



CIC Training Manual

Altera Training Course - for PC Users



July 1998

Course Outline - 1

- ◆ Introduction to PLD
- ◆ Altera Device Families
- ◆ Design Flow & Altera Tools
- ◆ Getting Started
- ◆ Graphic Design Entry
- ◆ AHDL Design Entry
- ◆ Waveform Design Entry

Course Outline - 2

◆ Design Implementation

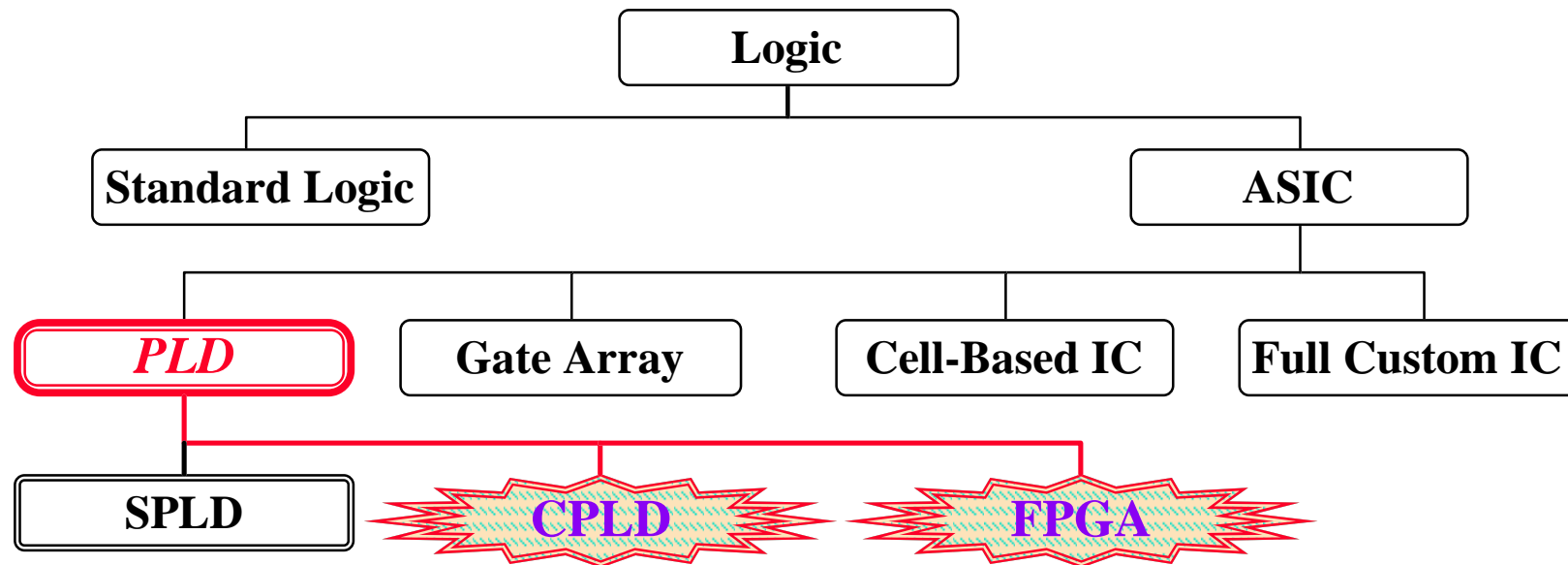
◆ Project Verification

- Functional Simulation
- Timing Analysis
- Timing Simulation

◆ Device Programming

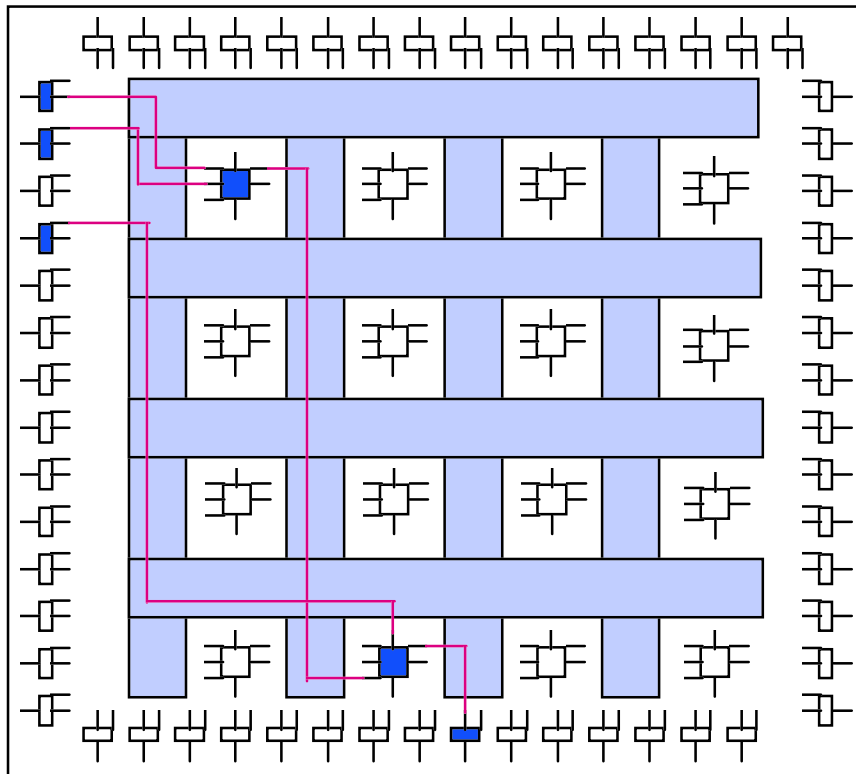
◆ Summary & Getting Help

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

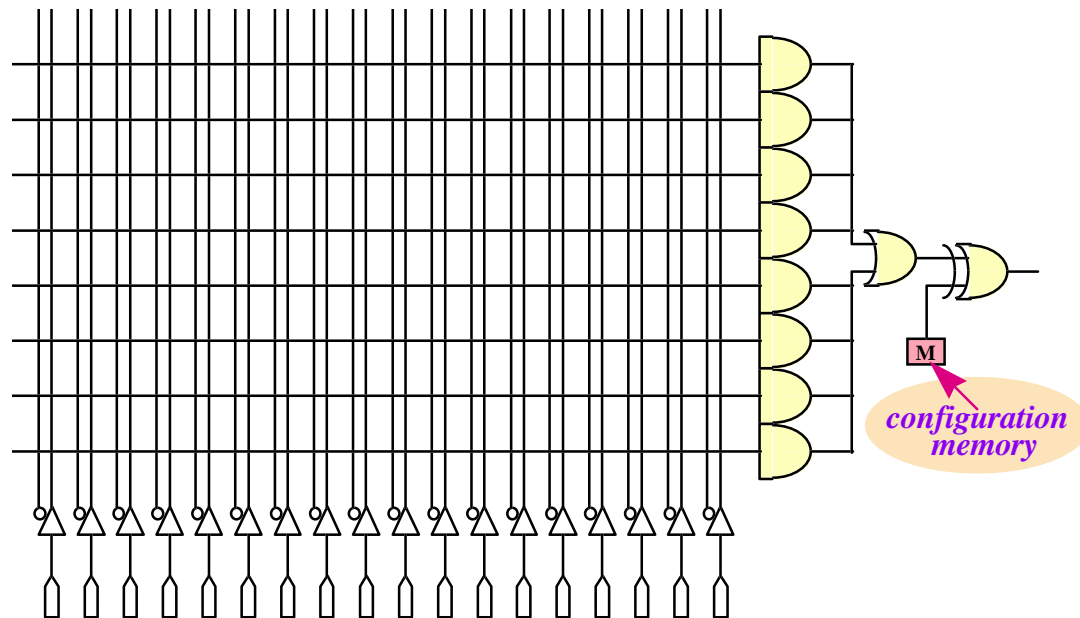


- ◆ Field-programmable
- ◆ Reprogrammable
- ◆ In-circuit design verification
- ◆ Rapid prototyping
- ◆ Fast time-to-market
- ◆ No IC-test & NRE cost
- ◆ H/W emulation instead of S/W simulation
- ◆ Good software
- ◆ ...

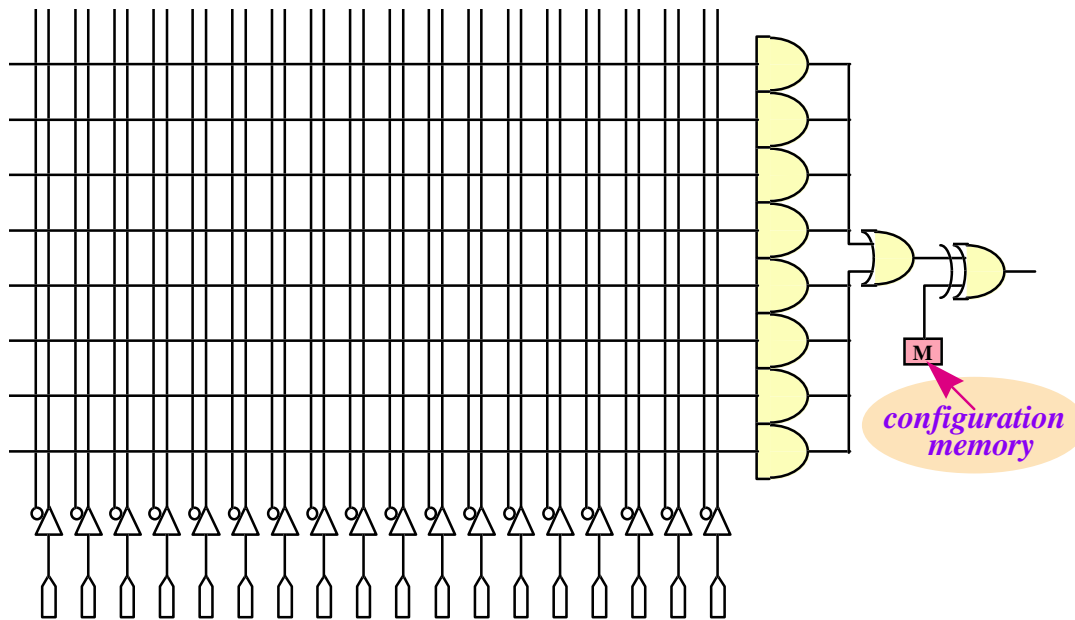
Programmability

◆ Why programmable? Why reprogrammable?

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

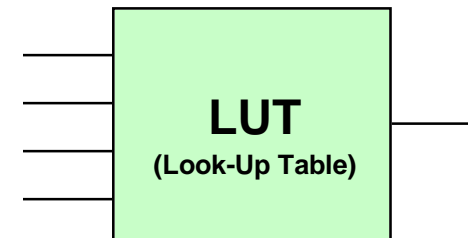


Programmable Combinational Logic



Product Term-based Building Block

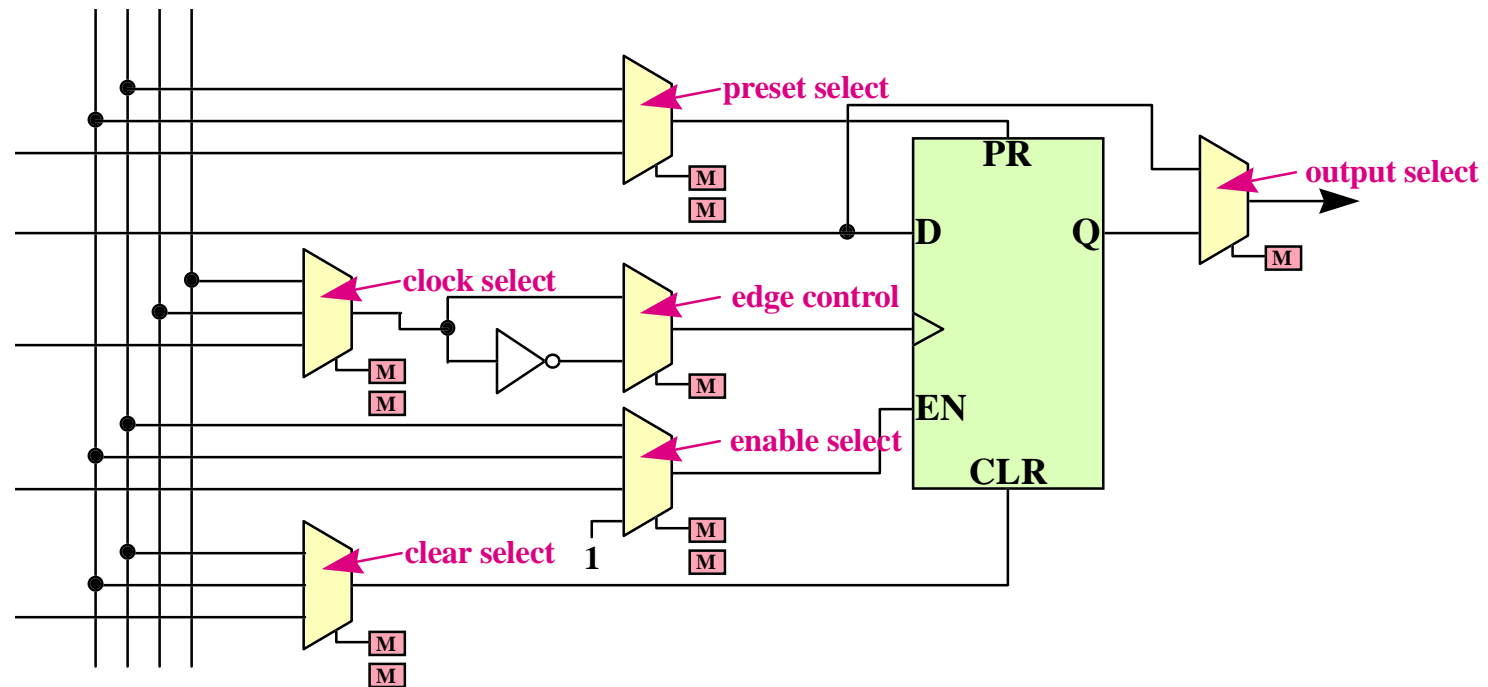
- * 2-level logic
- * High fan-in



Look-up Table-based Building Block

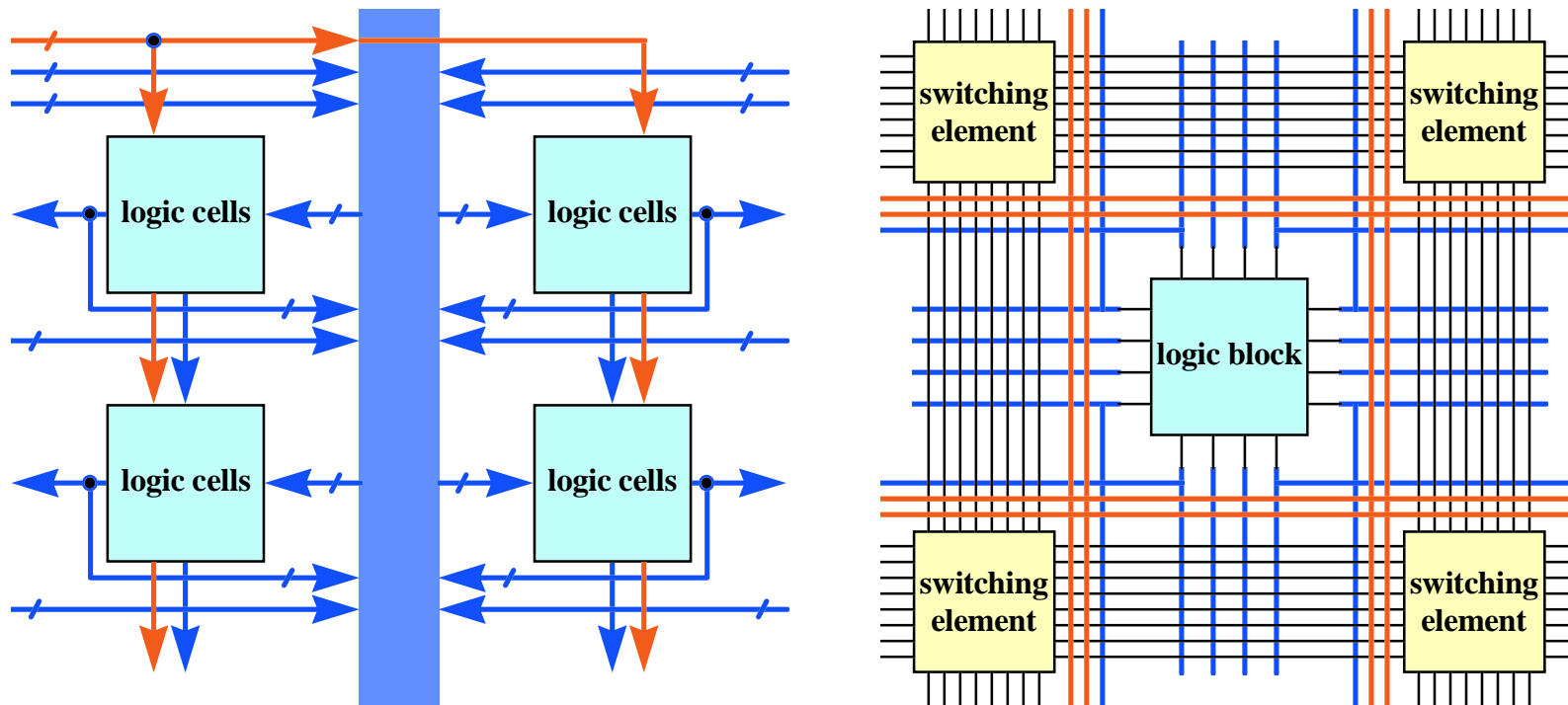
- * 4 to 5 inputs, fine grain architecture
- * ROM-like

Programmable Register



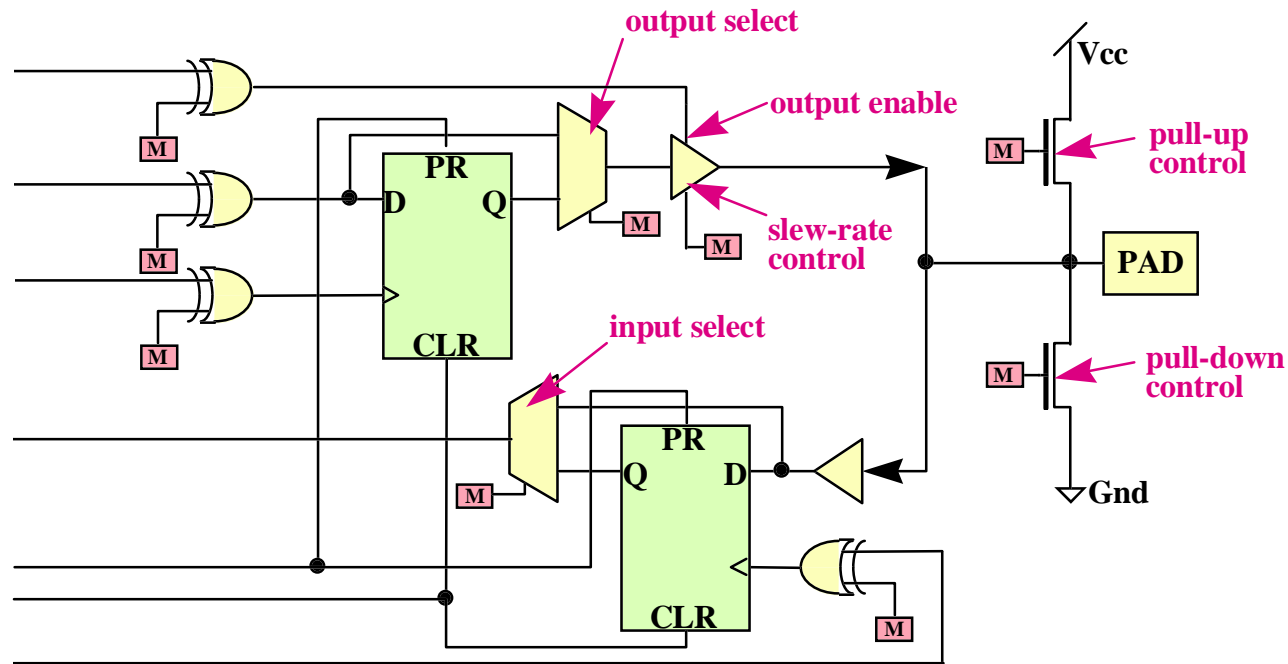
* Typical register controls: clock, enable, preset/clear, ...

Programmable Interconnect



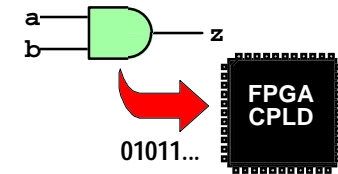
Typical routing resources: switching elements, local/global lines, clock buffers...

Programmable I/O



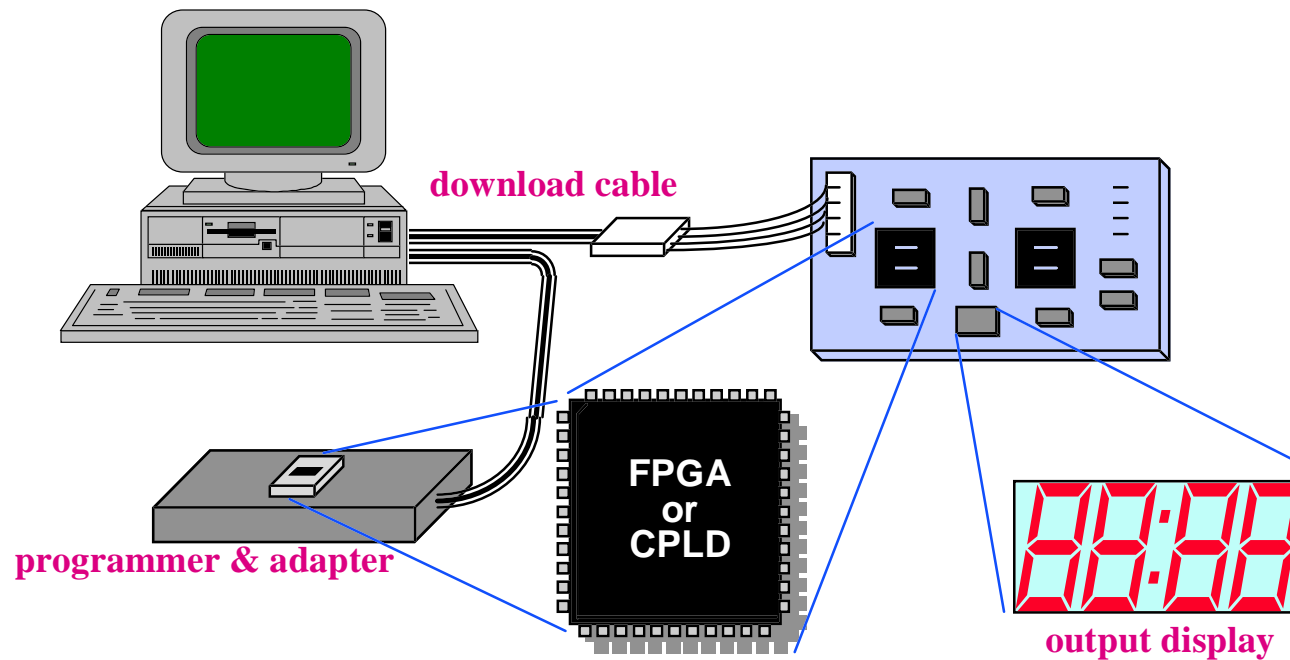
Typical I/O controls: direction, I/O registers, 3-state, slew rate, ...

Field-Programmability



◆ Why filed-programmable?

