pof, .sof, .jed)

Functional SNF Files (.snf)

Timing SNF Files (.snf)

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), Wavefrom 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)
To invoke MAX+PLUS II Compiler

Menu: MAX+PLUS II -> Compiler
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
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.
Compiler Modules - (8)

◆ 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 executes 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)
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
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
Routing Statistics

<table>
<thead>
<tr>
<th>Information on Selected Node/Pin/LAB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name is</strong>: <strong>Multiple Items</strong>-**</td>
</tr>
<tr>
<td><strong>Number is</strong>: LC1_B7</td>
</tr>
<tr>
<td><strong>LAB is</strong>: B7</td>
</tr>
<tr>
<td><strong>Row is</strong>: 7</td>
</tr>
<tr>
<td><strong>Logic Cell Fan-In</strong>: 5</td>
</tr>
<tr>
<td><strong>Logic Cell Carry-Out</strong>: Yes</td>
</tr>
<tr>
<td><strong>Logic Cell Fan-Out</strong>: 9</td>
</tr>
<tr>
<td><strong>Logic Cell Cascade-Out</strong>: No</td>
</tr>
<tr>
<td><strong>Cell Total Shared Expanders Used</strong>:</td>
</tr>
<tr>
<td><strong>Embedded Cell Depth (Bits)</strong>:</td>
</tr>
<tr>
<td><strong>LAB Total Shared Expanders Used</strong>:</td>
</tr>
<tr>
<td><strong>LAB External Interconnect Used</strong>: 6/24 (25%)</td>
</tr>
<tr>
<td><strong>Column Interconnect Channels Used</strong>:3/16 (18%)</td>
</tr>
<tr>
<td><strong>Full Row Interconnect Channels Used</strong>:128/168 (76%)</td>
</tr>
<tr>
<td><strong>Half Row Interconnect Channels Used</strong>:</td>
</tr>
<tr>
<td><strong>Logic Cell Inputs Borrowed from LC1</strong>:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Most Congested Areas in Current Chip</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Most Congested LAB (or EAB) is</strong>: A20</td>
</tr>
<tr>
<td><strong>LAB (or EAB) External Interconnect Used</strong>: 15/24 (62%)</td>
</tr>
<tr>
<td><strong>Most Congested Row is</strong>: 8</td>
</tr>
<tr>
<td><strong>Most Congested Column is</strong>: 6</td>
</tr>
<tr>
<td><strong>Column Interconnect Channels Used</strong>: 10/16 (62%)</td>
</tr>
</tbody>
</table>
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 this Column
Anywhere on Device

Anywhere on this Row
Project Verification Methodology

MAX+PLUS II Timing Analyzer

MAX+PLUS II Simulator

MAX+PLUS II Waveform Editor

- .snf
- .mif
- .hex
- .cm
- .tao
- .hst
- .sif
- .log
- .tbl
- .scf
- .vec
- .tbl
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
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

Initialize Memory

Address | Value
00 | 0000 0000 0000 0000 0000 0000 0000 0000
08 | FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
10 | 0000 0000 0000 0000 0000 0000 0000 0000
18 | FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
20 | 0000 0000 0000 0000 0000 0000 0000 0000
28 | FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
30 | 0000 0000 0000 0000 0000 0000 0000 0000
38 | FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
40 | 0000 0000 0000 0000 0000 0000 0000 0000
48 | FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF

Value Radios:
- BIN
- OCT
- DEC
- HEX

Add Radios:
- BIN
- OCT
- DEC
- HEX

List One Address Per Line

Memory Info:
- Depth: 256
- Width (Bits): 16

Type:
- RAM

Initialize to 0s
Initialize to 1s

Export Memory Content File

File Name: *.mil

Automatic Extension:
- .mil
- .hex

Import File... Export File...
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
:02000800fffff8
:02000900fffff7
:02000a00fffff6
:02000b00fffff5
:02000c00fffff4
:02000d00fffff3
:02000e00fffff2
:02000f00fffff1
...
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:

```
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     : F E 5; % Range starting from specific address
```
Notes for Compiling & Simulating RAM / ROM - (1)

❖ 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
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...

```
Destination

<table>
<thead>
<tr>
<th></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