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

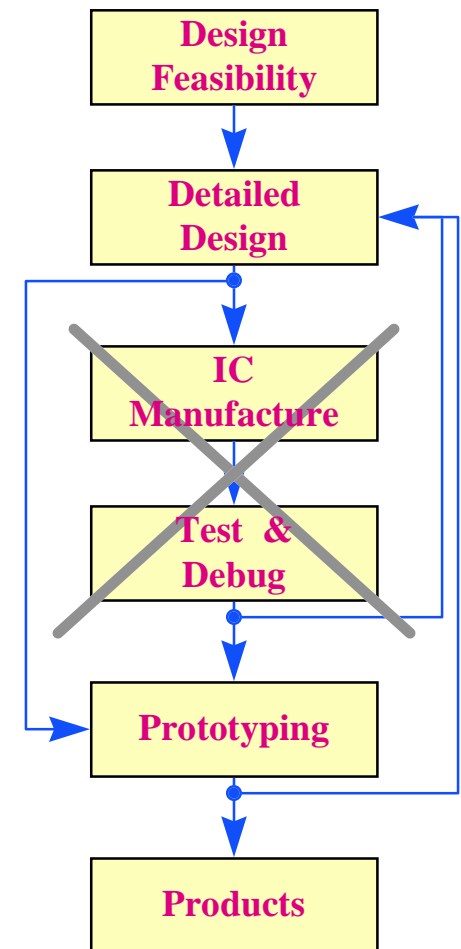
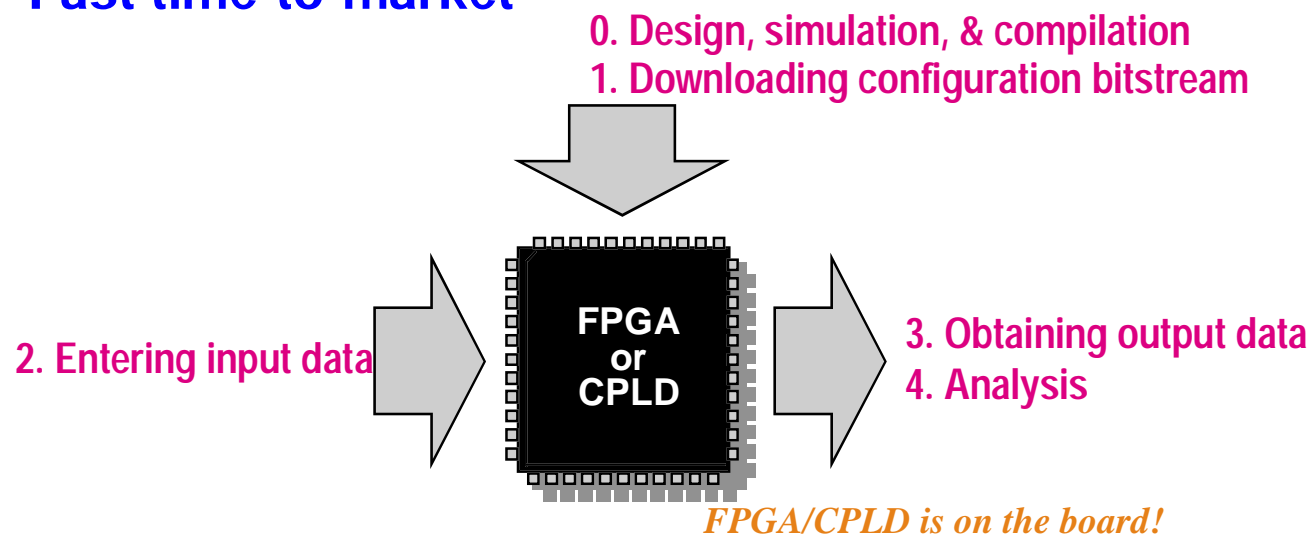


Rapid Prototyping

◆ 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



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

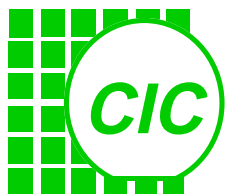
- Functional simulation & timing simulation netlist extraction

◆ Programmer/download program

FPGA/CPLD Benefits

	Full-Custom ICs	Cell-Based ICs	Gate Arrays	High-Density PLDs
Speed	✓✓	✓	✓	✓
Integration Density	✓✓	✓	✓	✓
High-Volume device cost	✓✓	✓✓	✓	✓
Low-volume device cost			✓	✓✓
Time to Market			✓	✓✓
Risk Reduction				✓✓
Future Modification				✓✓
Development Tool	✓	✓	✓	✓✓
Educational Purpose				✓✓

✓ Good
✓✓ Excellent



Altera & CIC



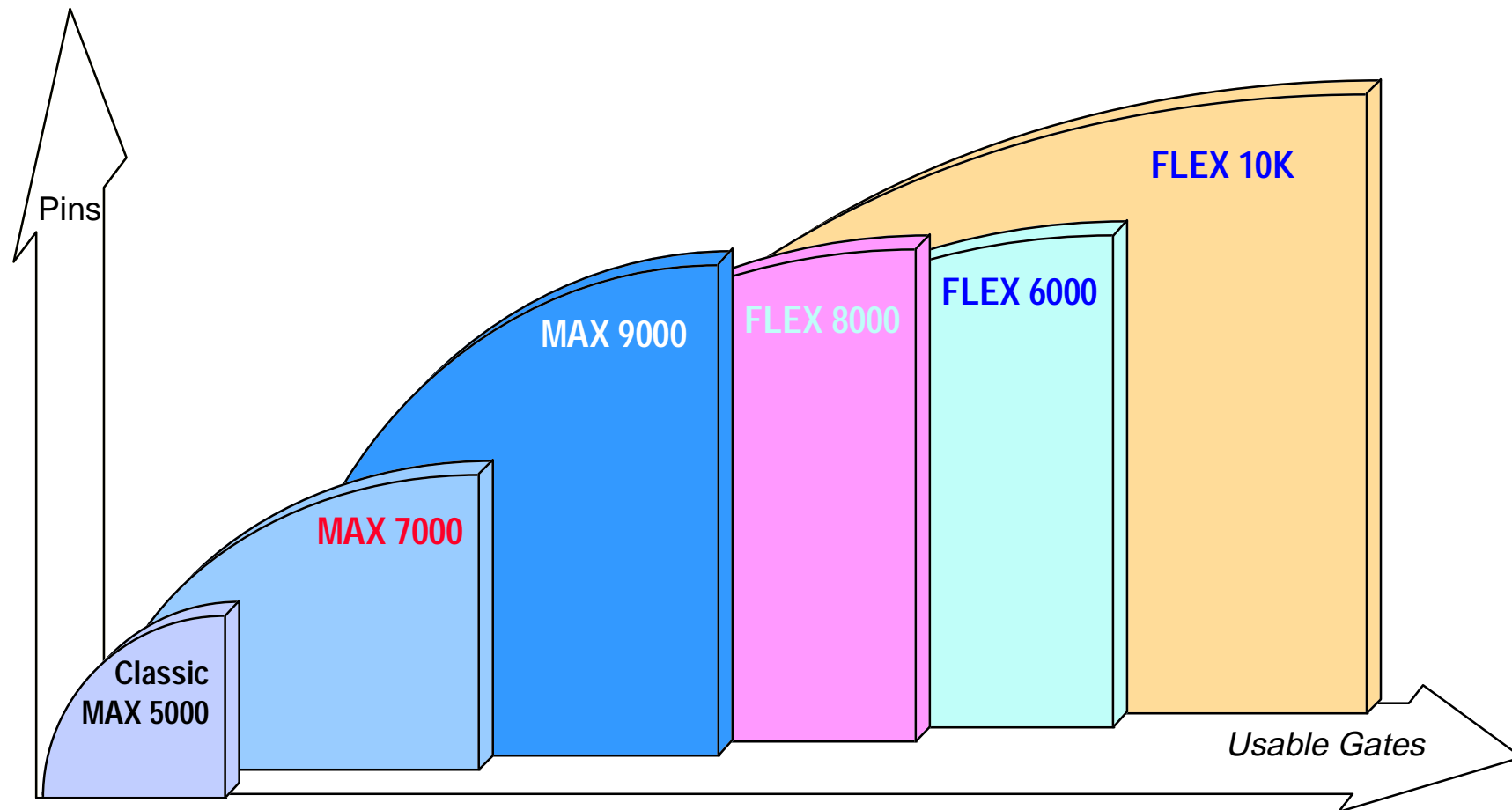
◆ Altera

- 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 - *it's awarded to good software applicants!*
 - 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/html/software/Altera>

Altera Device Families



Altera Device Families

◆ Altera offers 7 device families

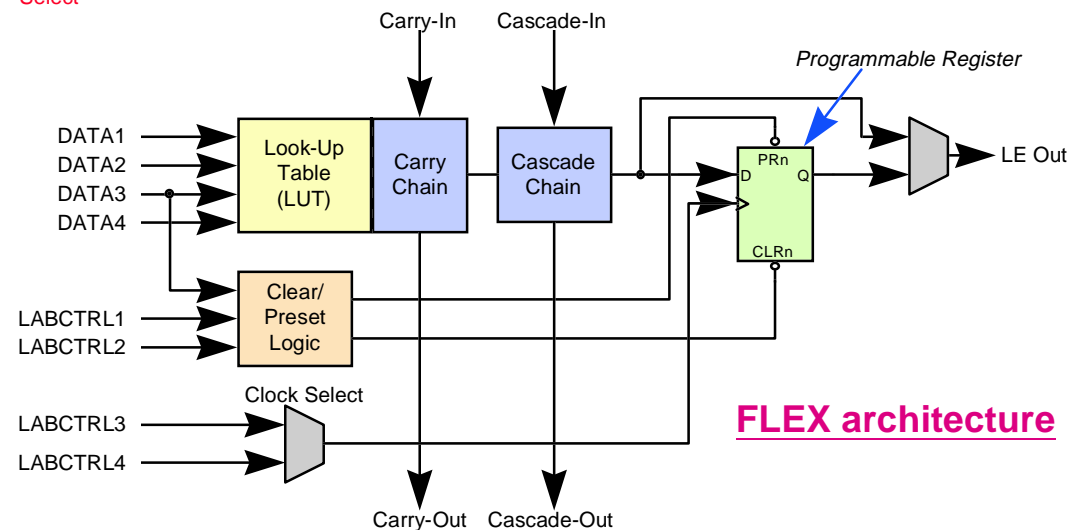
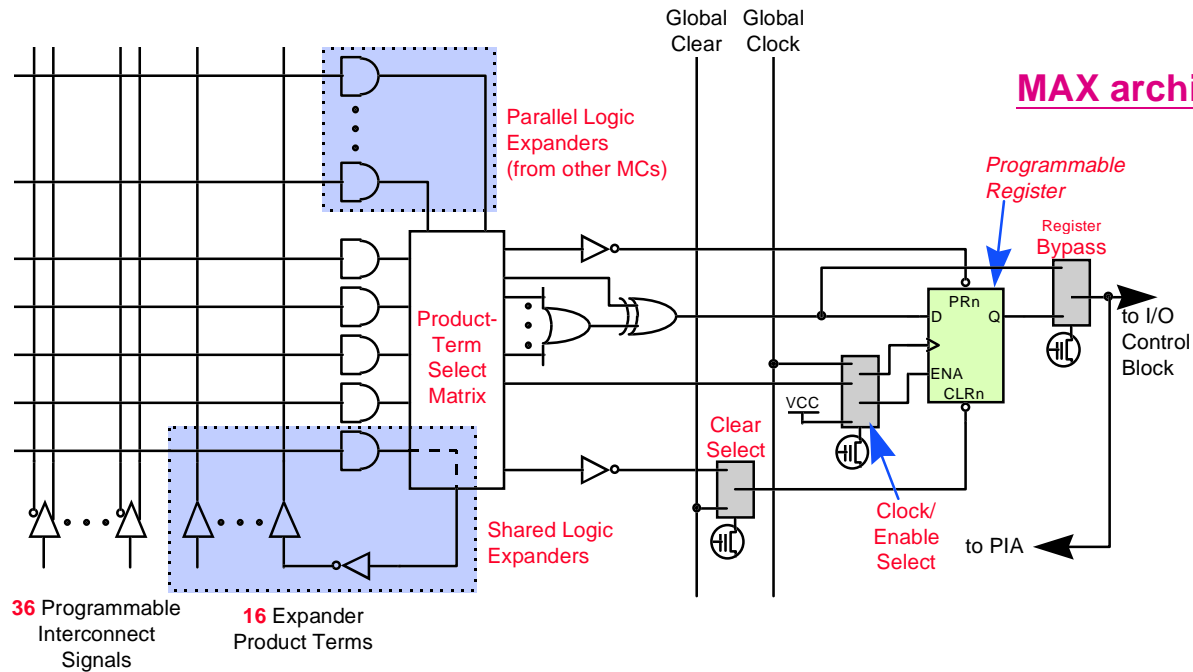
Device Family	Reconfigurable Element	Logic Cell Structure	Usable/Typical Gates	Family Members
Classic	EPROM	SOP	200 ~ 900	EP610, 910, 1810
MAX 5000	EPROM	SOP	800 ~ 3,200	EPM5032, 064, 128, 130, 192
MAX 7000/E/S	EEPROM	SOP	600 ~ 5,000	EPM7032/V/S, 064/S, 096/S, EPM7128E/S, 160E/S, 192E/S, 256E/S
FLEX 6000 ⁽¹⁾	SRAM	LUT	10,000 ~ 24,000	EPF6016/A, 024A
FLEX 8000A	SRAM	LUT	2,500 ~ 16,000	EPF8282A, 452A, 636A, 820A, 1188A, 1500A
MAX 9000/A ⁽¹⁾	EEPROM	SOP	6,000 ~ 12,000	EPM9320/A, 400/A, 480/A, 560/A
FLEX 10K/A/B ⁽¹⁾	SRAM	LUT	10,000 ~ 100,000	EPF10K10/A, 20/A, 30/A, 40/A, 50/V/A, EPF10K70/V/A, 100/A, 130/V/A, 250A

Note:

(1) Not all devices are currently available.

(2) Altera plans to ship new MAX7000A family in the near future.

MAX & FLEX Architectures - (1)



MAX & FLEX Architectures - (2)

◆ Choose the appropriate architecture

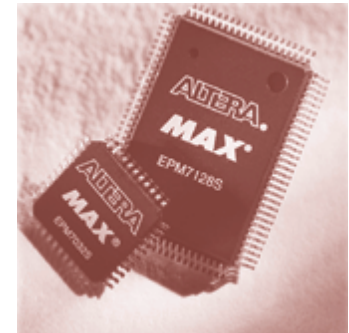
- Different PLD architectures provide different performance & capacity results for same application

Feature	MAX Architecture	FLEX Architecture
Basic Building Block	Course Grain	Fine Grain
Logic Cell Structure	SOP	LUT
Technology	EEPROM	SRAM
Optimization	Combinational-Intensive Logic e.g. Large Decoders, State Machines, ...	Register-Intensive, Arithmetic Functions e.g. Adders, Comparators, Counters, ...

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

Device	MCs	Gates	Speed Grade	Package Options	I/O Pins
EPM7032	32	600	-5,-6,-7,-10,-12,-15	PLCC44, TQFP44	36
EPM7032V	32	600	-12,-15,-20	PLCC44, TQFP44	36
EPM7064	64	1,250	-6,-7,-10,-12,-15	PLCC44/68/84, PQFP100, TQFP44	36,52,68
EPM7096	96	1,800	-7,-10,-12,-15	PLCC68/84, PQFP100	52,64,76
EPM7128E	128	2,500	-7,-10,-10P,-12,-15,-20	PLCC84, PQFP100/160	68,84,100
EPM7160E	160	3,200	-10,-10P,-12,-15,-20	PLCC84, PQFP100/160	64,84,100
EPM7192E	192	3,750	-12,-12P,-15,-20	PQFP160, PGA160	124
EPM7256E	256	5,000	-12,-12P,-15,-20	PQFP160, PGA192, RQFP208	132,164
EPM7032S	32	600	-5,-6,-7,-10	PLCC44, TQFP44	36
EPM7064S	64	1,250	-6,-7,-10	PLCC44/84, PQFP100, TQFP44/100	36,52,68
EPM7096S	96	1,800	-6,-7,-10	PLCC84, PQFP100, TQFP100	52,64,76
EPM7128S	128	2,500	-7,-10,-15	PLCC84, PQFP100/160, TQFP100	68,84,100
EPM7160S	160	3,200	-7,-10,-15	PLCC84, PQFP100/160, TQFP100	64,84,104
EPM7192S	192	3,750	-7,-10,-15	PQFP160	124
EPM7256S	256	5,000	-7,-10,-12,-15	PQFP160, RQFP208	132,164

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

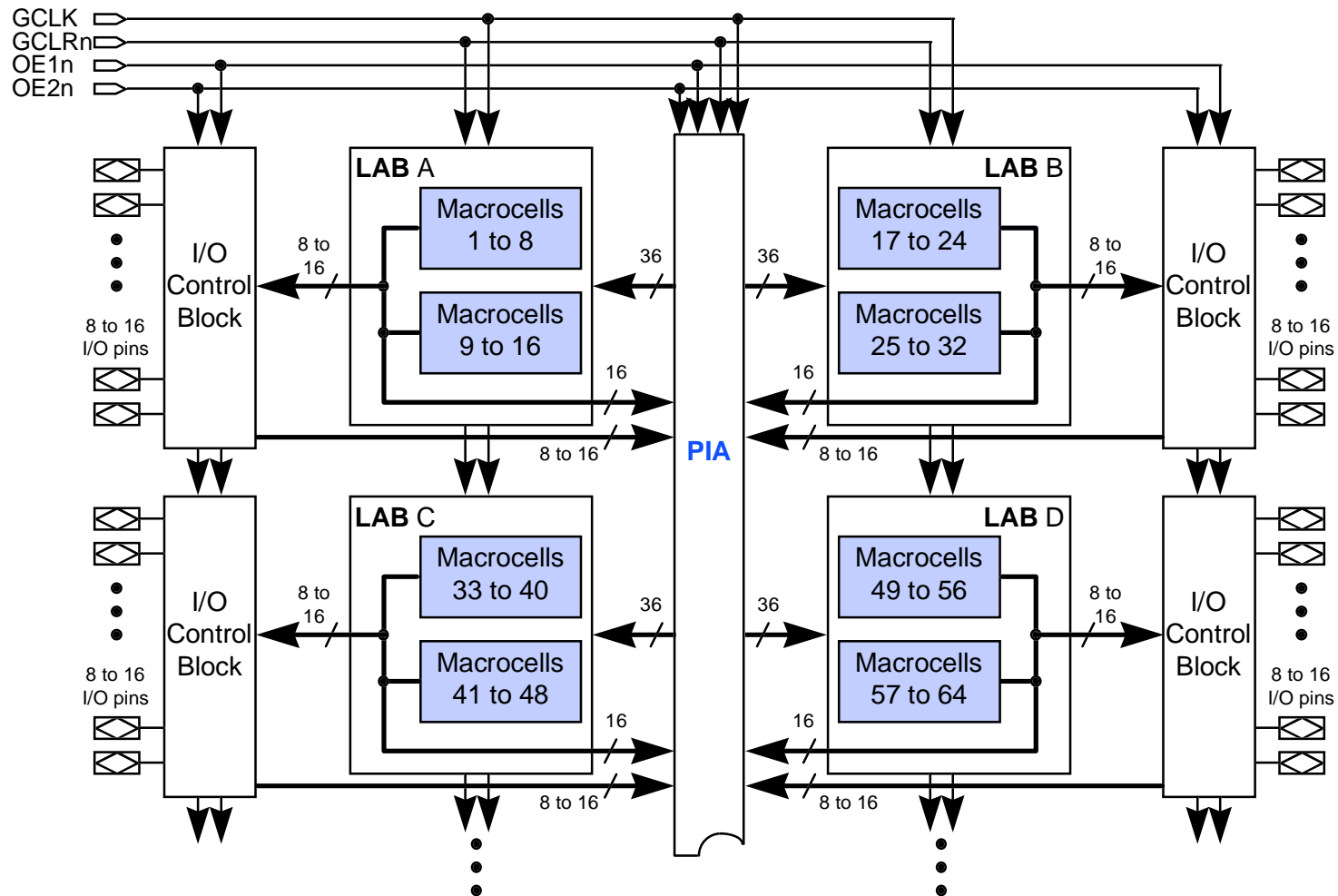
◆ MAX 7000E (128MCs and up) enhanced features...

- More output enable control signals & more global clocking capabilities
- Fast input registers
- Programmable output **slew-rate control**
- More interconnect resources

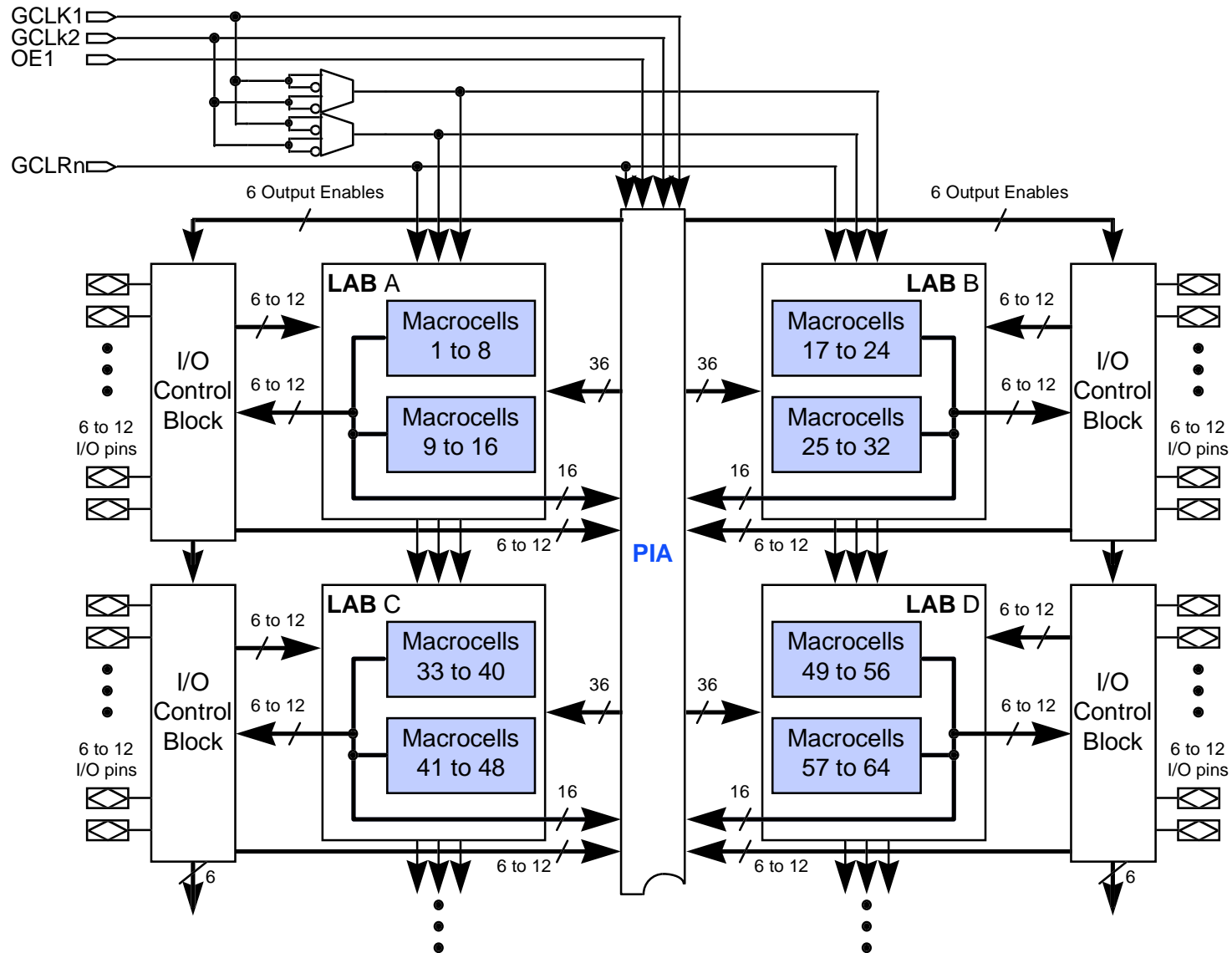
◆ MAX 7000S enhanced features

- 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

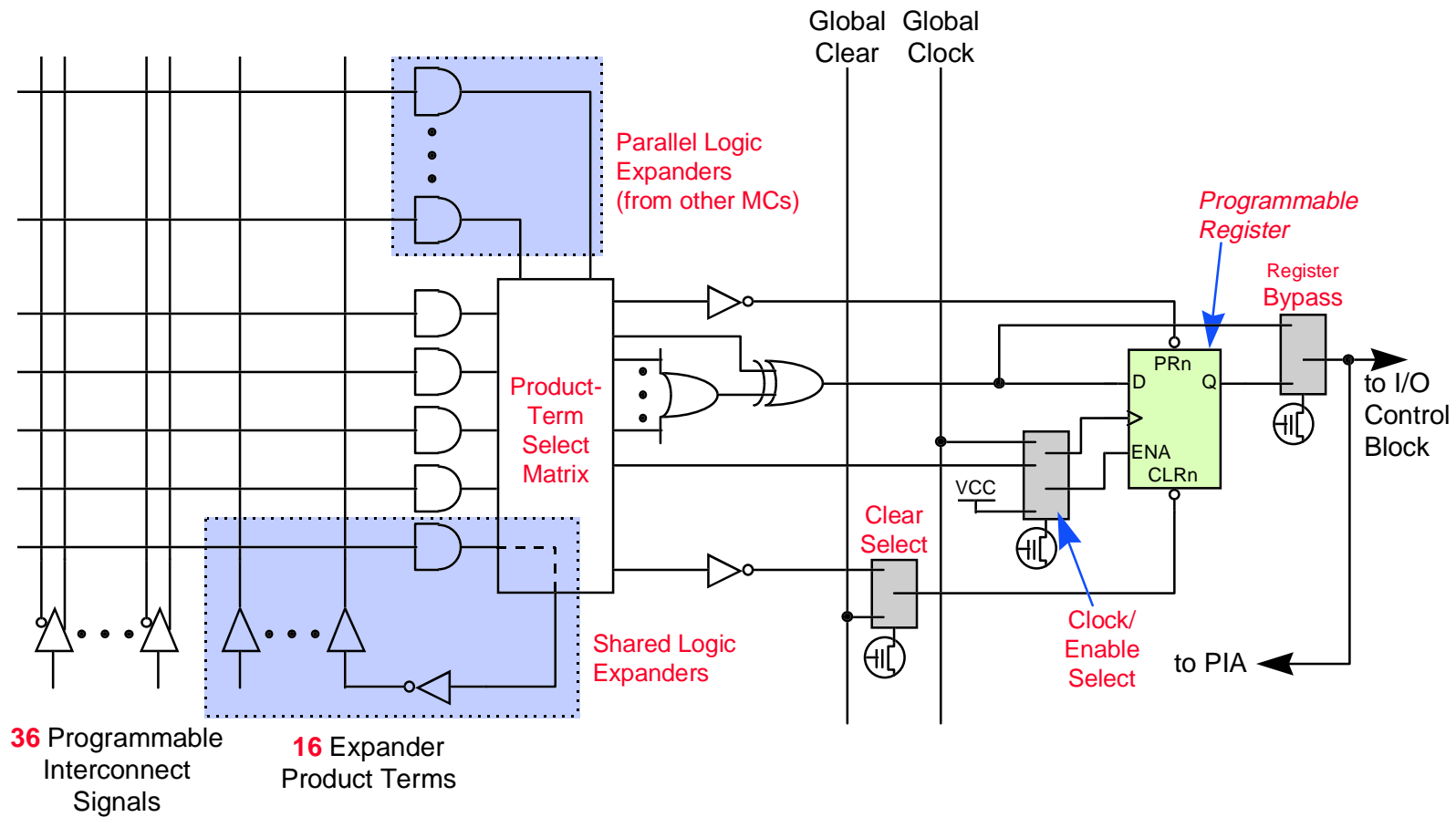
MAX 7000 Architecture



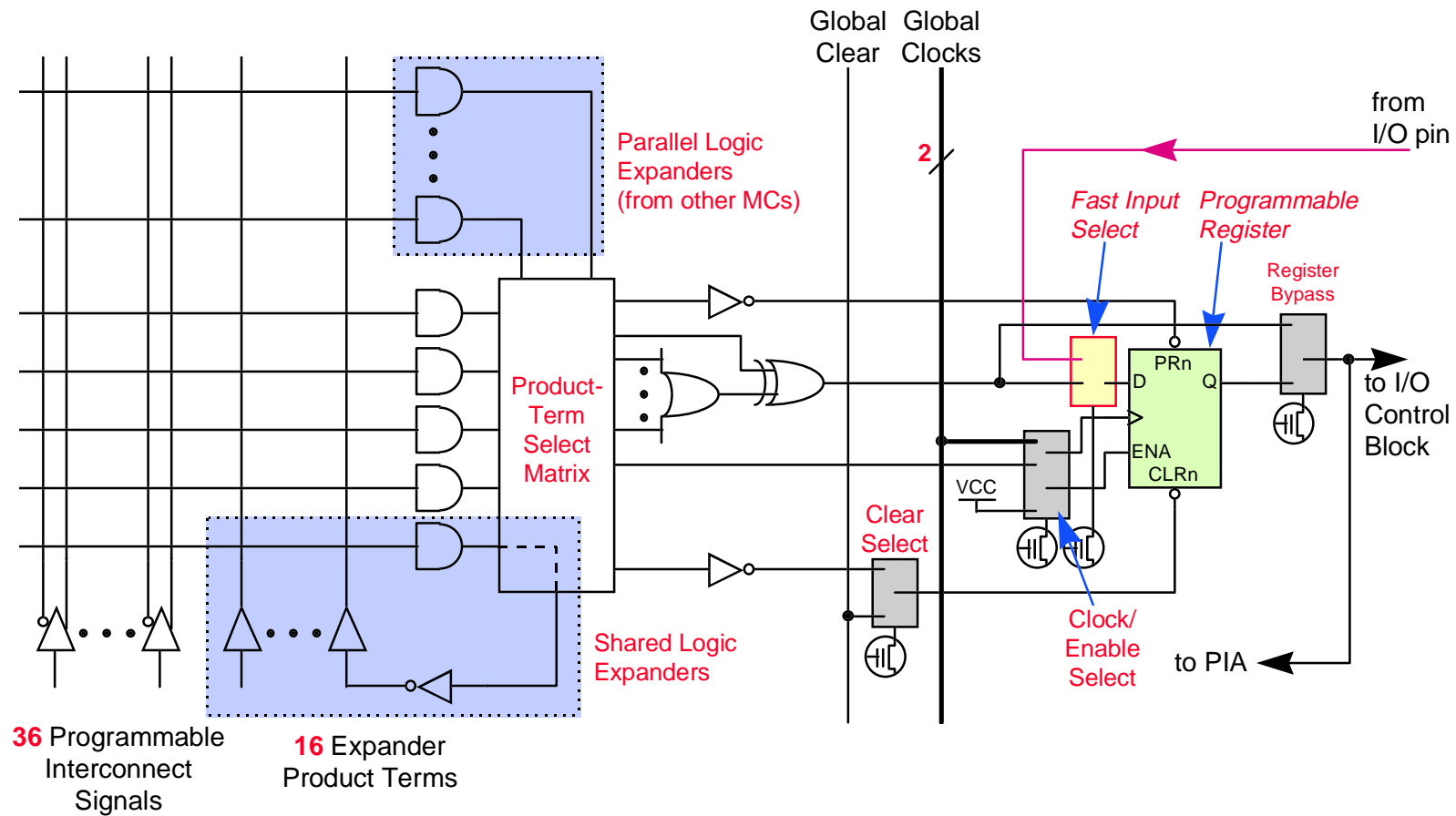
MAX 7000E/S Architecture



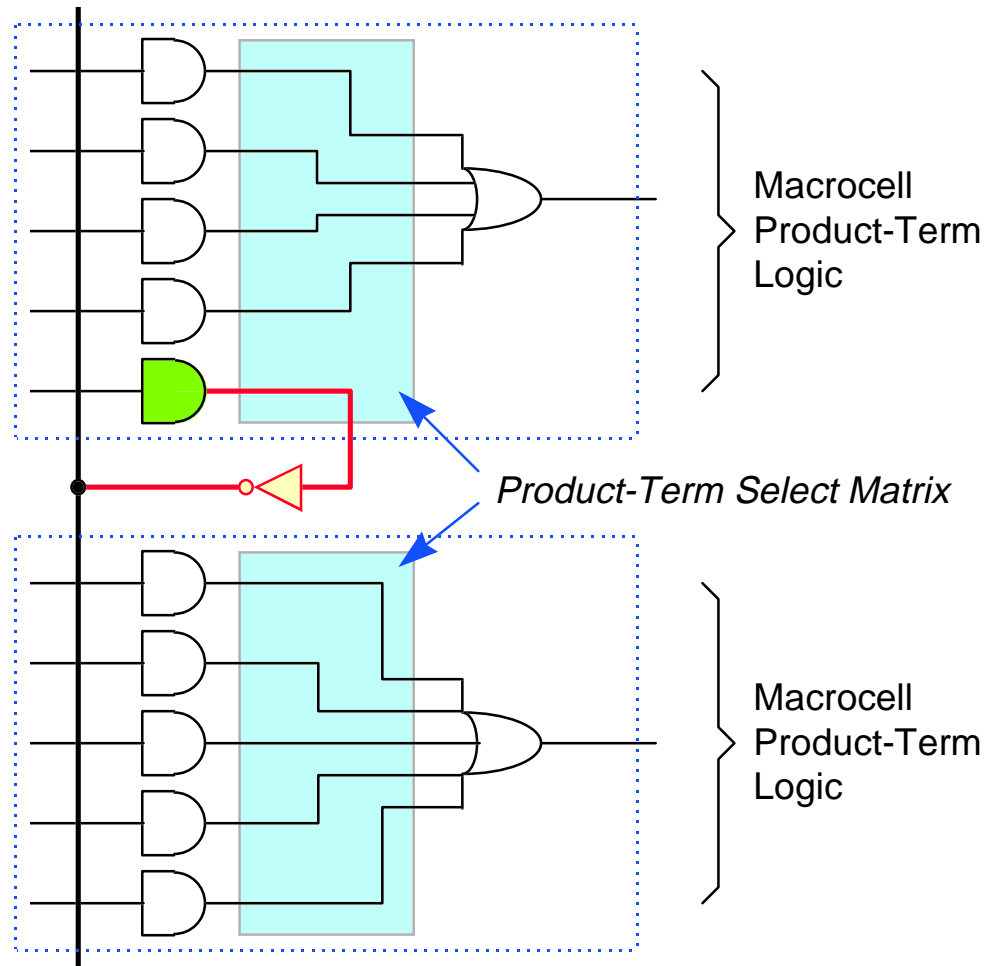
MAX 7000 Macrocell



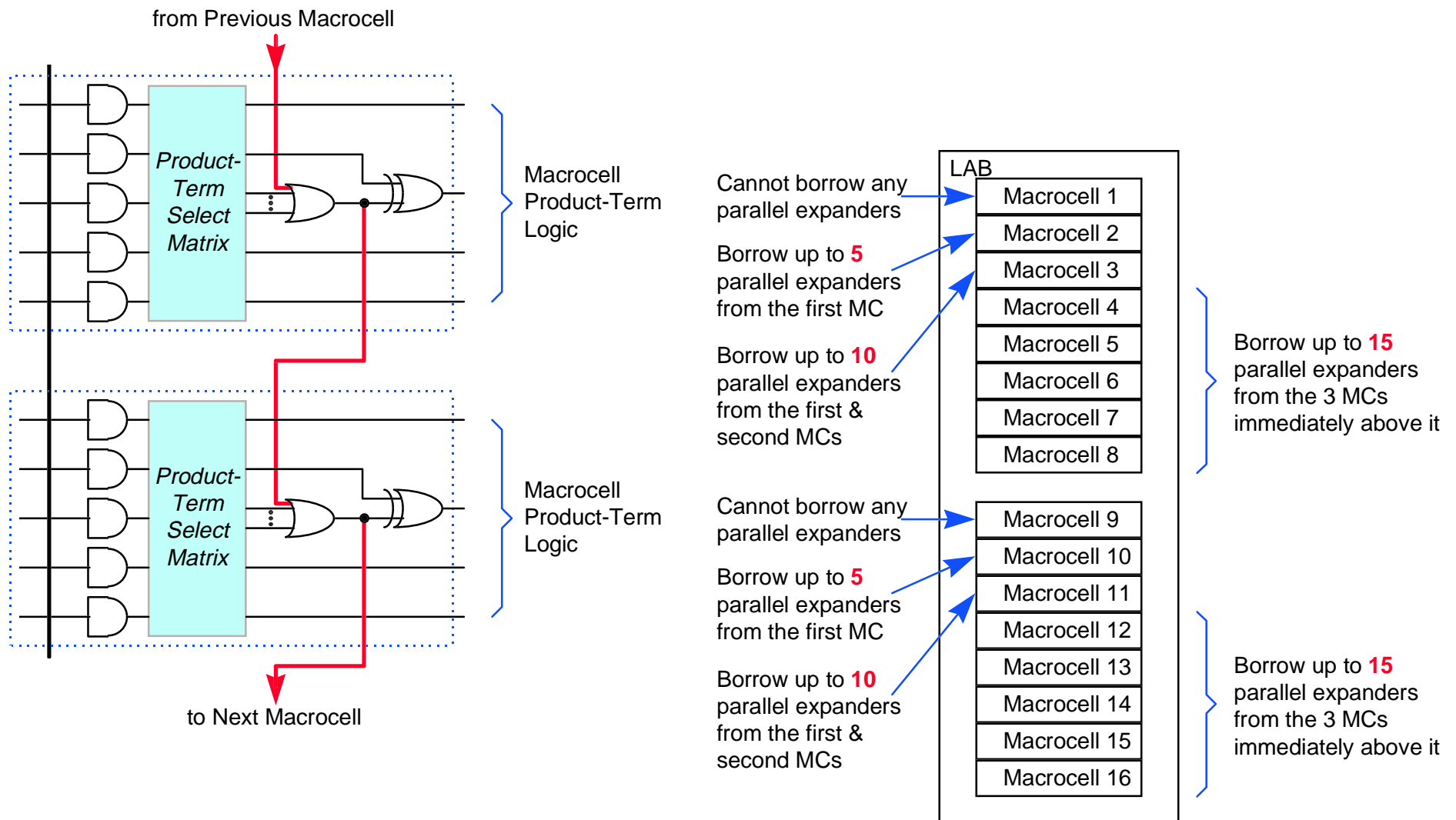
MAX 7000E/S Macrocell



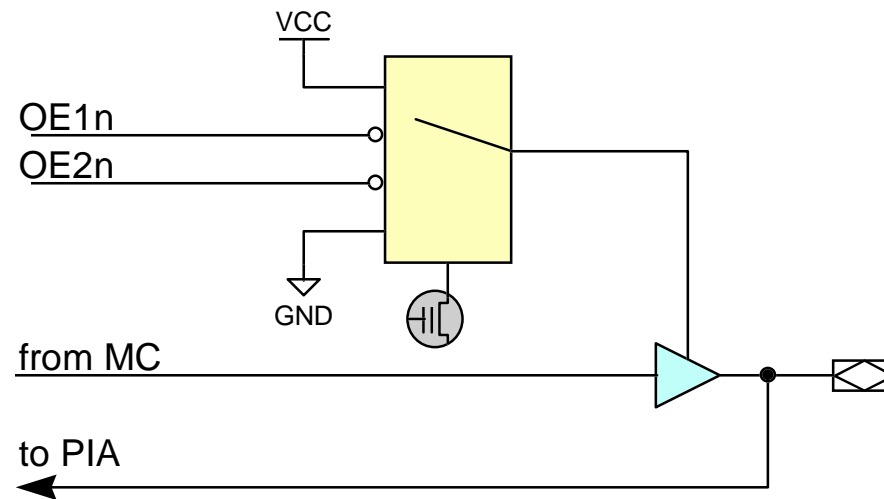
Shareable Expanders



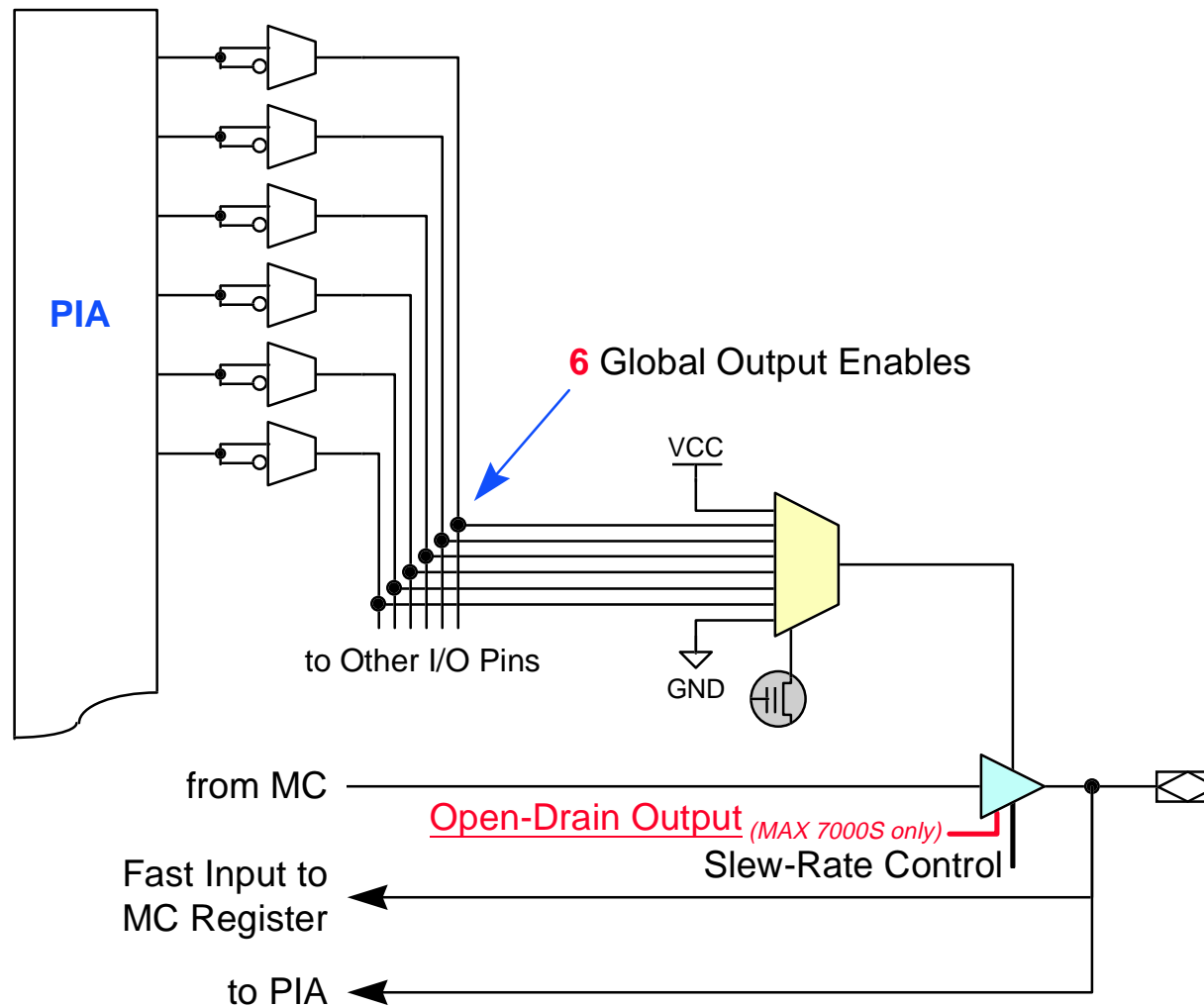
Parallel Expanders



MAX 7000 I/O Control Block

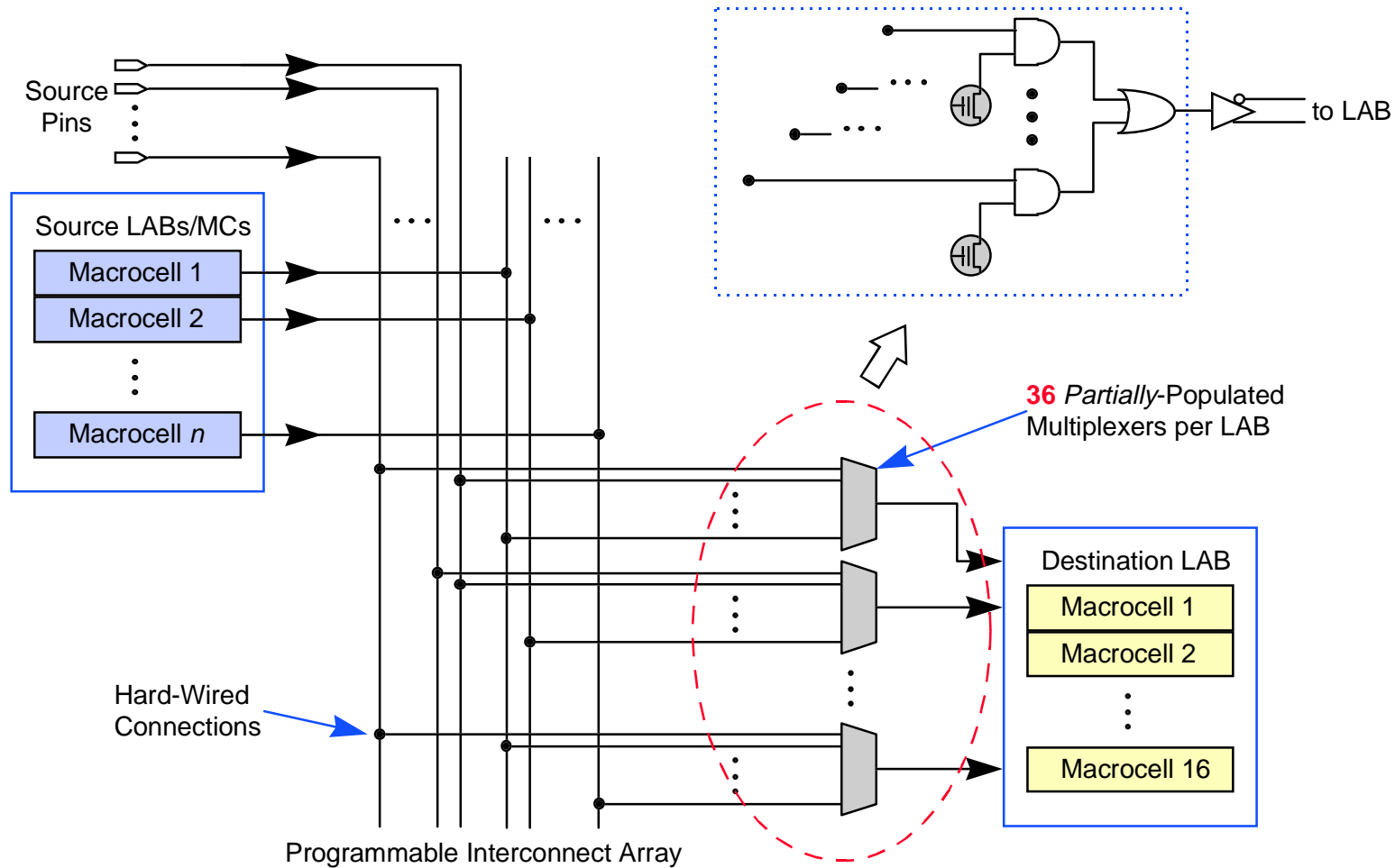


MAX 7000E/S I/O Control Block



MAX 7000/E/S PIA

(Programmable Interconnect Array)

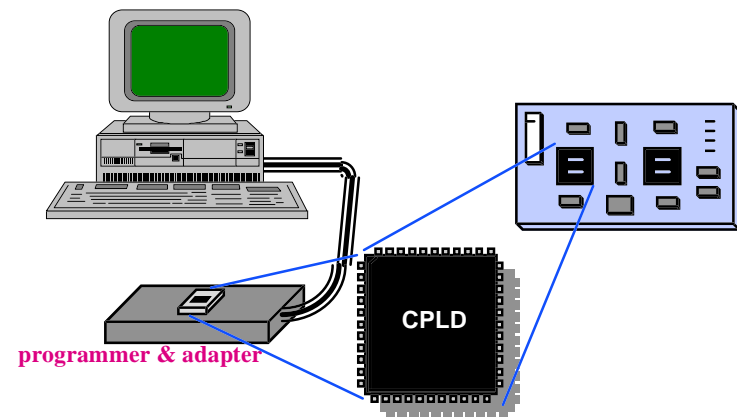


MAX 7000/E/S Device Programming

◆ 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

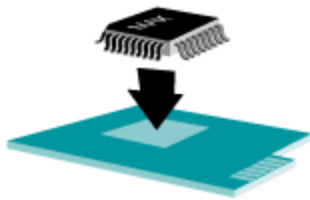
◆ MAX 7000S ISP



What's ISP?

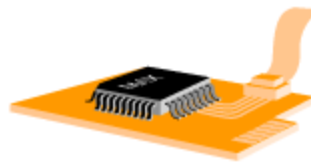
◆ ISP: In-System Programming

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



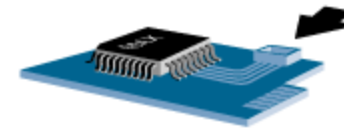
Mount Unprogrammed

- * Eliminates handling of devices
- * Prevents bent leads



Program In-System

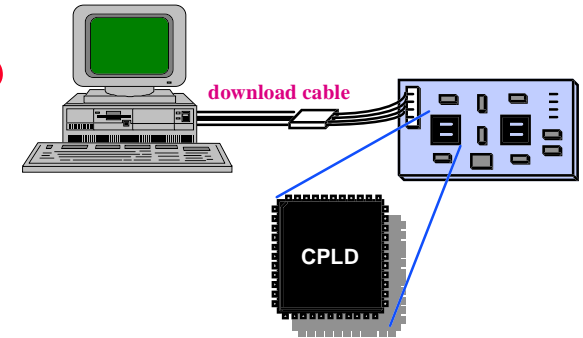
- * Allows generic end-product inventory
- * Specific test protocol or algorithm can be programmed during manufacturing or test flow



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*

FLEX 8000A Family



◆ Today's FLEX 8000A family members

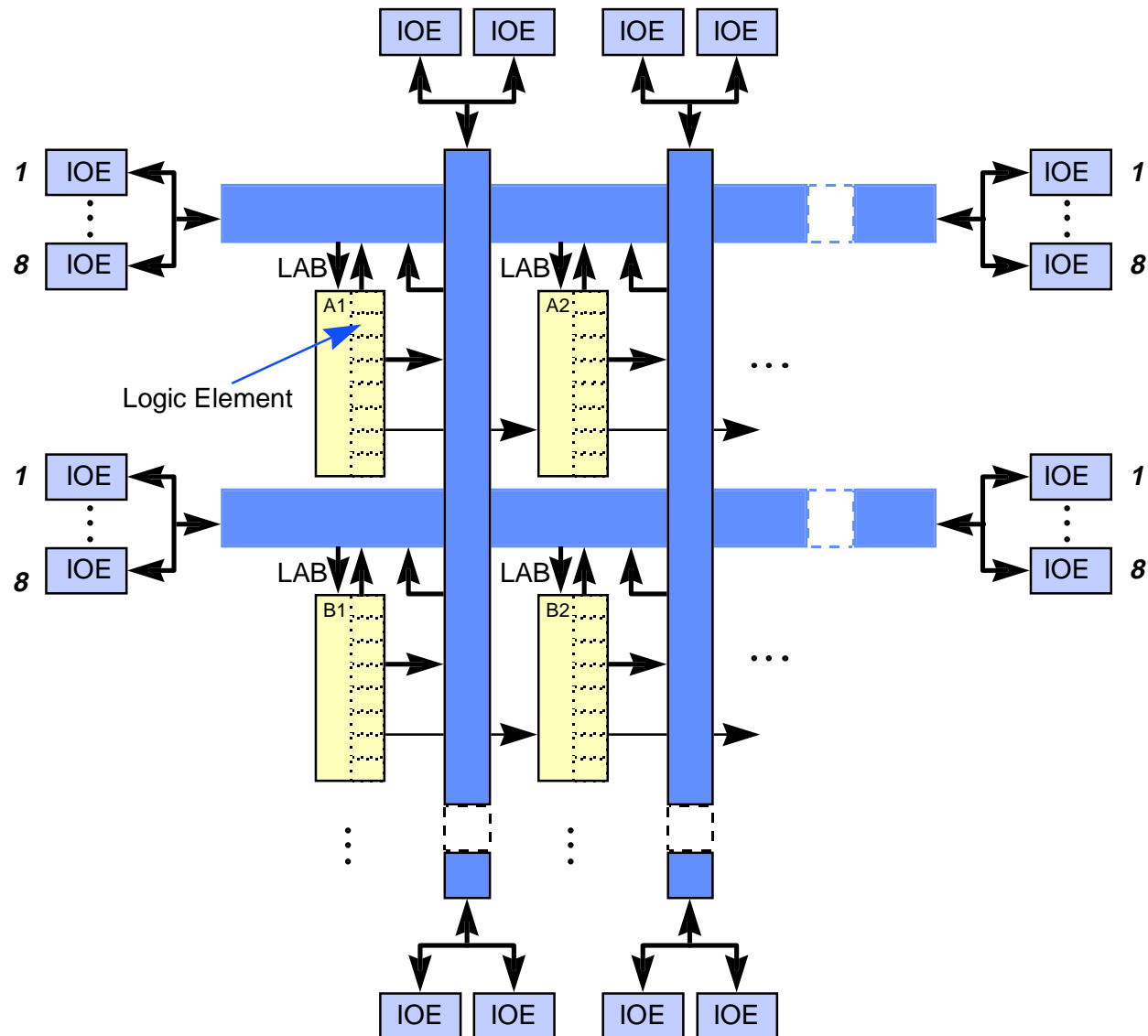
Device	Gates	LEs	FFs	Speed Grade	Package Options	I/O Pins
EPF8282A	2,500	208	282	-2,-3,-4	PLCC84, TQFP100	68,78
EPF8282AV	2,500	208	282	-4	TQFP100	68,78
EPF8452A	4,000	336	452	-2,-3,-4	PLCC84, TQFP100, PQFP160, PGA160	68,120
EPF8636A	6,000	504	636	-2,-3,-4	PLCC84, PQFP160/208, PGA192	68,118,136
EPF8820A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

FLEX 8000A Features

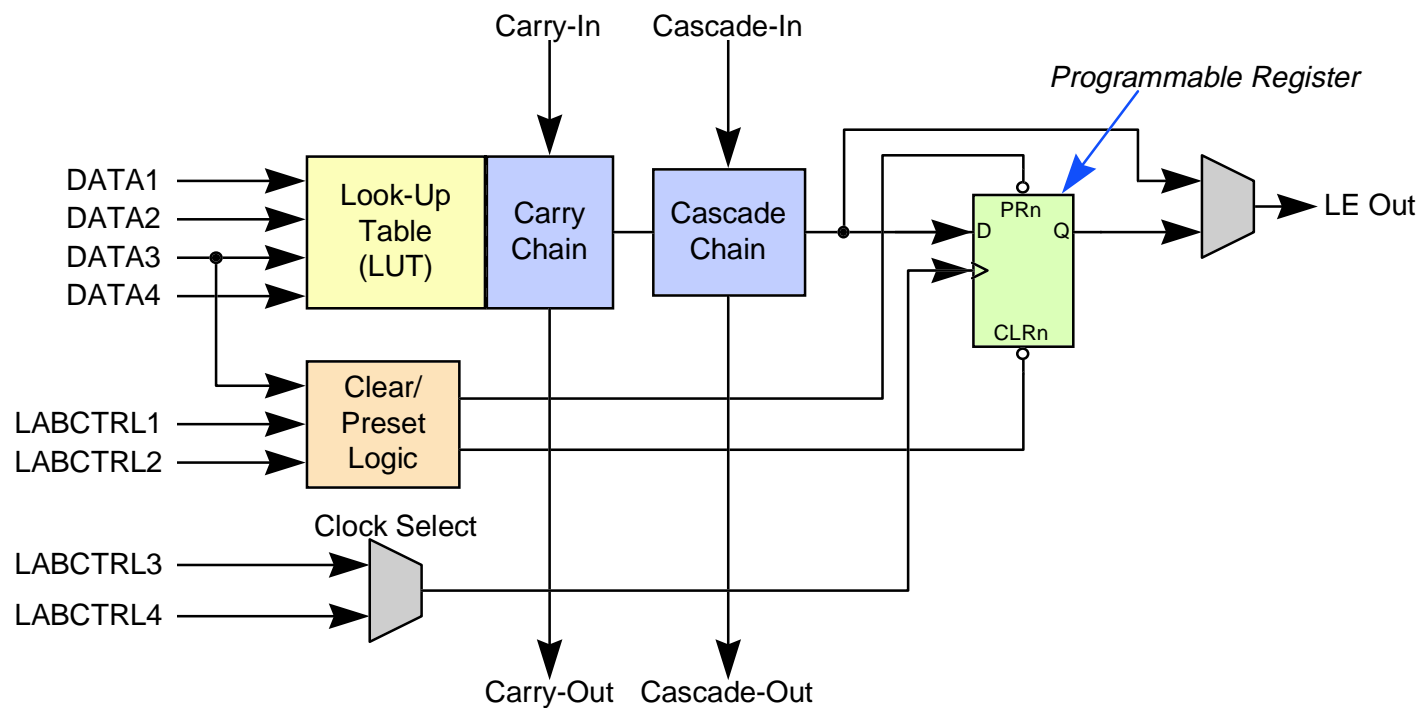
◆ FLEX 8000A main features...

- 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

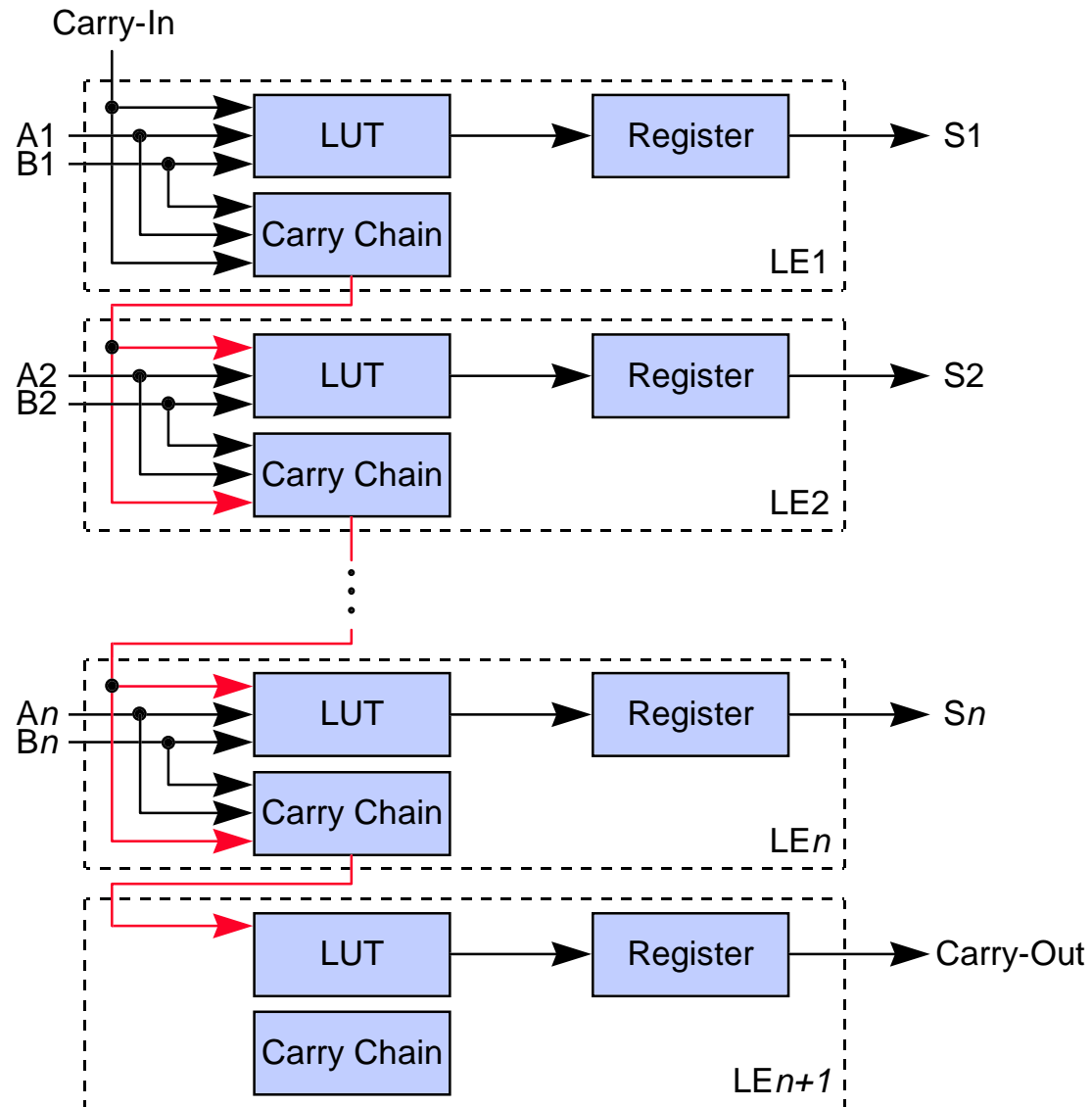
FLEX 8000A Architecture



FLEX 8000A Logic Element

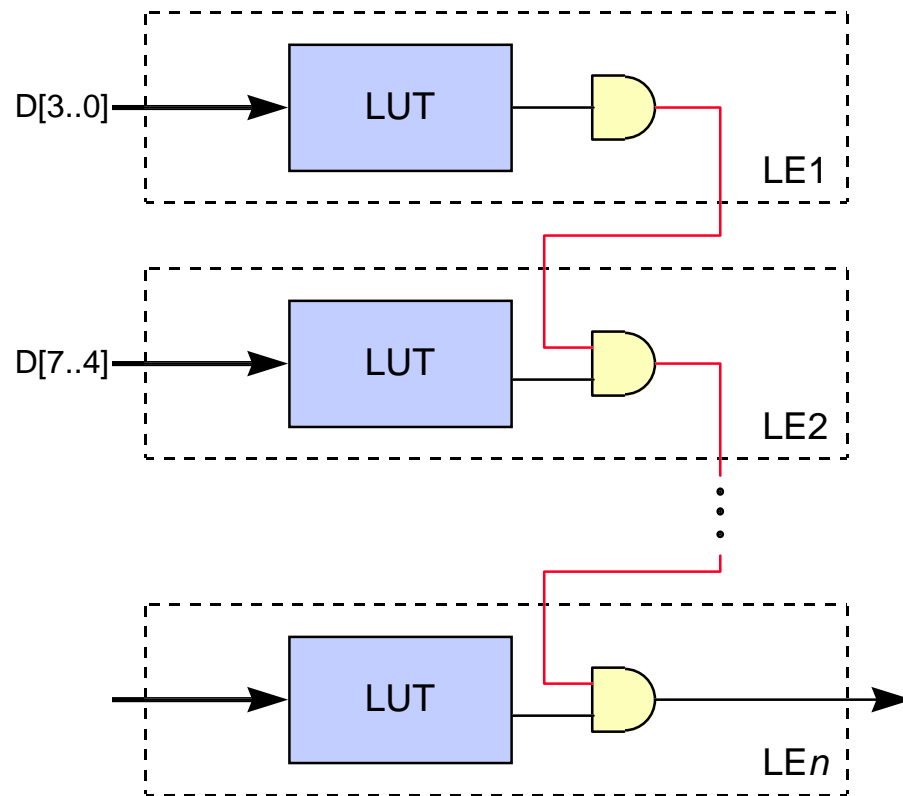


Carry Chains

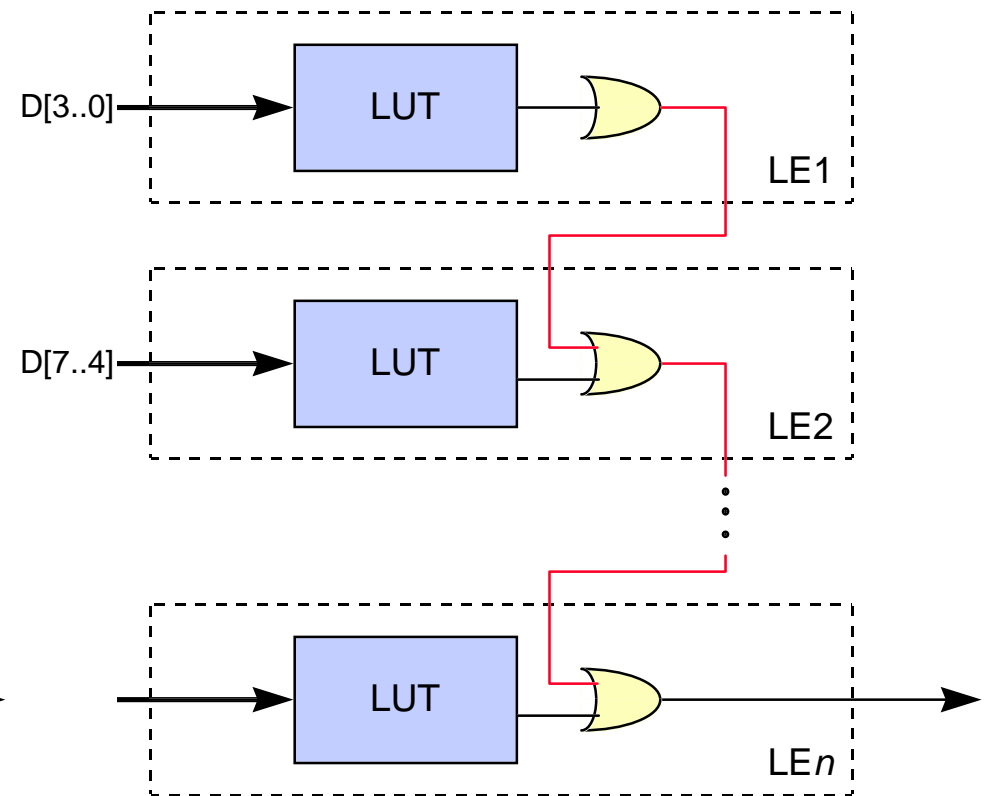


Cascade Chains

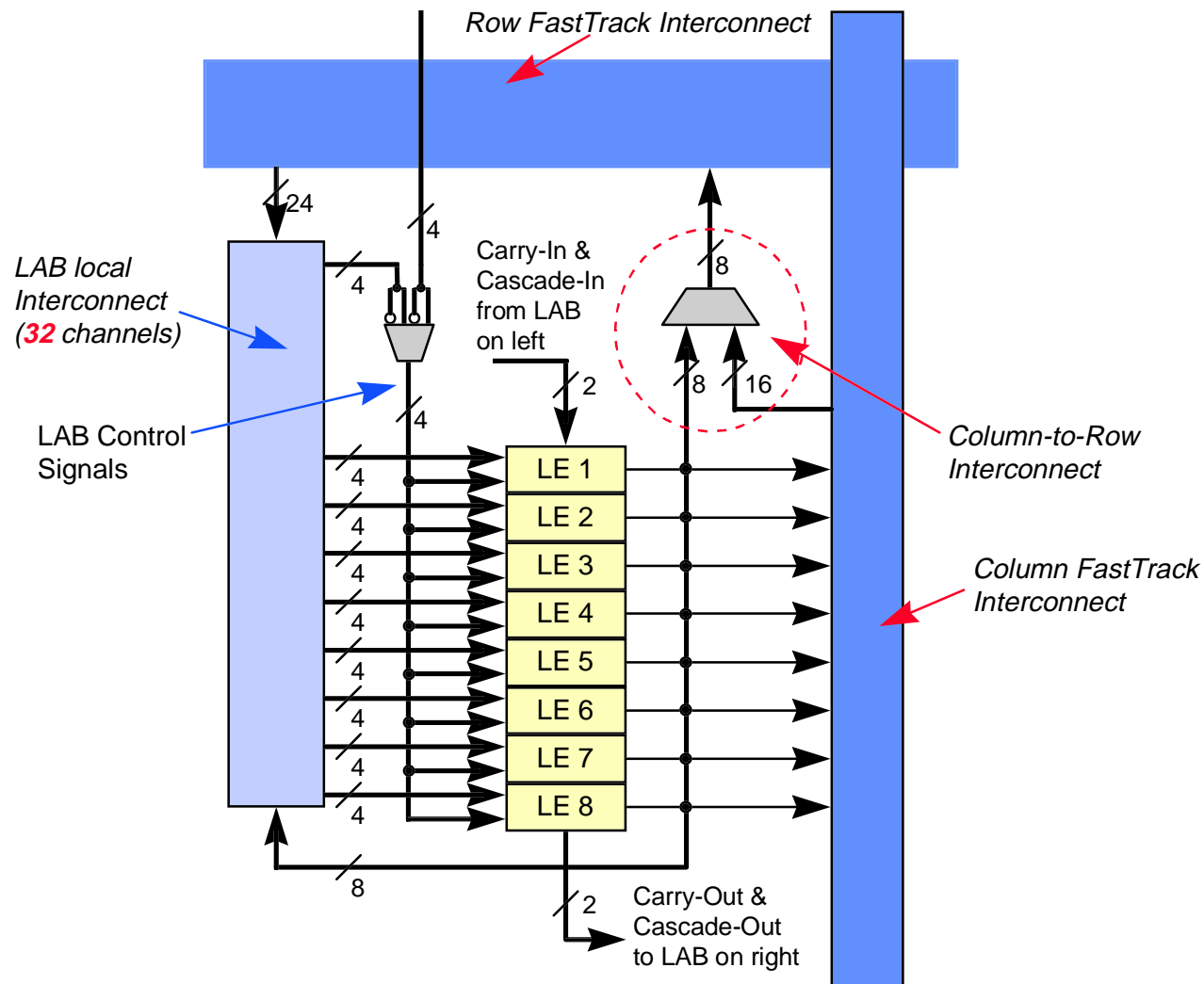
AND Cascade Chain



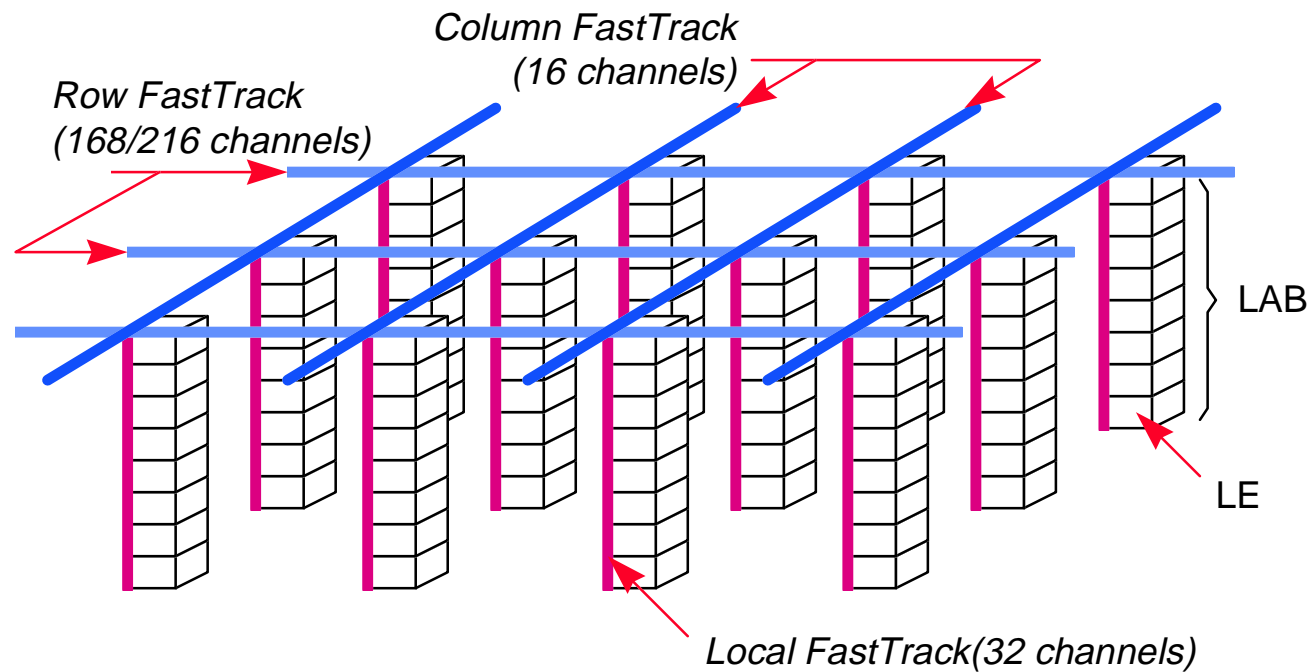
OR Cascade Chain



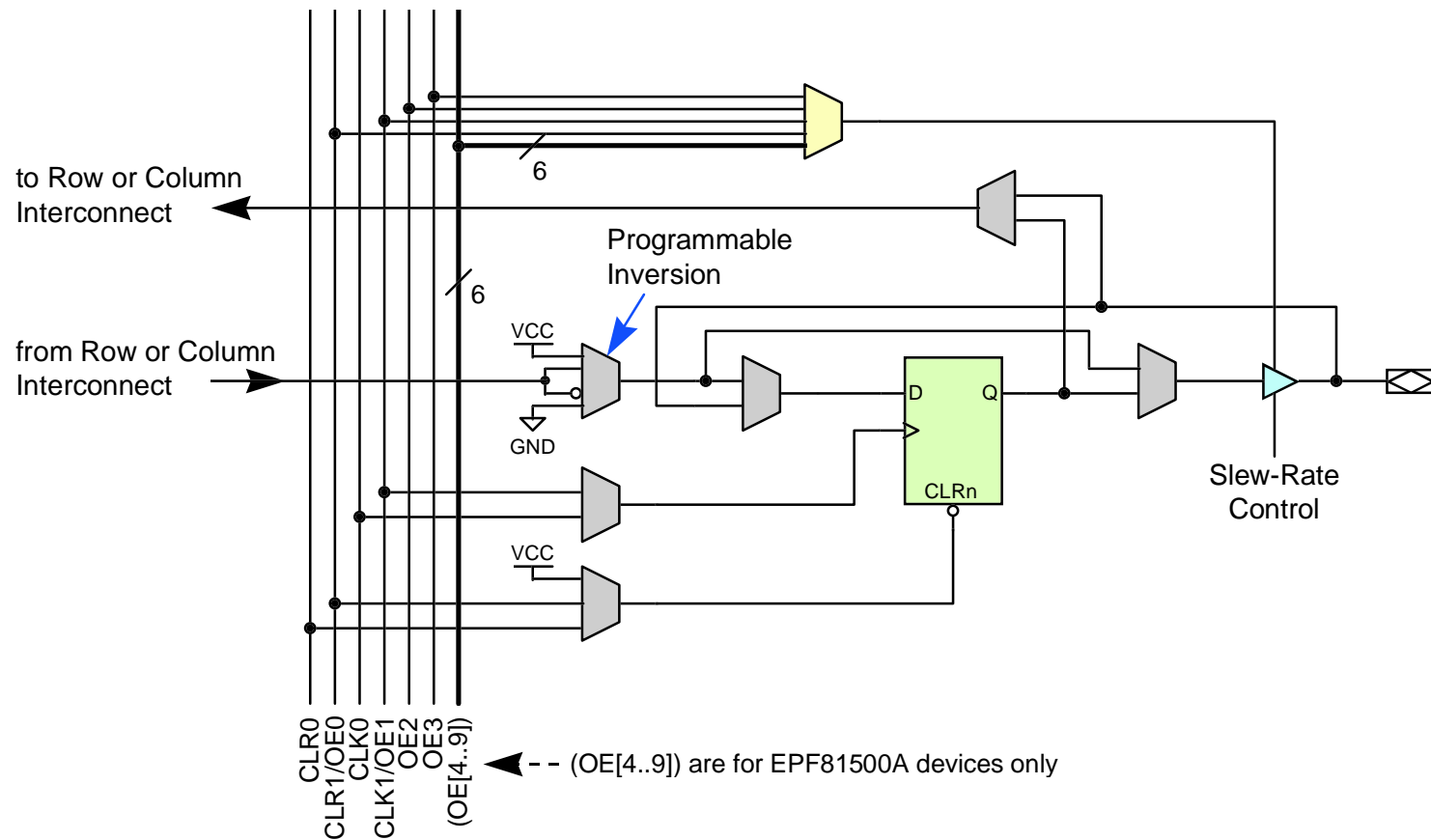
FLEX 8000A Logic Array Block



FLEX 8000A FastTrack Interconnect



FLEX 8000A I/O Element



FLEX 8000A Configuration

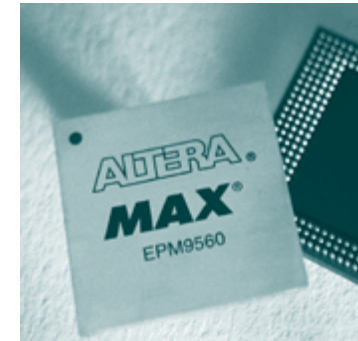
◆ Configuration schemes & data source

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

	Configuration Scheme	Data Source
AS	(Active Serial)	Serial configuration EPROM
APU	(Active Parallel Up)	Parallel EPROM
APD	(Active Parallel Down)	Parallel EPROM
PS	(Passive Serial)	Serial data path (e.g. serial download cable)
PPS	(Passive Parallel Synchronous)	Intelligent host
PPA	(Passive Parallel Asynchronous)	Intelligent host

MAX 9000 Family

◆ Today's MAX 9000 family members



Device	MCs	Gates	Speed Grade	Package Options	I/O Pins
EPM9320	320	6,000	-15,-20	PLCC84, RQFP208, PGA280, BGA356	60,132,168
EPM9400	400	8,000	-15,-20	PLCC84, RQFP208/240	59,139,159
EPM9480	480	10,000	-15,-20	RQFP208/240	146,175
EPM9560	560	12,000	-15,-20	RQFP208/240/304, PGA280, BGA356	153,191,216

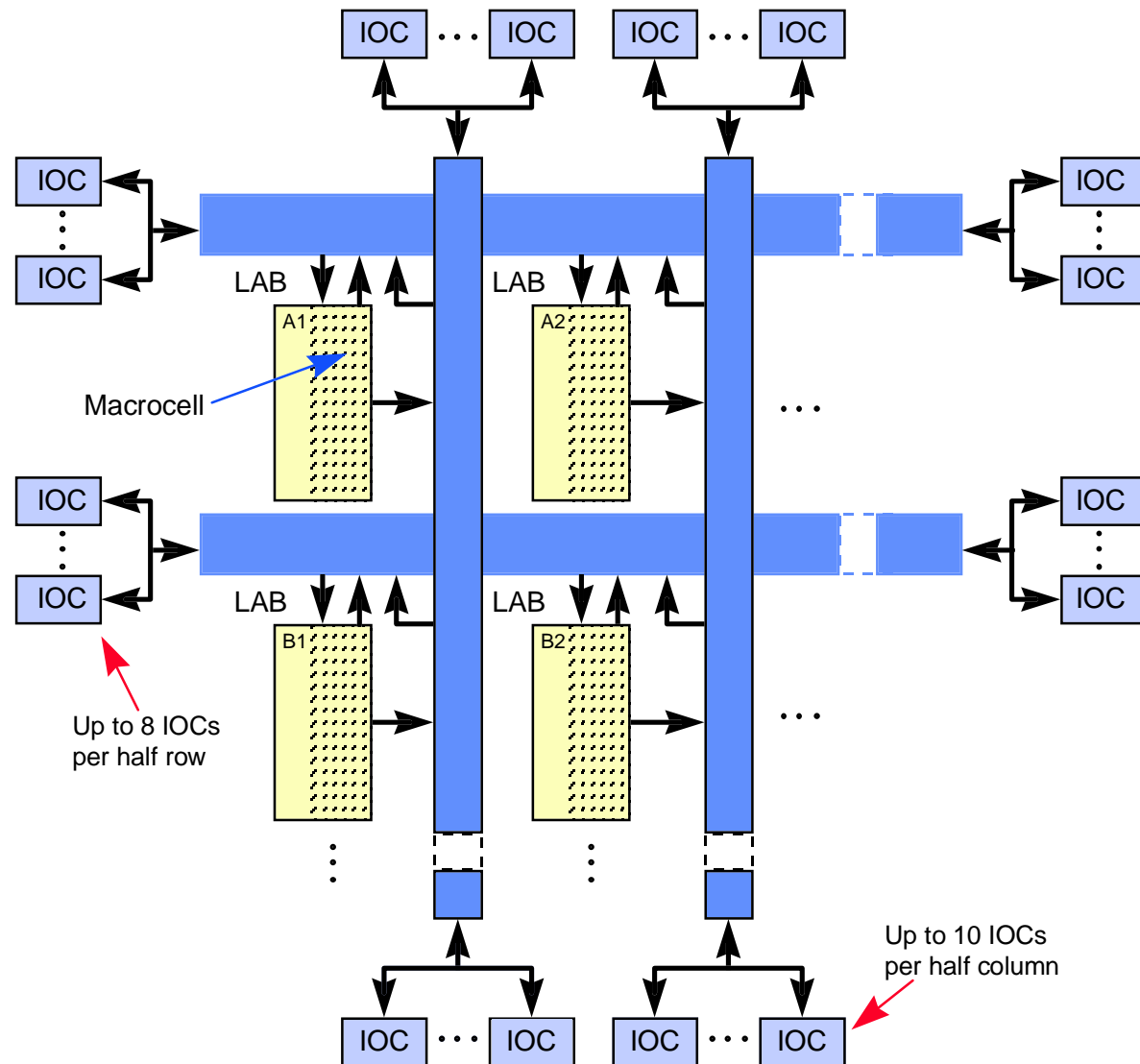
New MAX9000A Devices (not available yet) :
EPM9320A, 9400A, 9480A, 9560A

MAX 9000 Features

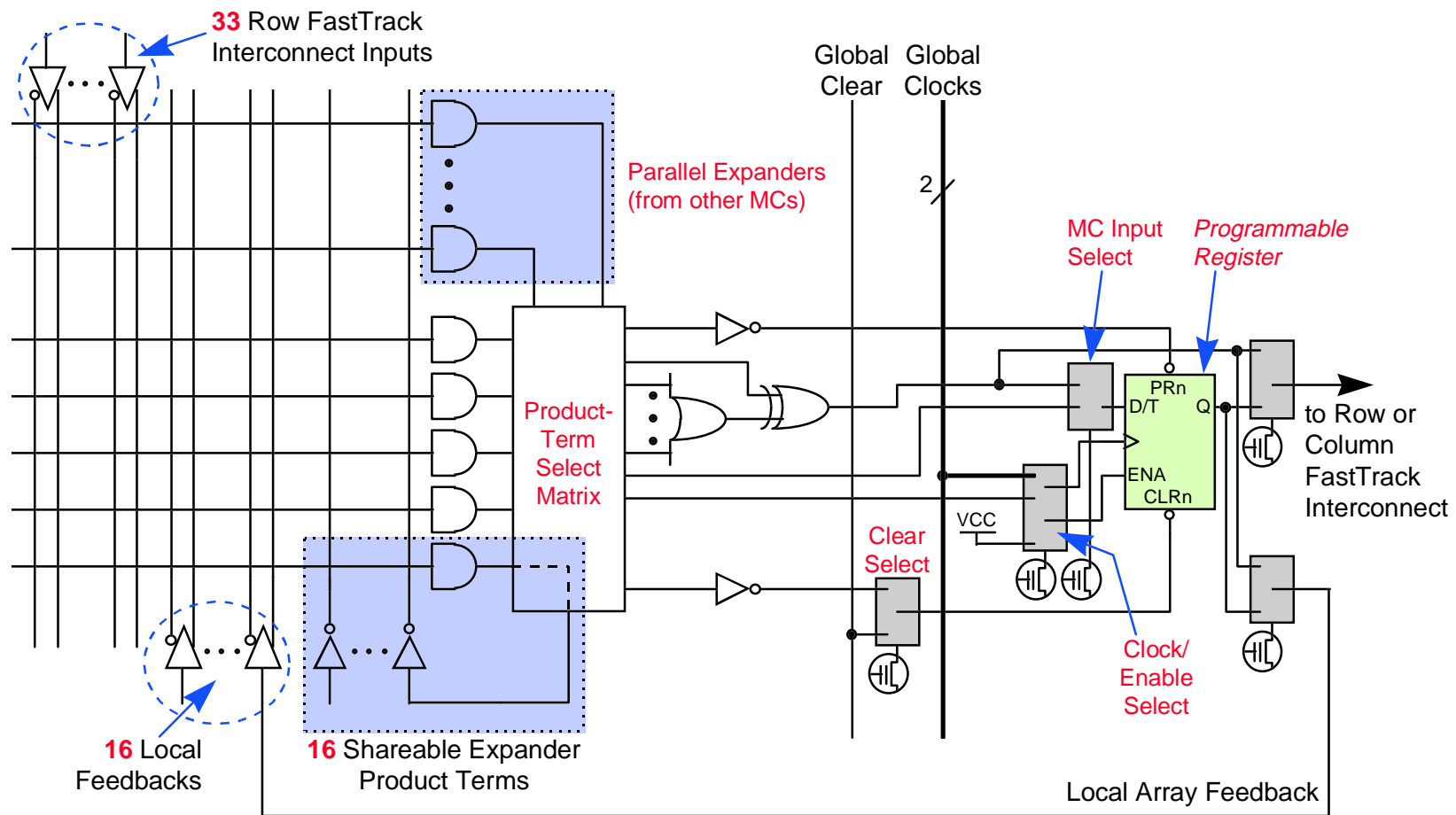
◆ MAX 9000 main features...

- EEPROM-based devices based on Altera's MAX architecture
- 320 ~ 560 macrocells
- 6,000 ~ 12,000 usable gates
- Programmable flip-flops with individual clear, preset & clock enable controls
- Dual-output macrocell structure
- 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 power-saving mode in each macrocell
- Programmable security bit
- 5-V ISP through built-in JTAG interface
- PCI-compliant -12 speed grade
- 3.3-V or 5-V I/O operation on all devices

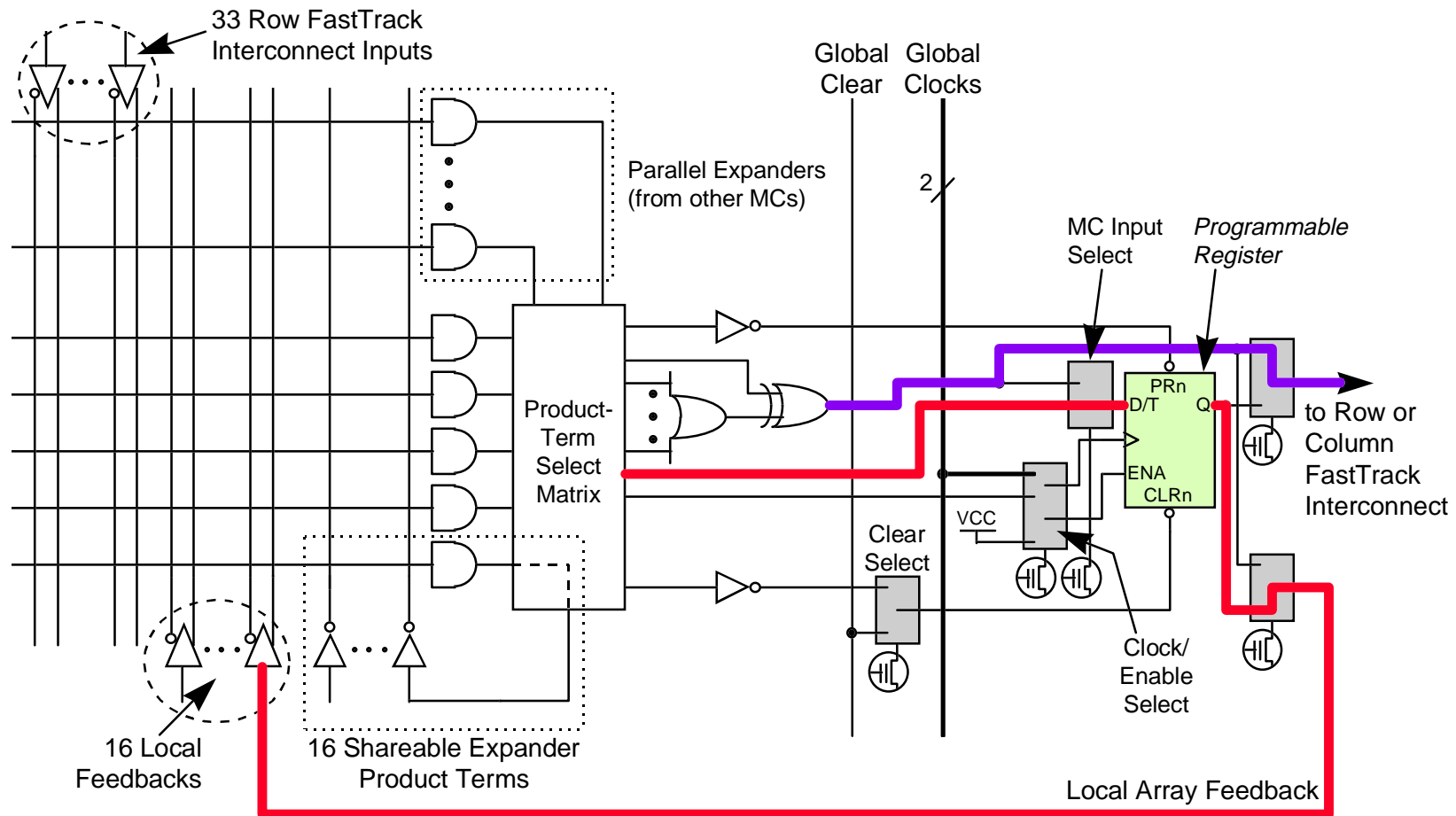
MAX 9000 Architecture



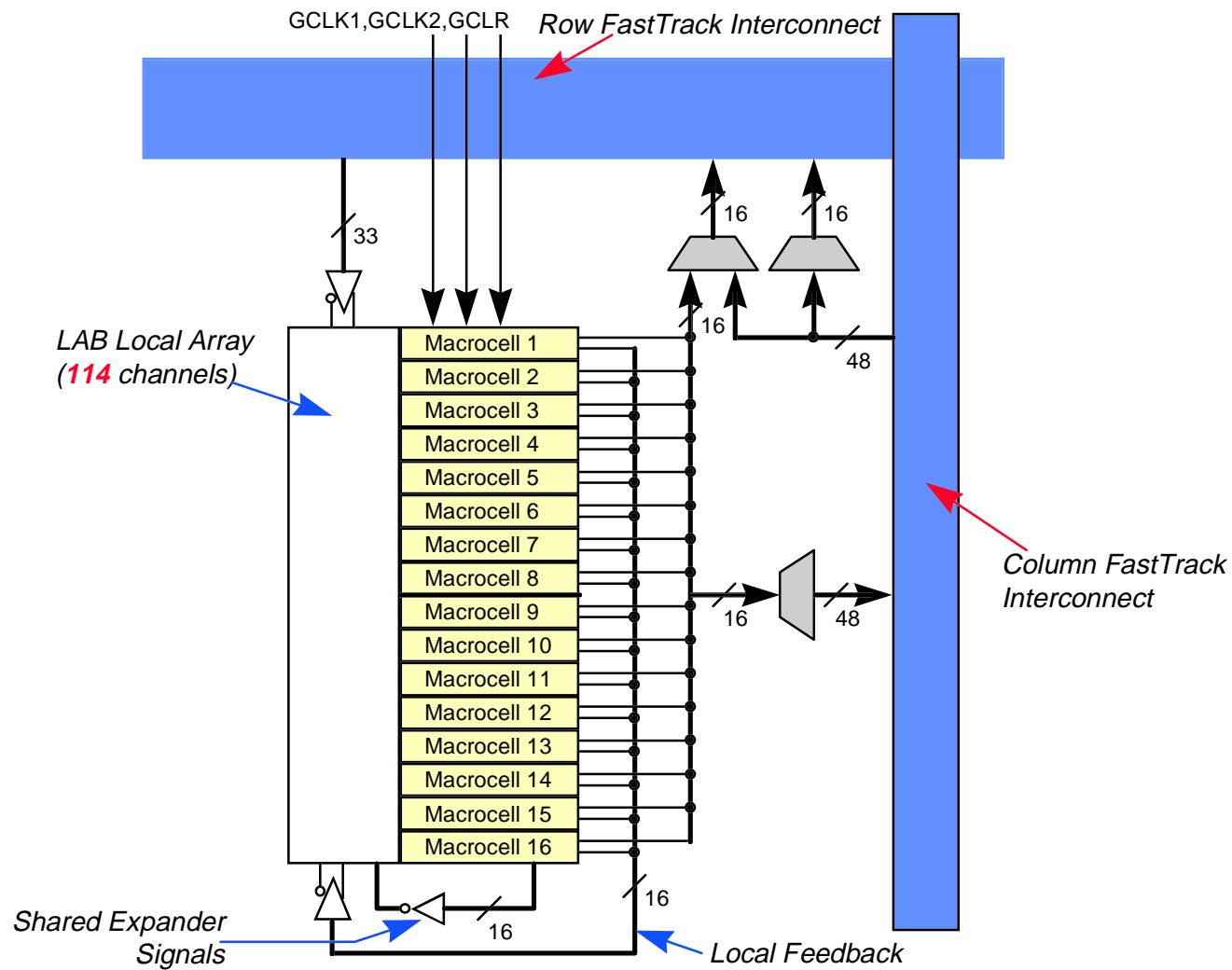
MAX 9000 Macrocell



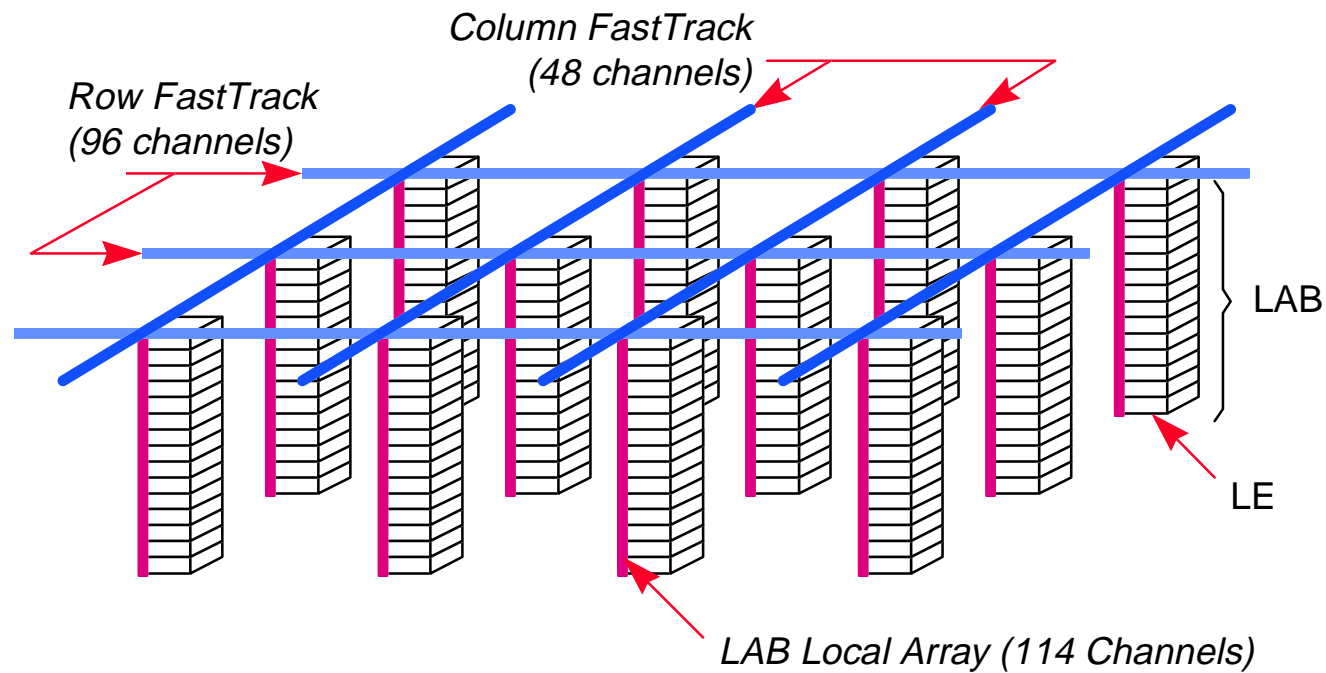
MAX 9000 Register Packing



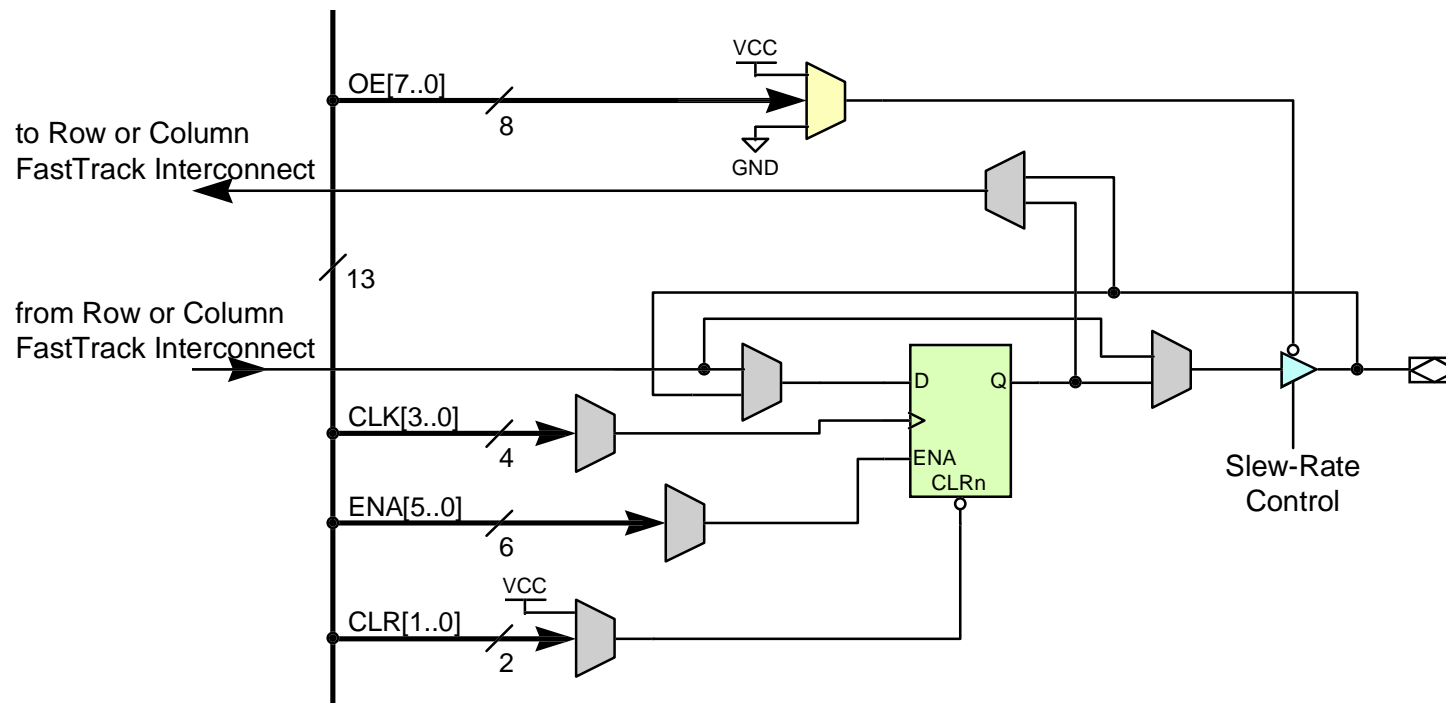
MAX 9000 Logic Array Block



MAX 9000 FastTrack Interconnect



MAX 9000 I/O Cell



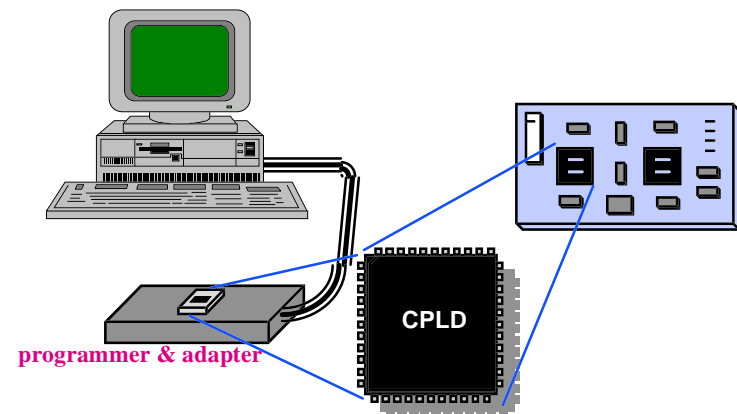
Peripheral Control Bus[12..0] : OE/ENA[4..0],OE5,OE6,OE7/CLR1,CLR0/ENA5,CLK[3..0]

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

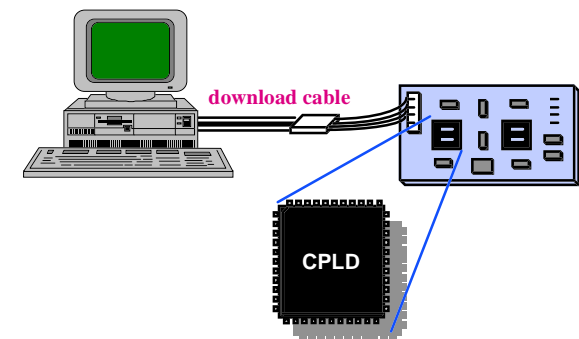
◆ MAX 9000 ISP



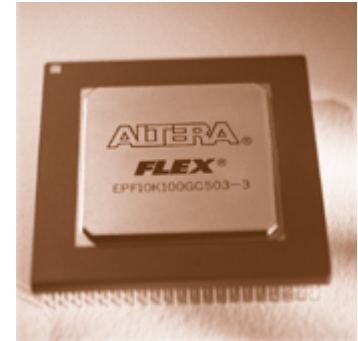
MAX 9000 ISP

◆ 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*



FLEX 10K/A Families



◆ Today's FLEX 10K/A family members

Device	Gates	EAB	LEs	FFs	Speed Grade	Package Options	I/O Pins
EPF10K10	10,000	3	576	720	-3,-4	PLCC84, TQFP144, RQFP208	59,107,134
EPF10K20	20,000	6	1,152	1,344	-3,-4	TQFP144, RQFP208 /240	107,147,189
EPF10K30	30,000	6	1,728	1,968	-3,-4	TQFP144, RQFP208 /240, BGA356	107,147,189,246
EPF10K40	40,000	8	2,304	2,576	-3,-4	RQFP208/240	147,189
EPF10K50	50,000	10	2,880	3,184	-3,-4	RQFP240, BGA356, PGA403	189,274,310
EPF10K70	70,000	9	3,744	4,096	-3,-4	RQFP240, PGA503	189,358
EPF10K100	100,000	12	4,992	5,392	-3,-4	PGA503	406
<i>Available FLEX10KA Devices</i>							
EPF10K50V	50,000	10	2,880	3,184	-3,-4	RQFP240, BGA356	189,274
EPF10K130V	130,000	16	6,656	7,126	-3,-4	BGA596, PGA599	470

FLEX 10K Features

◆ FLEX 10K/A main features...

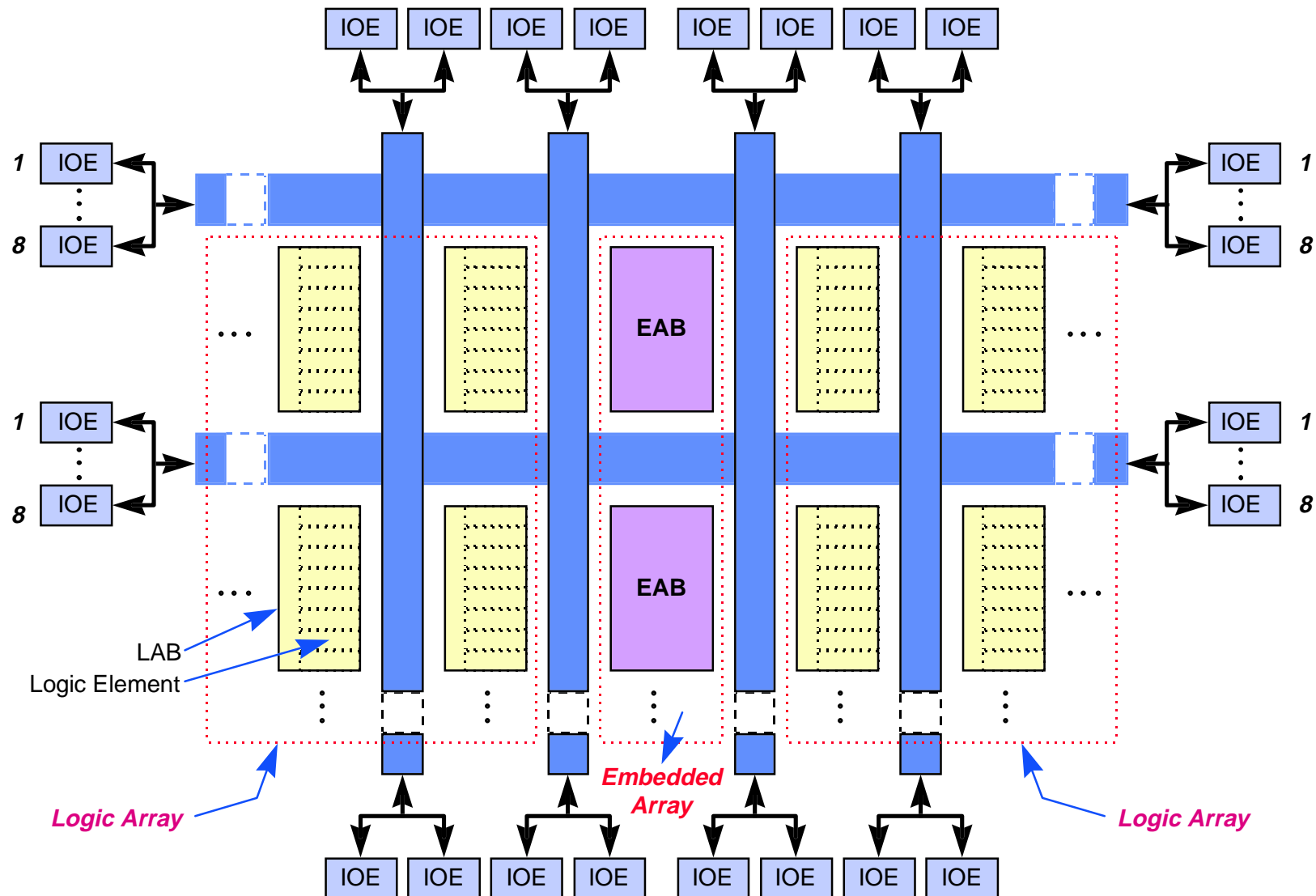
- **SRAM-based** devices based on Altera's FLEX architecture
- Embedded programmable logic family
 - **Embedded array** for implementing RAMs & specialized logic functions
 - Logic array for general logic functions
- High density
 - 10,000 ~ 100,000 typical gates (logic & RAMs)
 - 720 ~ 5,392 registers
 - 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)

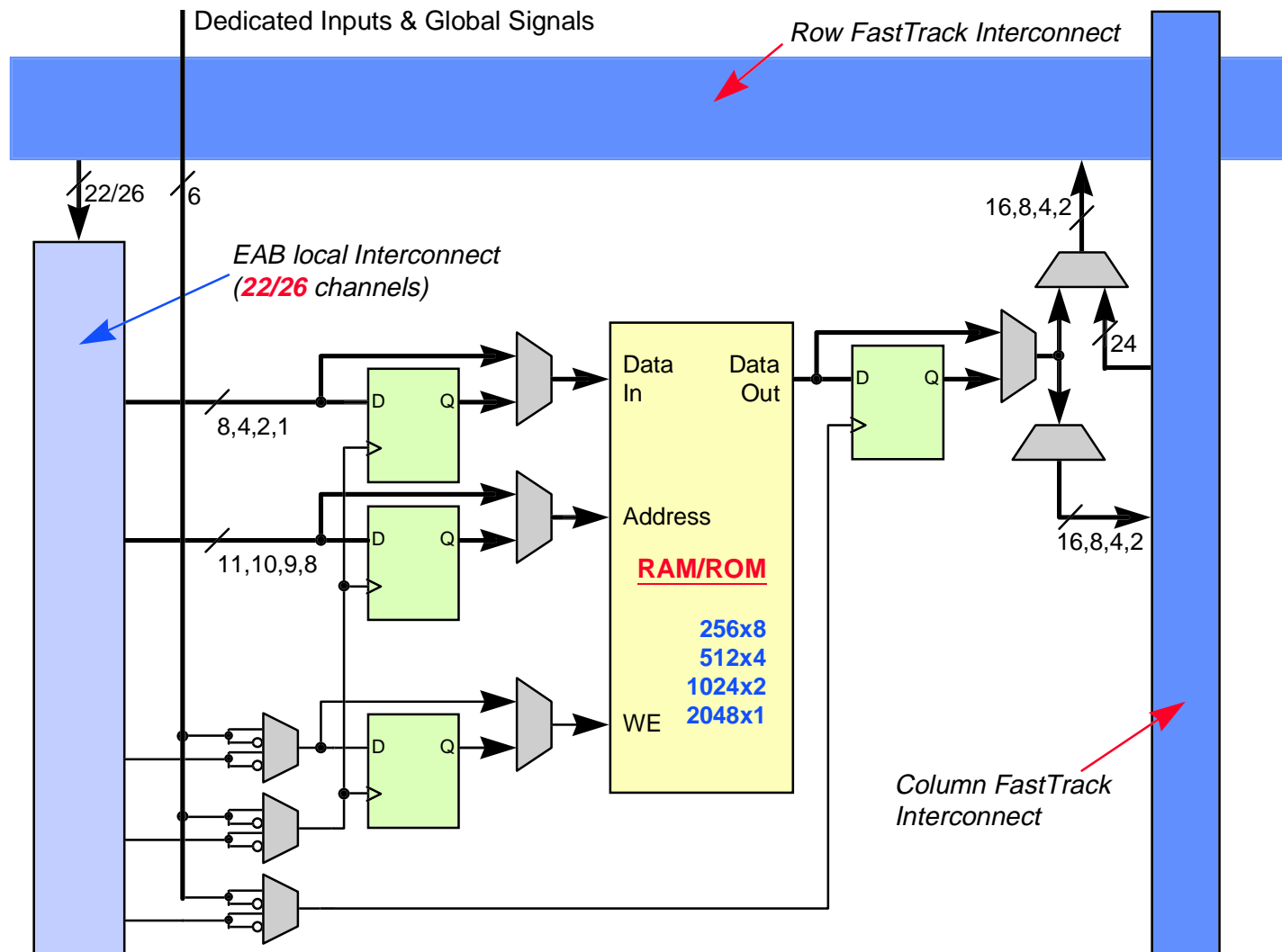
◆ FLEX 10K main features... (continued)

- 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

FLEX 10K Architecture



FLEX 10K Embedded Array Block



What is the EAB?

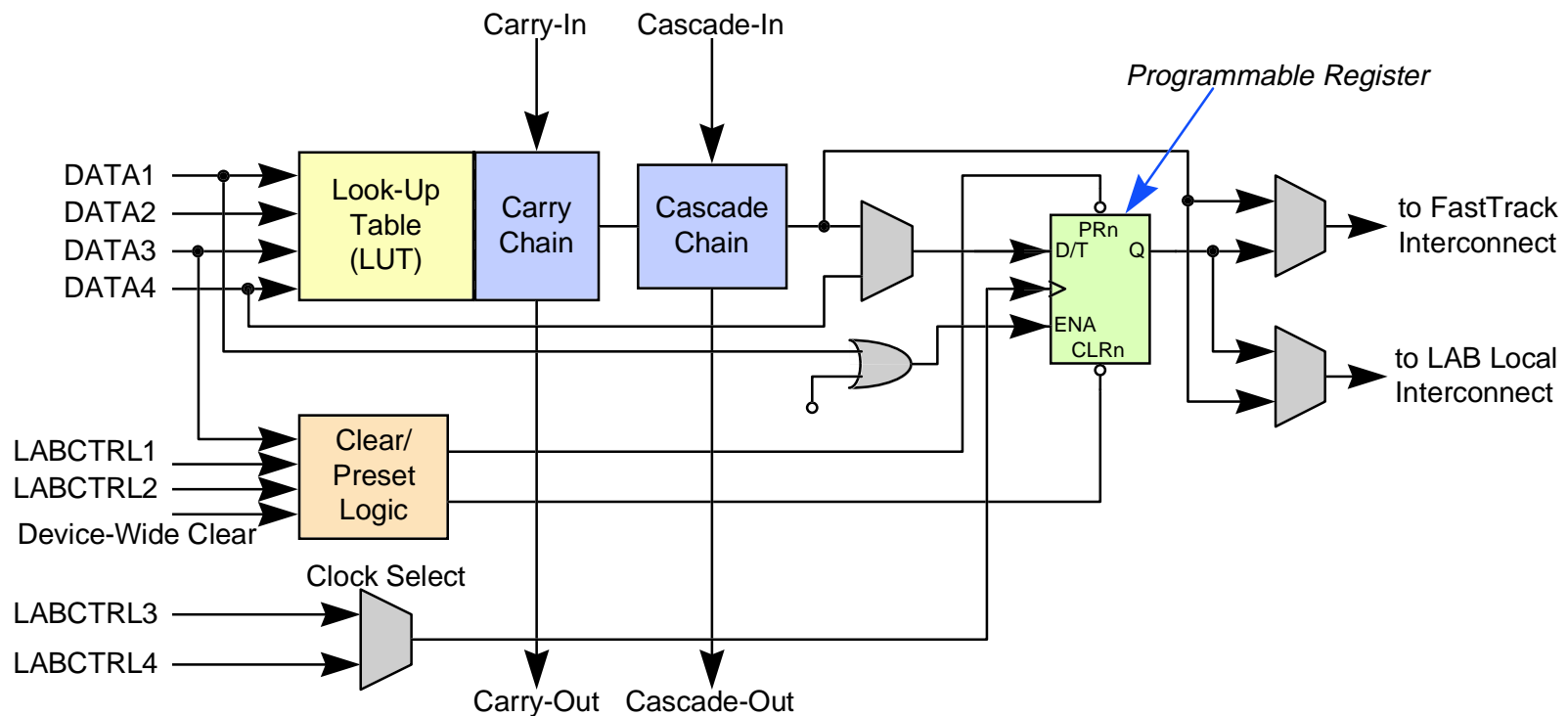
◆ What is the EAB?

- 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

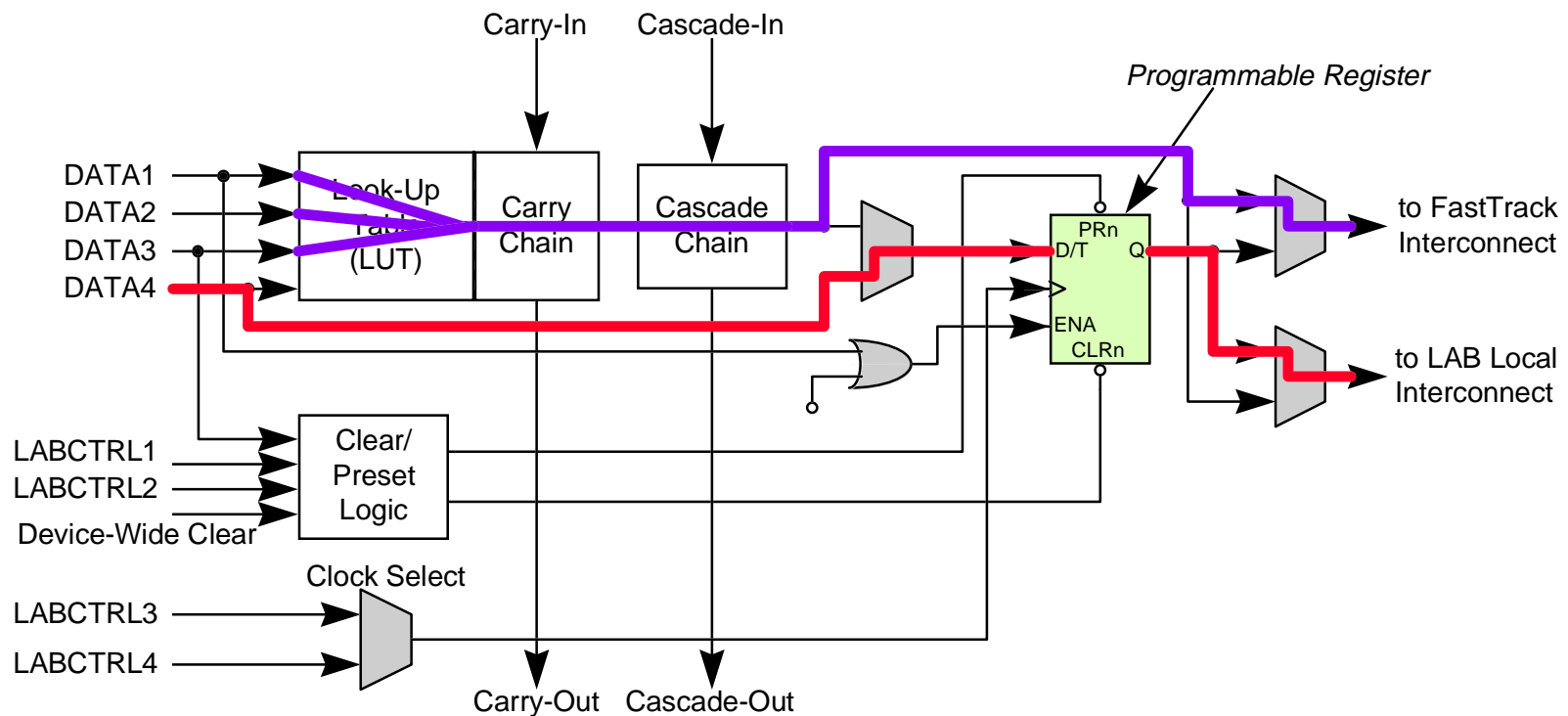
◆ EAB as logic

- 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)

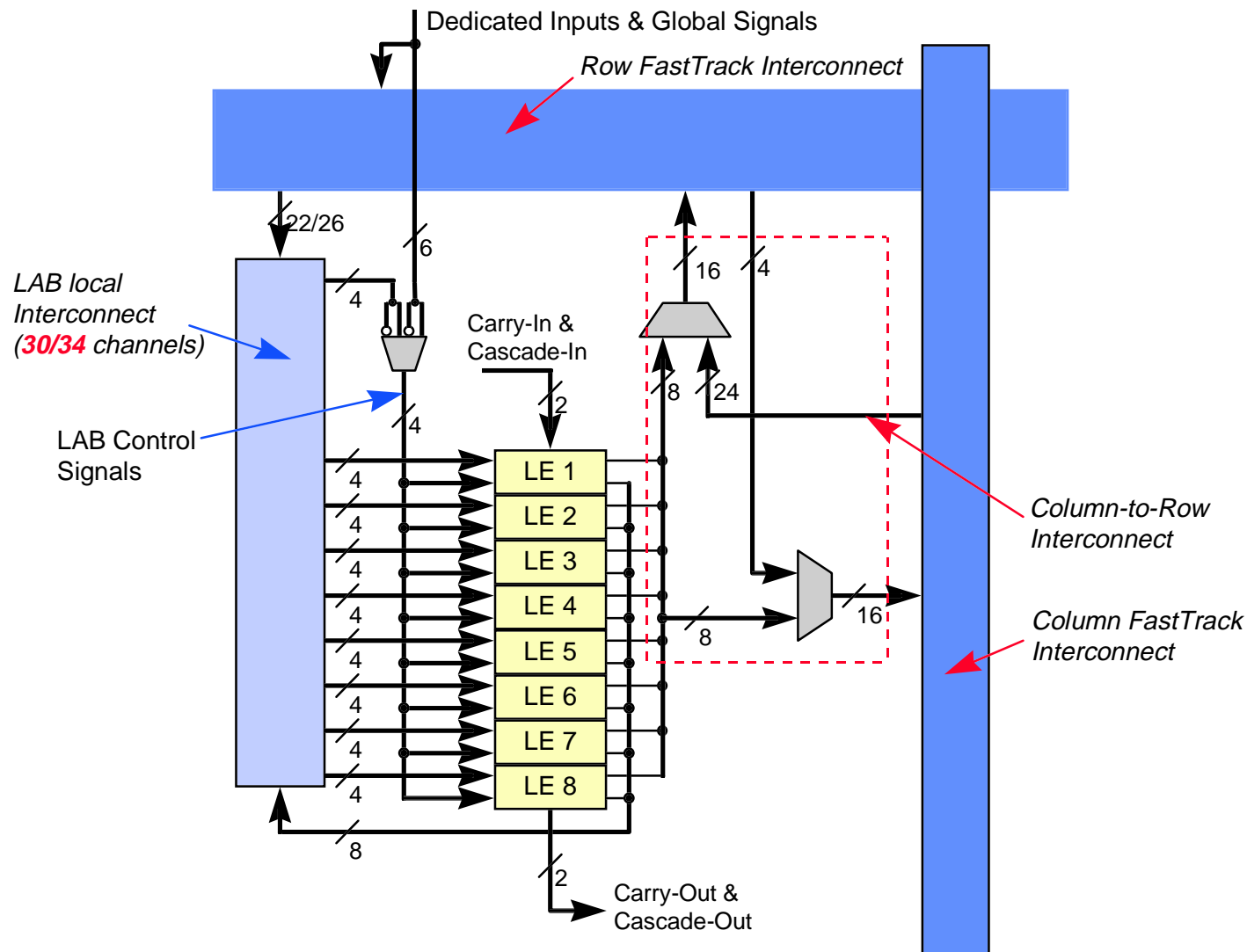
FLEX 10K Logic Element



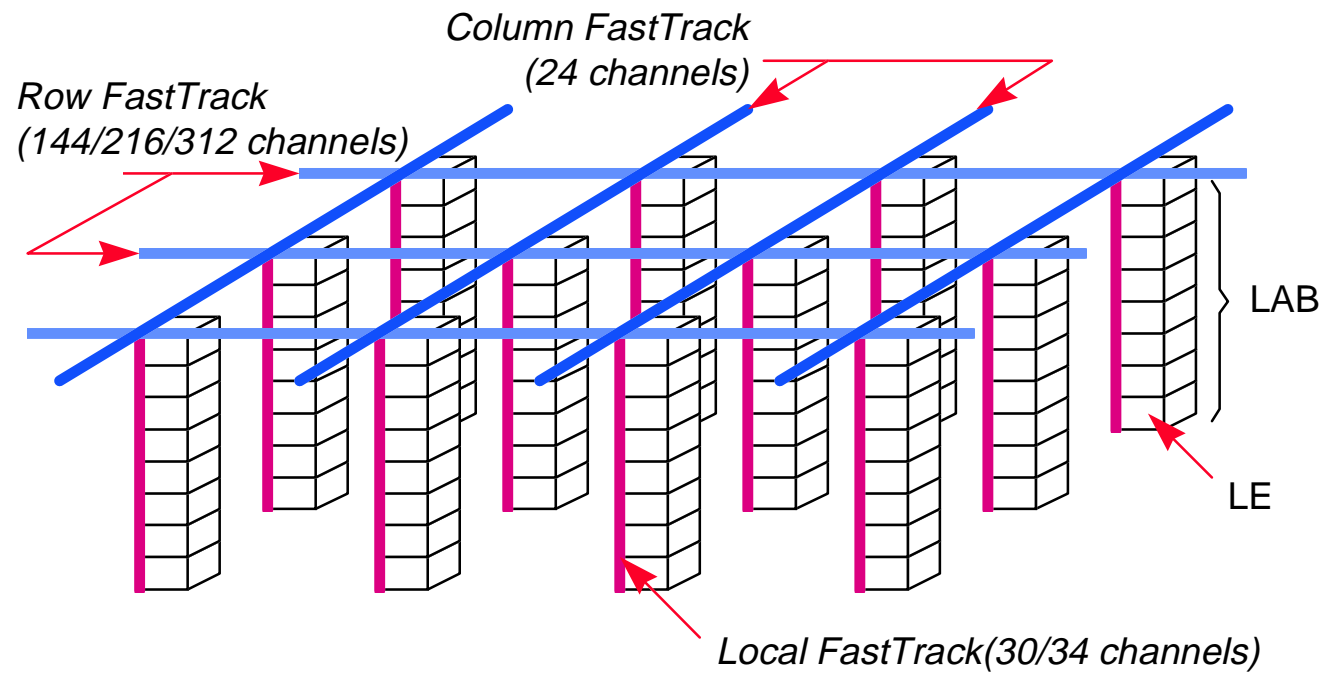
FLEX 10K Register Packing



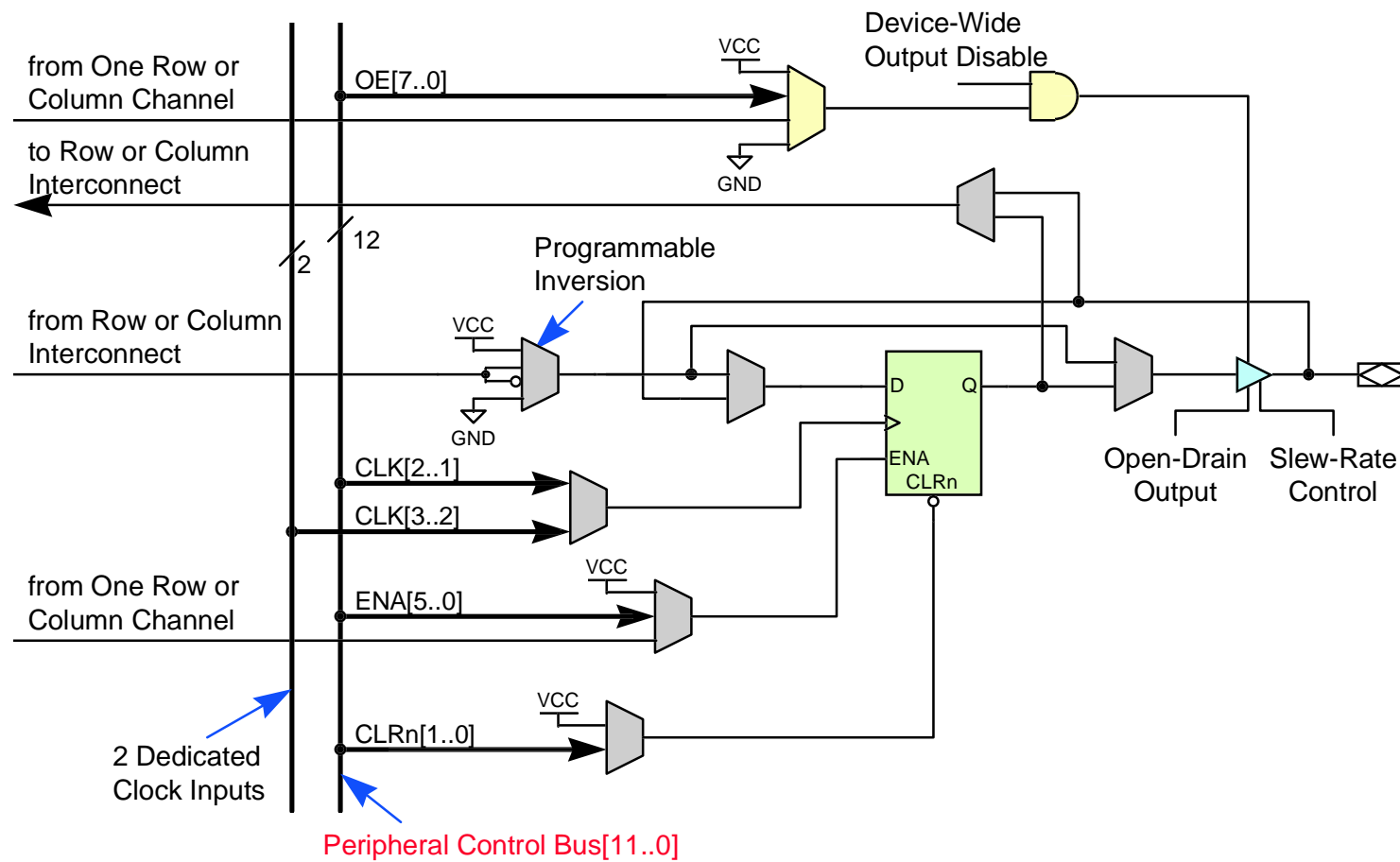
FLEX 10K Logic Array Block



FLEX 10K FastTrack Interconnect



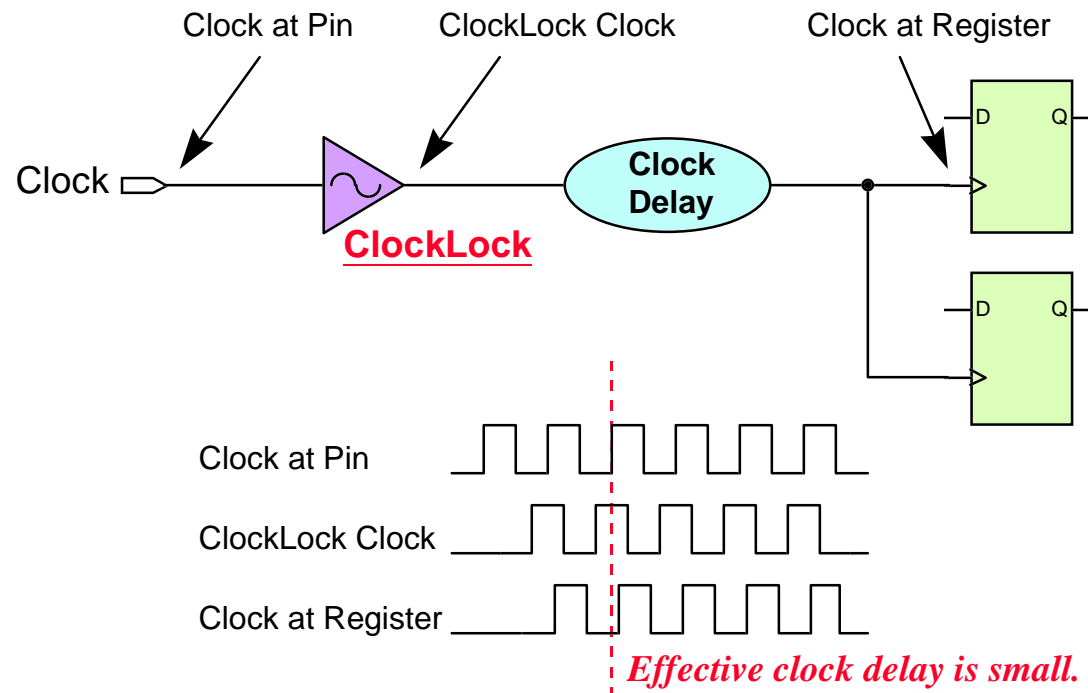
FLEX 10K I/O Element



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



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.

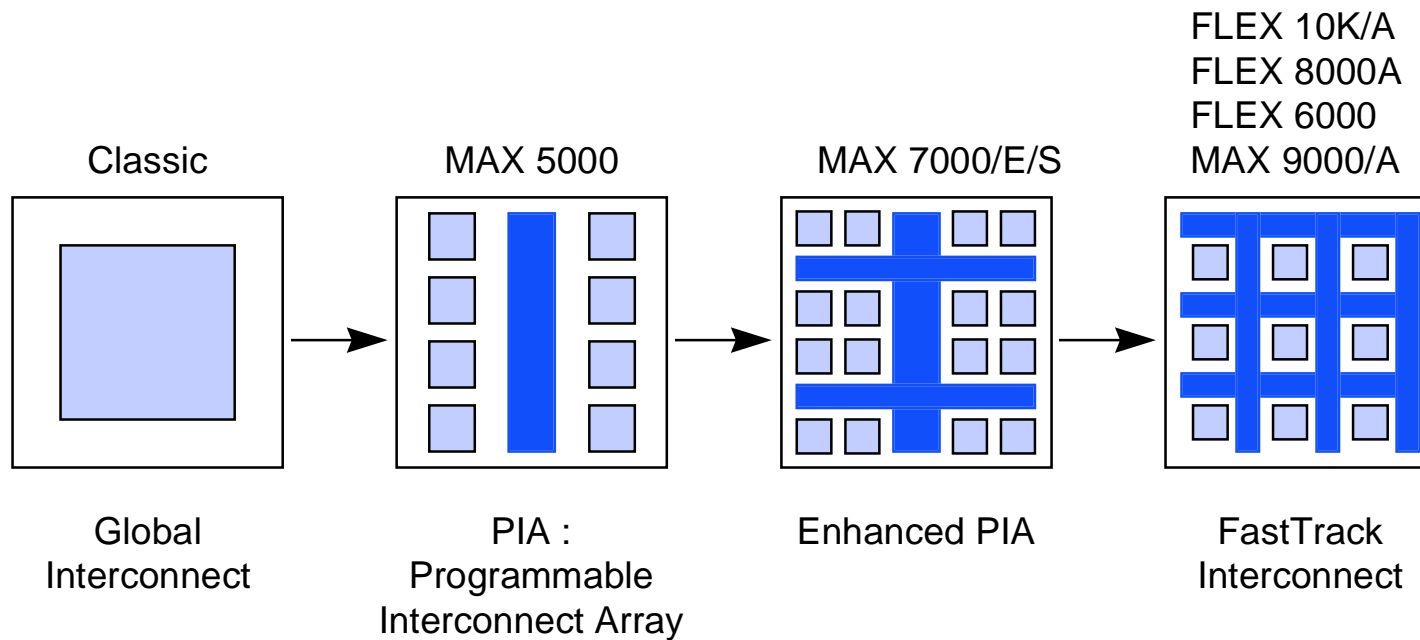
FLEX 10K Configuration

◆ Configuration schemes & data source

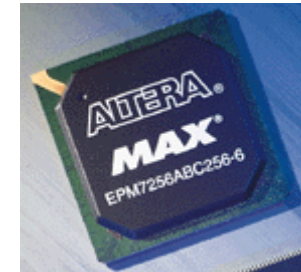
- Refer to Altera's *Application Notes* for details
 - AN059: *Configuring FLEX 10K Devices*
 - AN039: *JTAG Boundary-Scan Testing in Altera Devices*

Configuration Scheme	Data Source
PS (Passive Serial)	Altera's EPC1 configuration EPROM, BitBlaster or ByteBlaster download cable, serial data source
PPS (Passive Parallel Synchronous)	Intelligent host, parallel data source
PPA (Passive Parallel Asynchronous)	Intelligent host, parallel data source
JTAG	JTAG controller

Altera Architecture Evolution



MAX Roadmap

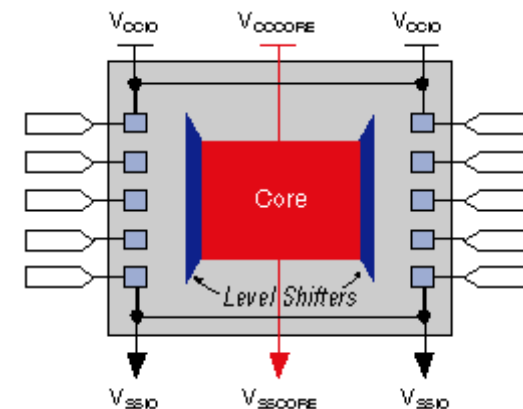


◆ MAX 7000A device family

- 3.3-V, 0.35um process
- 32 ~ 1,024 macrocells (600 ~ 20,000 useable gates)
 - 7032A, 7064A, 7128A, 7256A, 7384A, 7512A and 71024A
- “MultiVolt” I/O interface enabling device core to run at 3.3V, while I/O pins are compatible with 5.0-V, 3.3-V, and 2.5-V logic levels
- ClockBoost feature

◆ MAX 9000A device family

- 5-V, 0.5um process
- MultiVolt I/O interface supports 5-V and 3.3-V
- 7.5ns t_{pd}



FLEX Roadmap



◆ FLEX 10KA & 10KB device family

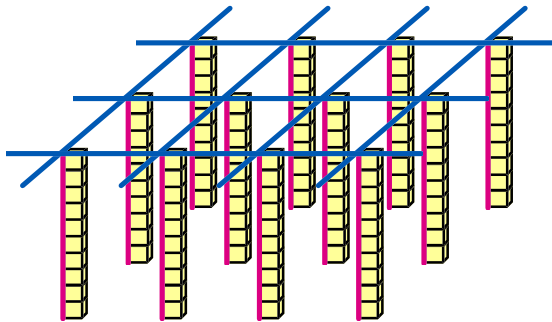
- FLEX 10KA: 3.3-V, 0.35um process
 - 10K10A, 10K30A, 10K50V, 10K100A, 10K130V, and 10K250A
- FLEX 10KB: 2.5-V, 0.25um process
 - 10K30B, 10K50B, 10K100B, 10K130B, 10K180B, and 10K250B
- Up to 250,000 gates (EPF10K250A/B: 12,160 LEs and 20 EABs)
- "MultiVolt" I/O interface

◆ FLEX 6000 & 6000A device family

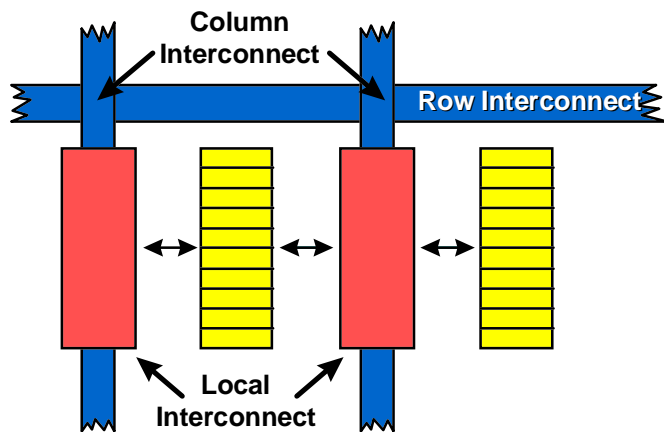
- FLEX 6000: 5-V, 0.5um process ; FLEX 6000A: 3.3-V, 0.35um process
 - 6016, 6016A and 6024A
- Up to 24,000 gates (EPF6024A: 1,960 LEs)
- "OptiFLEX" architecture, advanced bond pad technology, interleaved LABs, and an optimized I/O structure to increase the level of programmable logic efficiency
- "MultiVolt" I/O interface
- Low-cost, high volume solution for gate array designers

Appendix: FLEX 6000 Architecture

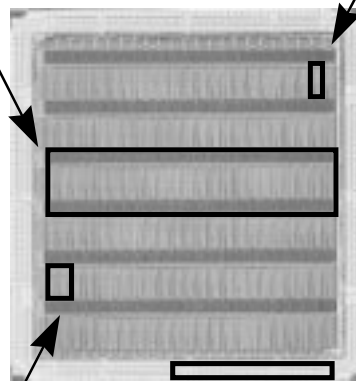
FastTrack™ Interconnect



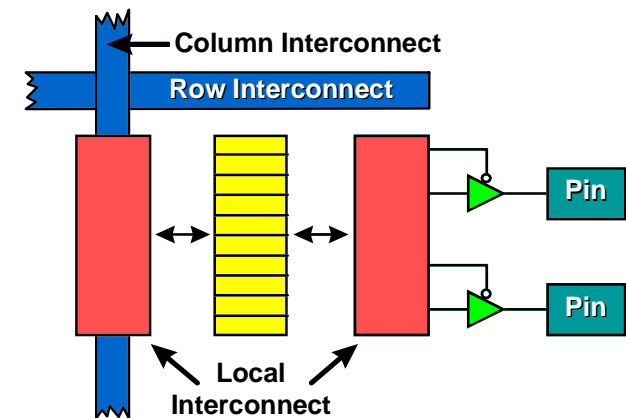
Interleaved LABs



FLEX 6000 Die

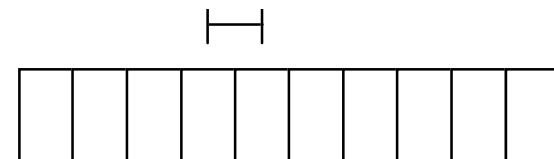


FastFLEX™ I/O



μPitch™ Technology

3.2 mil (81 μm)



Bond Pads

Design Flow & Altera Tools

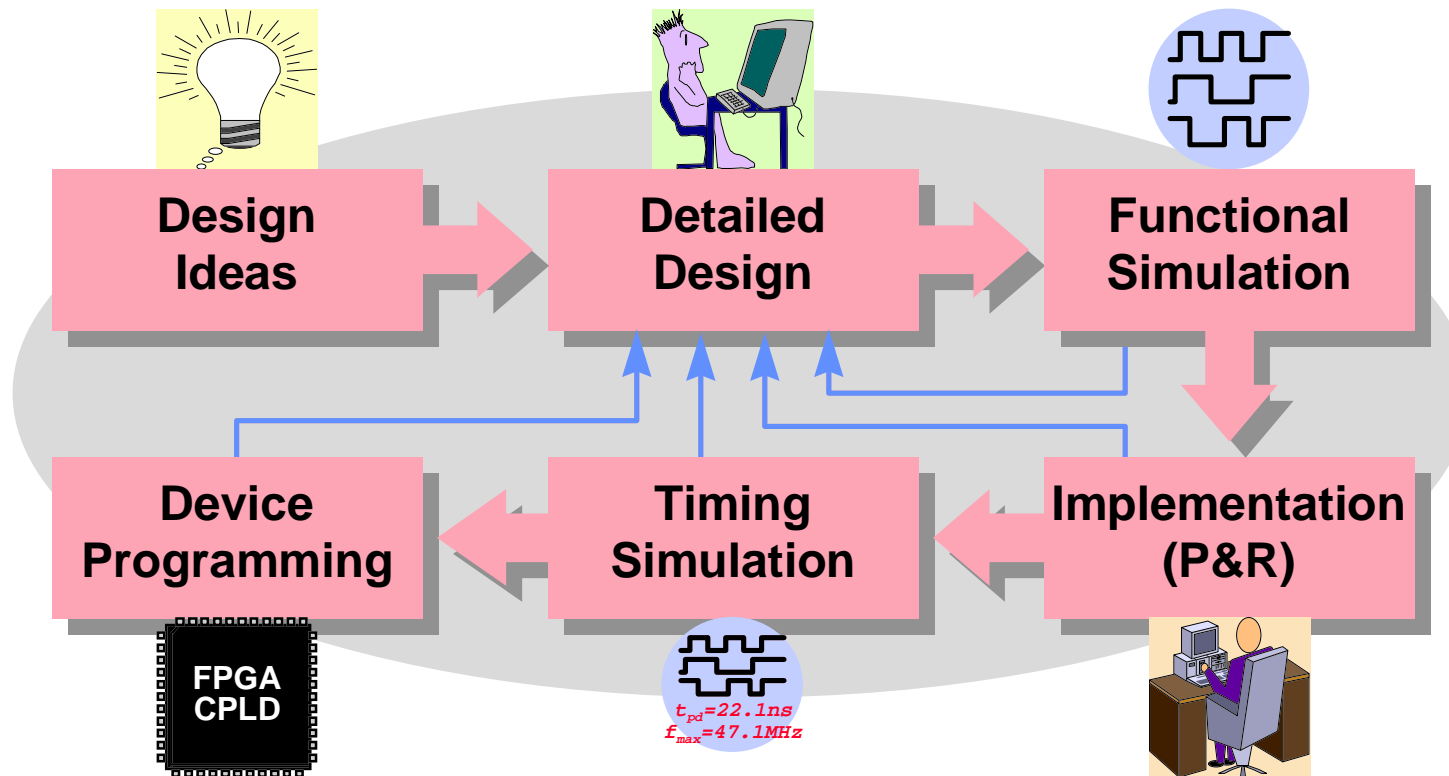
◆ FPGA/CPLD Design Flow

- Design Ideas
- Detailed Design
- Functional Simulation
- Synthesis & Implementation
- Timing Simulation
- Device Programming

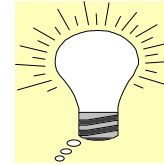
◆ Altera MAX+PLUS II Development Software

- Design Entry
- Project Processing
- Project Verification
- Device Programming

FPGA/CPLD Design Flow



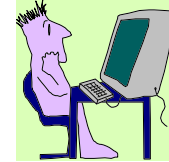
Design Ideas



◆ 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



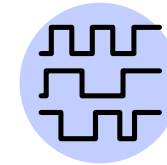
◆ Choose the design entry method

- 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

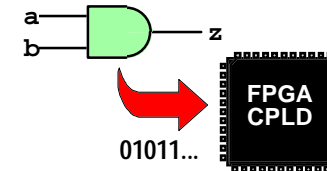
◆ Simulation results

- 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

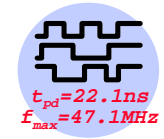
◆ Implementation results

- 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

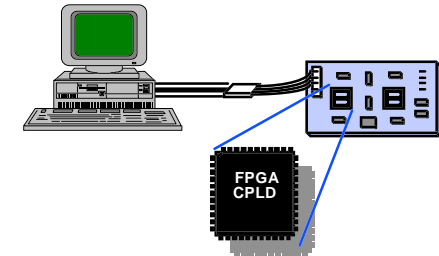
◆ Timing simulation

- 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



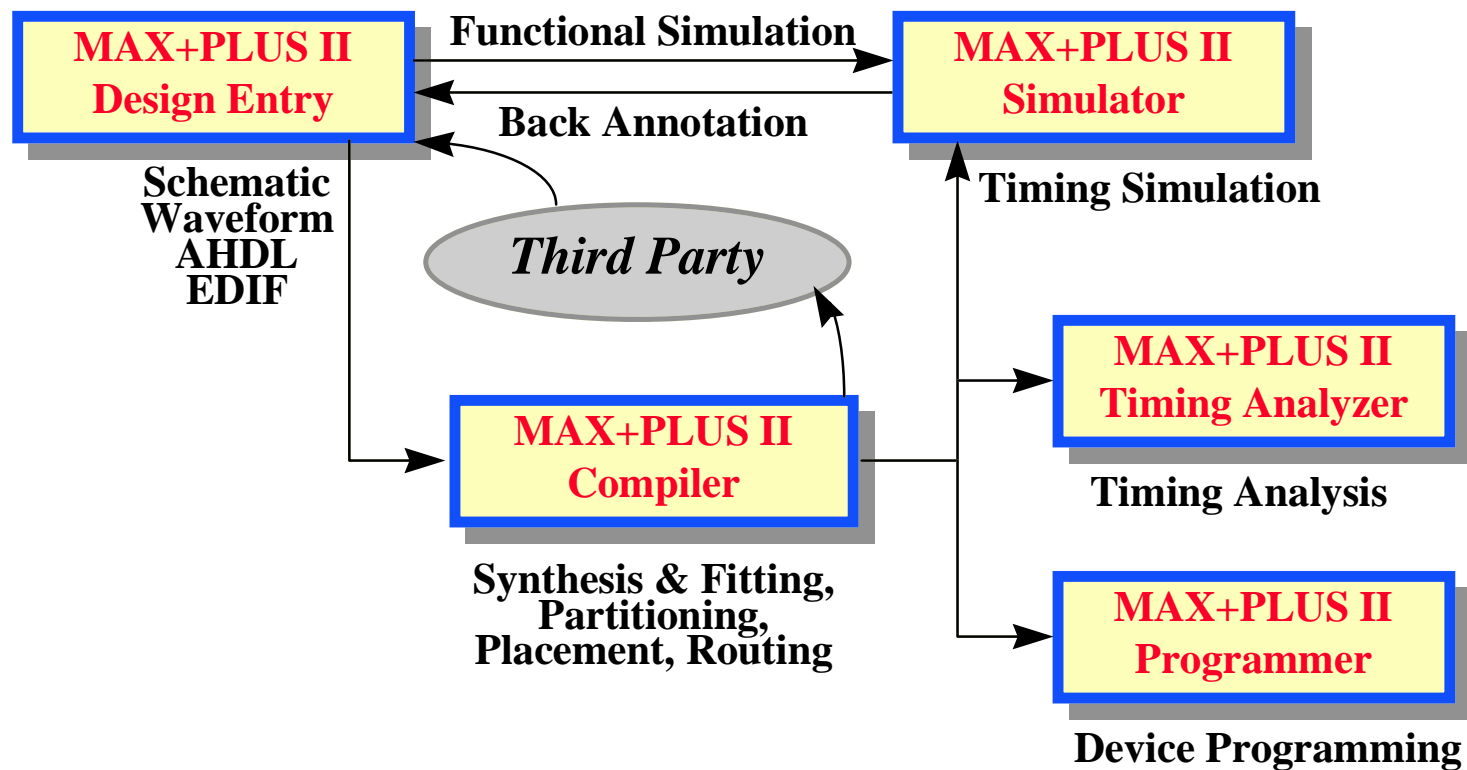
◆ Finish the board design

◆ Program the device

◆ Challenge

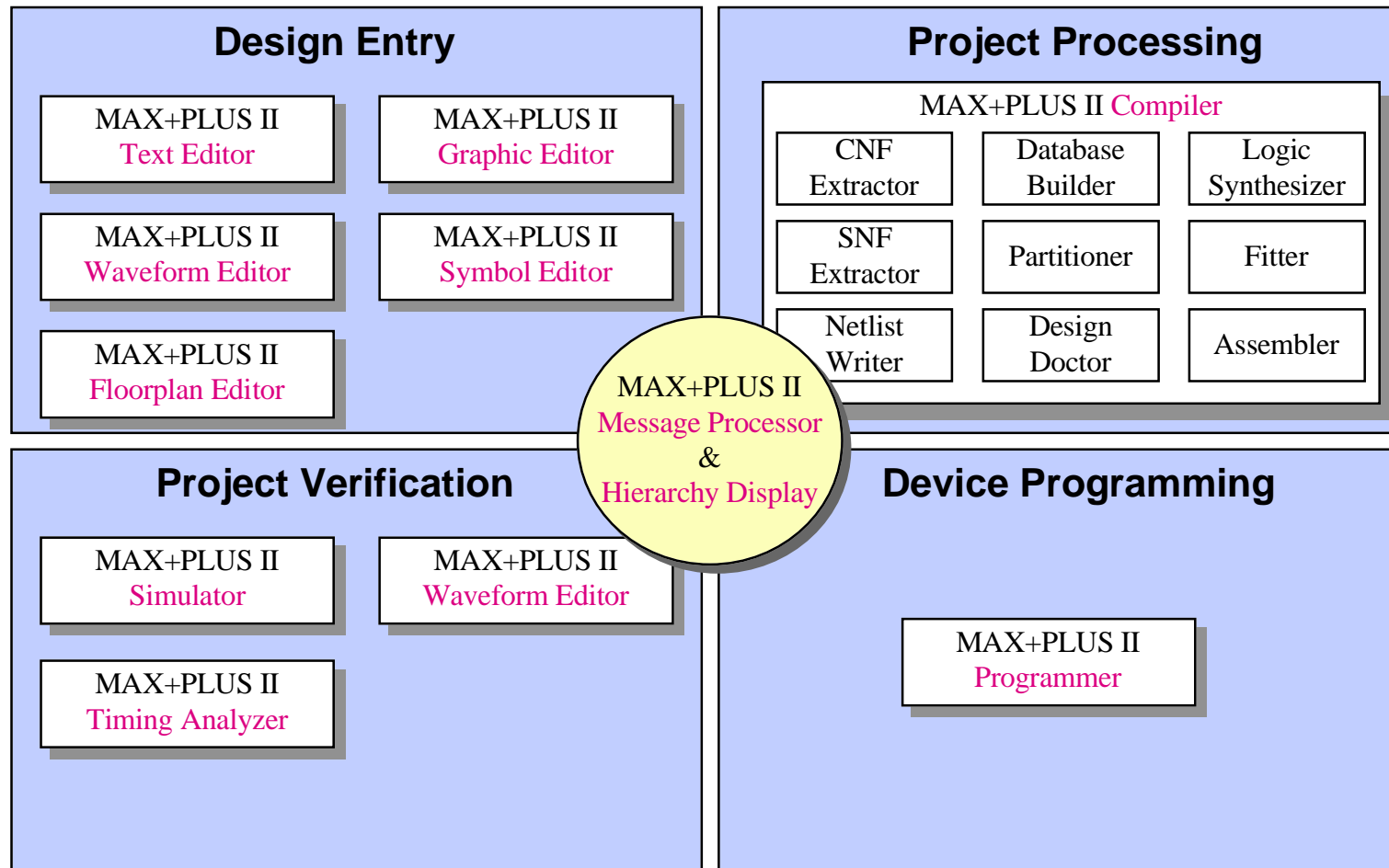
- Board design
- System considerations

Altera Design Flow - For PC Users



MAX+PLUS II

Altera's Fully-Integrated Development System



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

The screenshot displays the MAX+PLUS II Design Entry environment with four main windows:

- chiptrip.gdf - Graphic Editor:** Shows a logic diagram with components **AUTO_MAX**, **TIME_CNT**, and **TICK_C**. Inputs include **dir[1..0]**, **accel**, **clock**, **reset**, and **enable**. Outputs include **get_ticket1**, **get_ticket2**, and **speed**.
- speed_ch.wdf - Waveform Editor:** Displays a timing diagram for signals **accel_in**, **reset**, **clk**, **speed**, and **get_ticket**. The **speed** signal is shown as a sequence of "legal" states. The time scale is set to 185.0ns, and a specific value of 600.0ns is indicated.
- chiptrip.sym - Symbol Editor:** Shows a symbol for the **chiptrip** component with inputs **DIR[1..0]**, **ACCEL**, **CLOCK**, **RESET**, and **ENABLE**, and outputs **TIME[7..0]**, **AT_ALTE**, and **TICKET[3..0]**.
- time_cnt.tdf - Text Editor:** Contains the VHDL code for the **time_cnt** subdesign:


```

SUBDESIGN time_cnt
(
    enable, clk      : INPUT;
    time[7..0]      : OUTPUT;
)
VARIABLE
    count[7..0]     : DFF;
BEGIN
    count[0].clk = clk;
    time[] = count[];

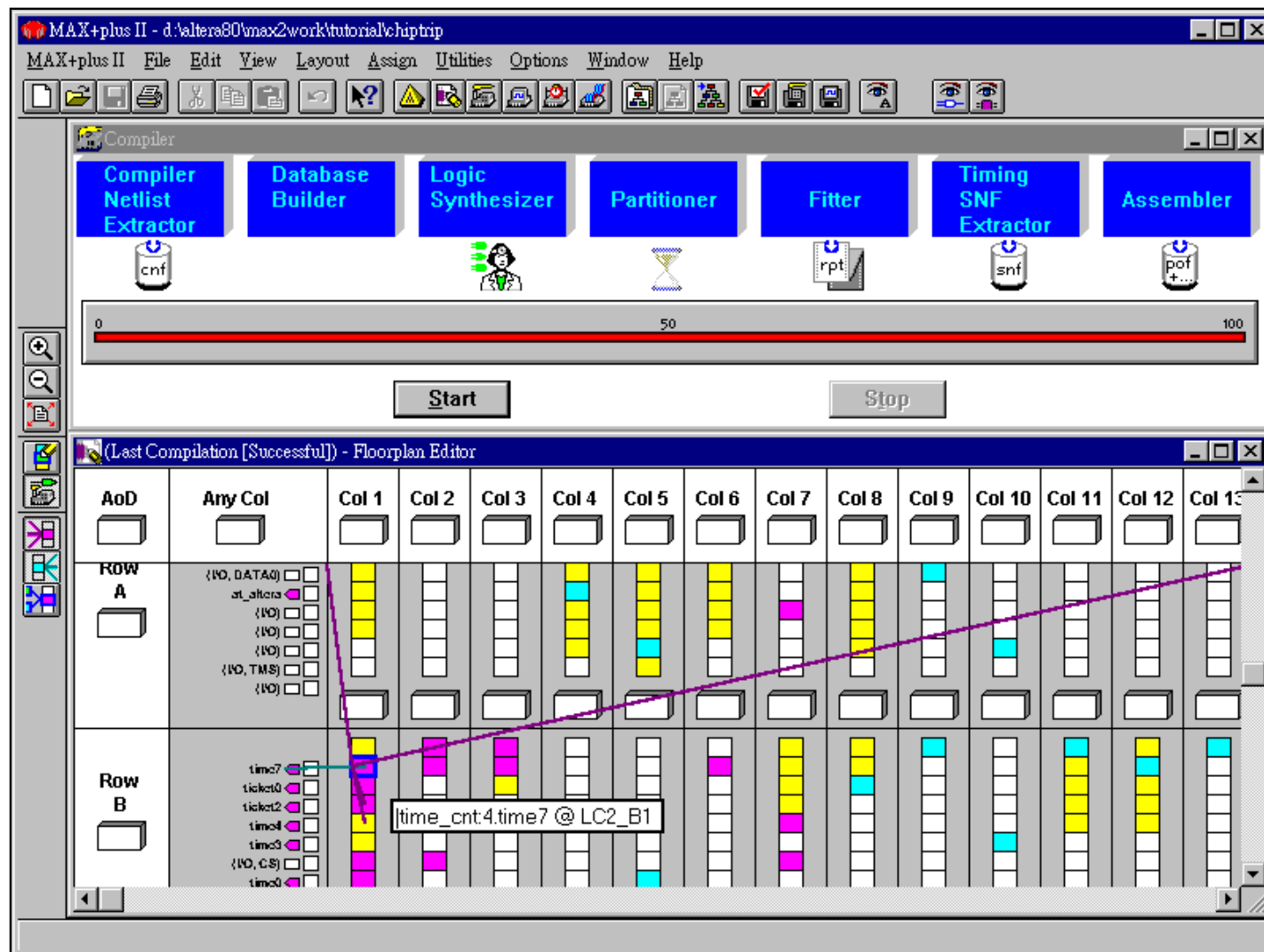
```

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

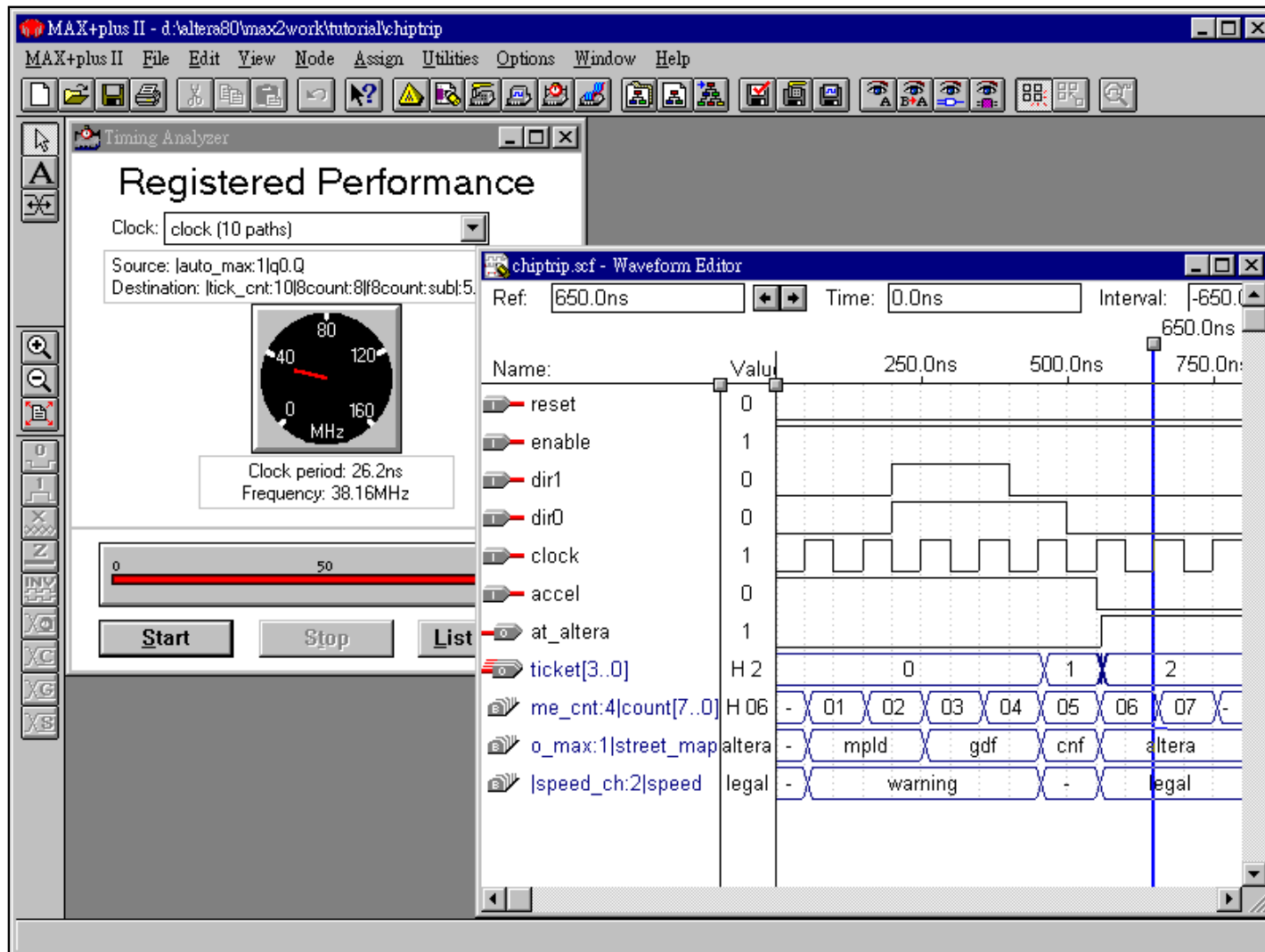


Project Verification

◆ MAX+PLUS II tools for project verification

- MAX+PLUS II Simulator
- MAX+PLUS II Waveform Editor
- MAX+PLUS II Timing Analyzer

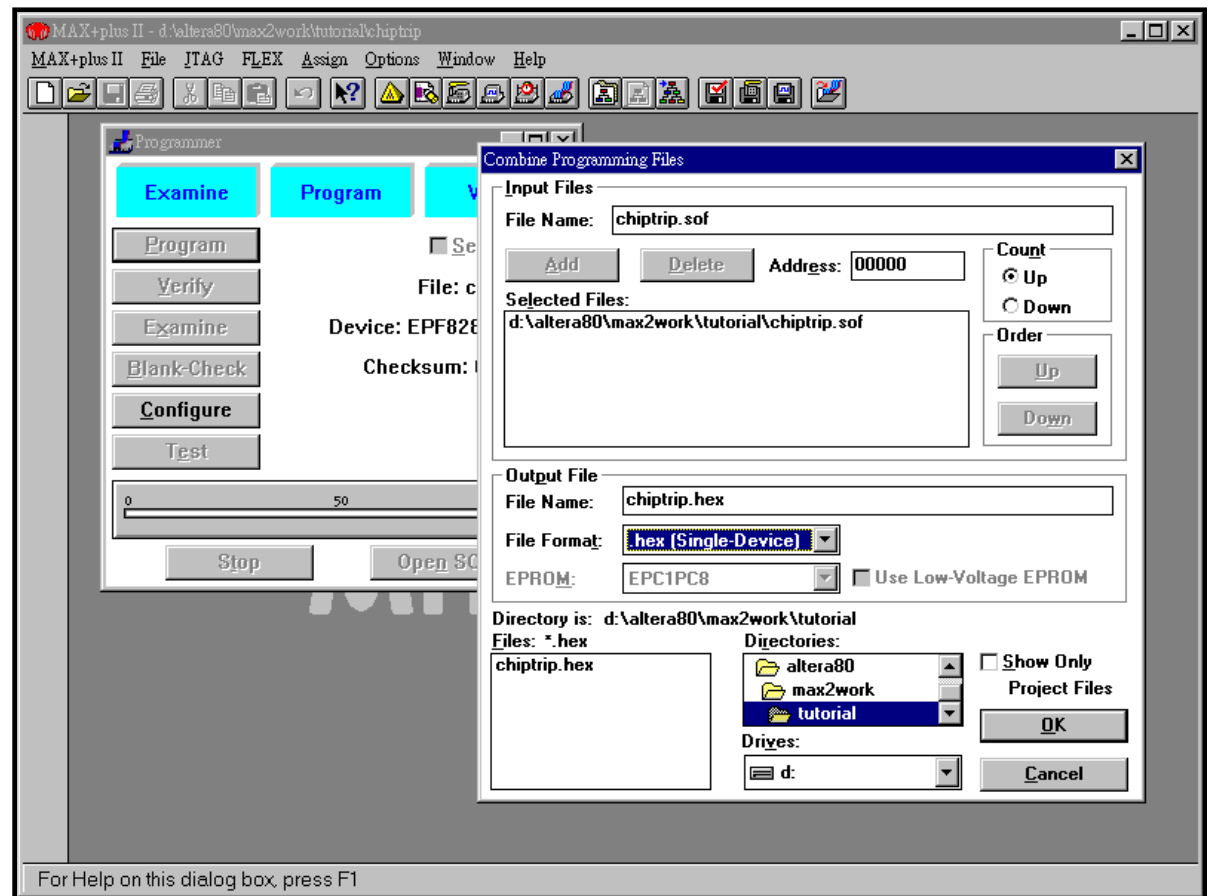
MAX+PLUS II Project Verification



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 **32MB** 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

Altera Software Guard

◆ MAX+PLUS II is not a freeware nor shareware

- Altera Software Guard (key-pro) is required to run MAX+PLUS II

◆ Prevent damage to the software guard

- If your software guard is faulty or damaged, CIC will not help replace a good one for you. Remember:
 - Disconnect the software guard before using the parallel port with the Interlink file transfer program!
 - Never try to crack the software guard

Starting MAX+PLUS II

◆ To start MAX+PLUS II...

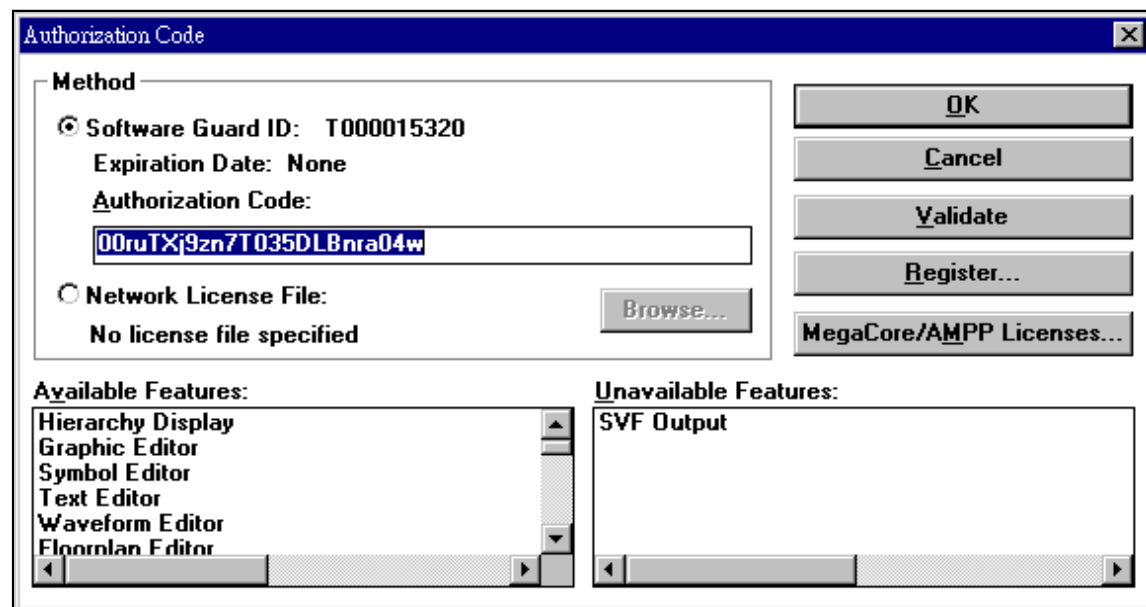
- Attach Altera Software Guard (key-pro) to the parallel port
- Double click on the MAX+PLUS II icon



Entering the Authorization Code

◆ When starting MAX+PLUS II for the first time

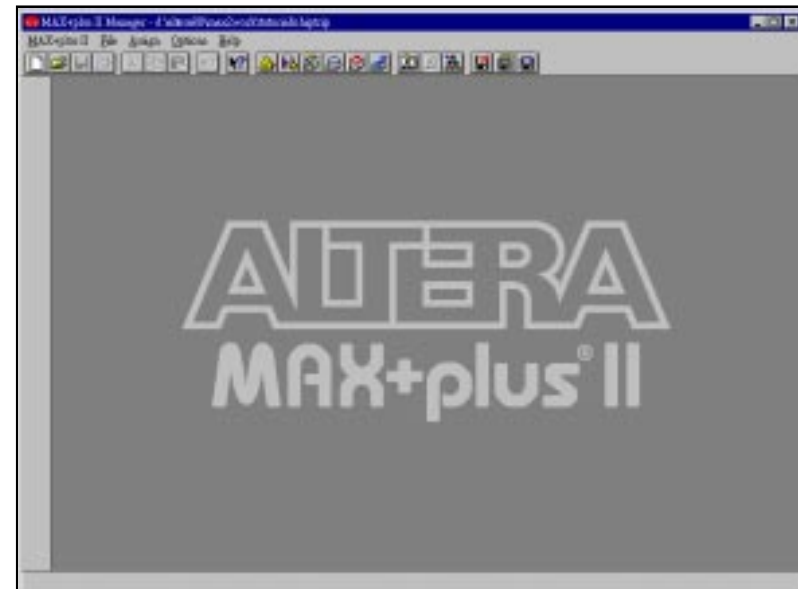
- MAX+PLUS II Authorization Code dialog box appears automatically
- You must enter an authorization code obtained from CIC according to the Guard-ID number, then click "Validate" button
- You can use all most MAX+PLUS II features after enter the correct auth-code



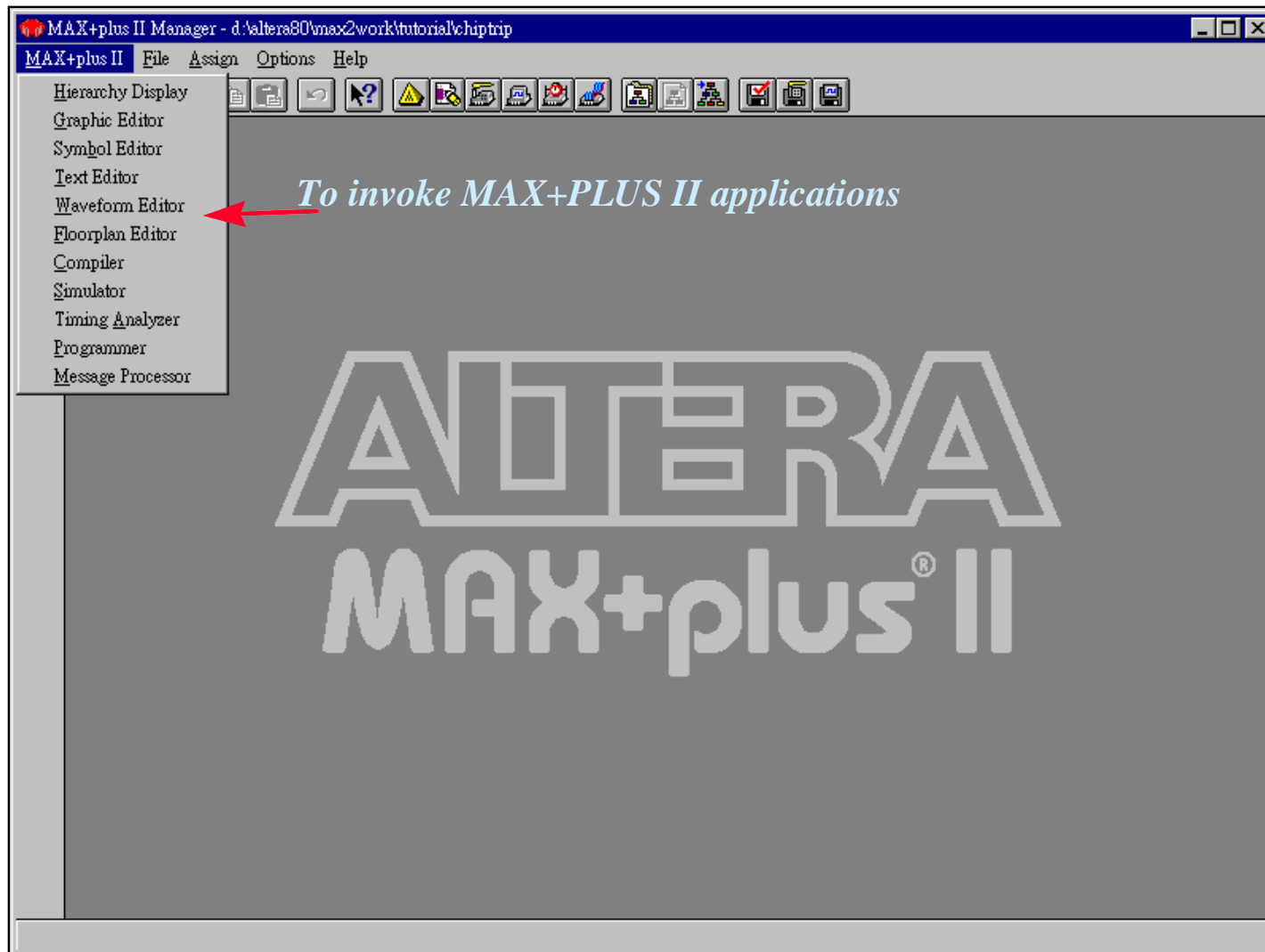
MAX+PLUS II Manager

◆ MAX+PLUS II Manager window

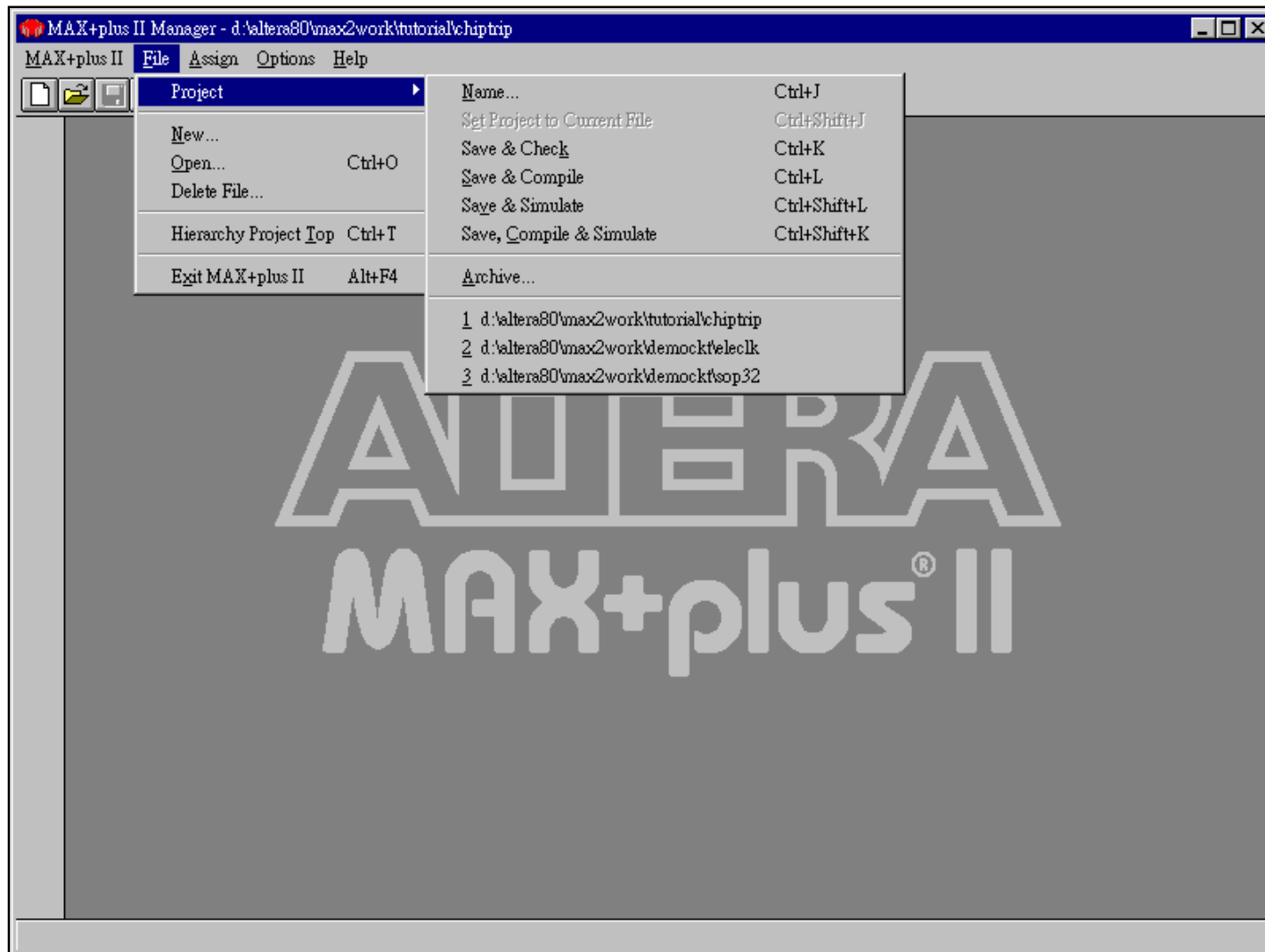
- Toolbar provides shortcuts for frequently used commands
- Status bar provides a brief description of each menu command and toolbar button
- Commands available from MAX+PLUS II Manager menus are also available in all other MAX+PLUS II applications.



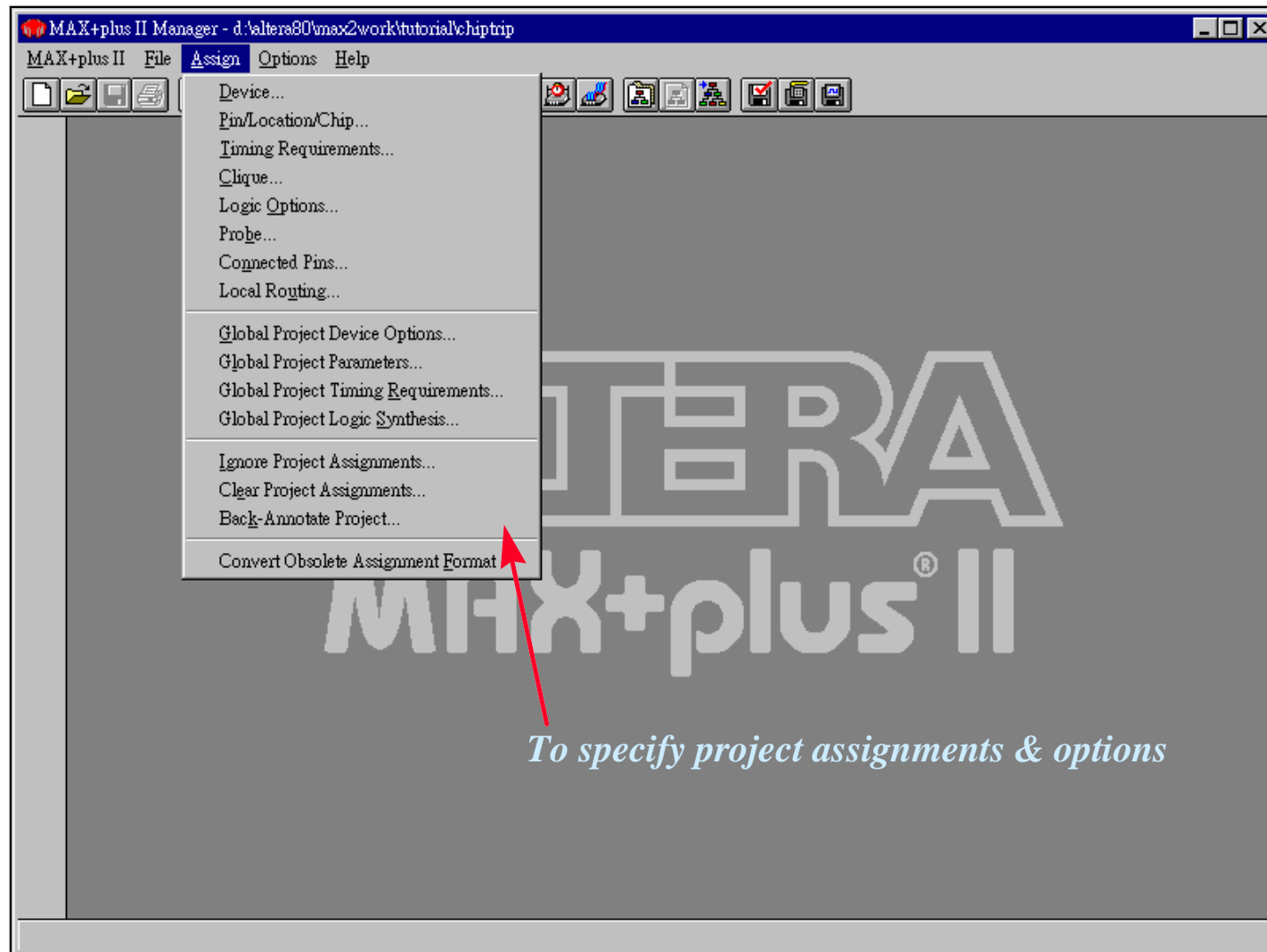
MAX+PLUS II Menu



File Menu

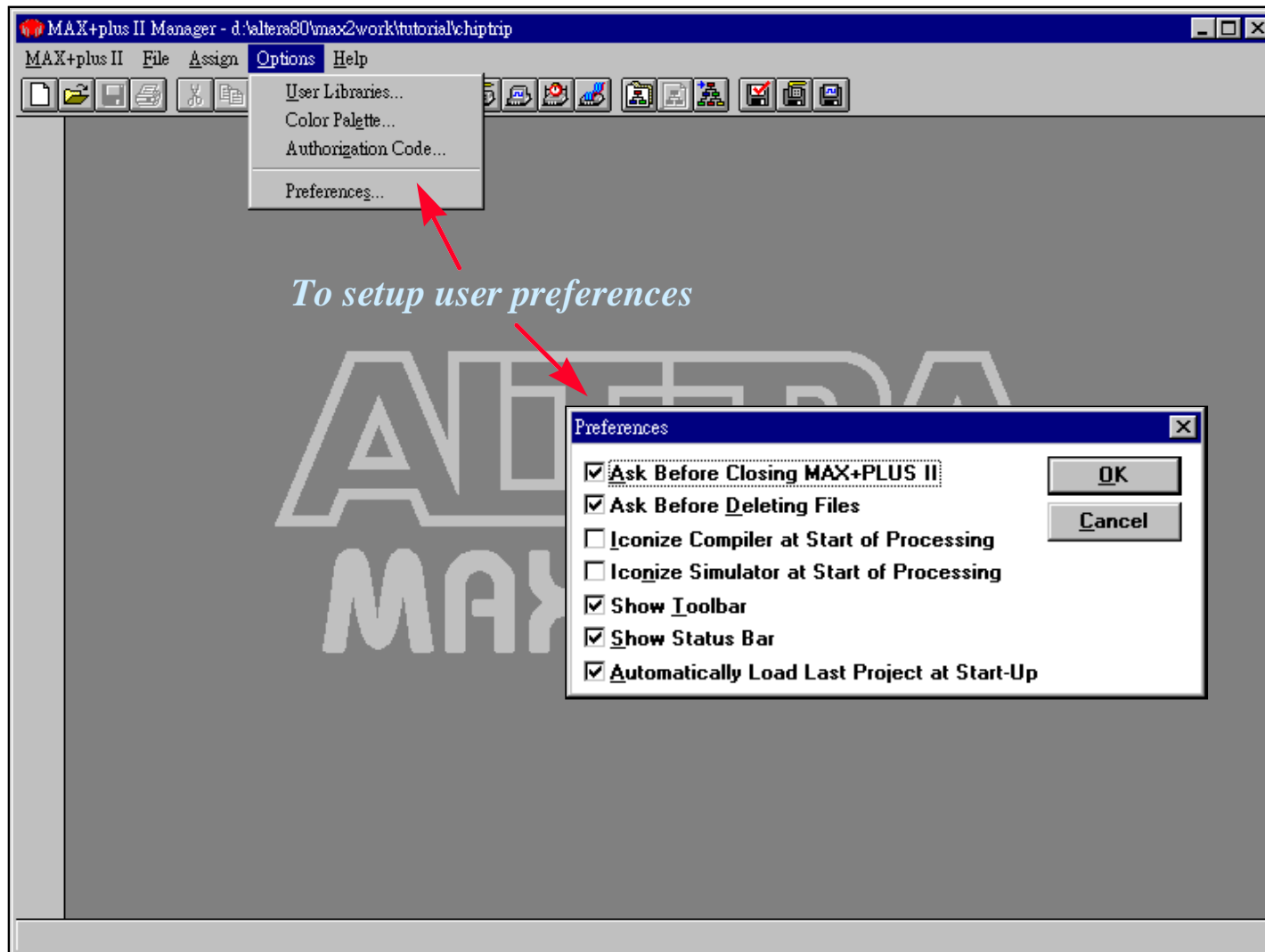


Assign Menu

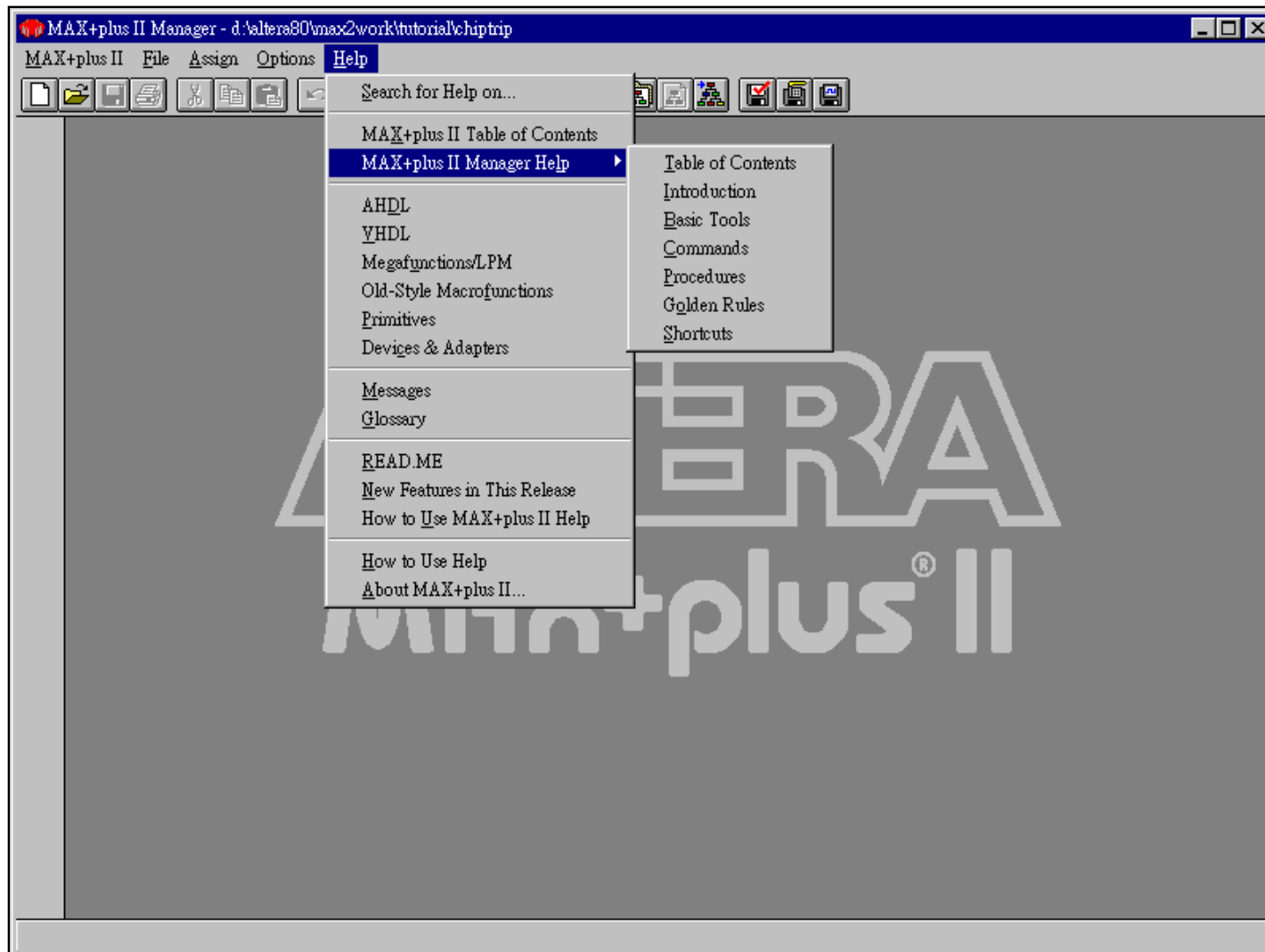


To specify project assignments & options

Options Menu



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 Project

◆ MAX+PLUS II project

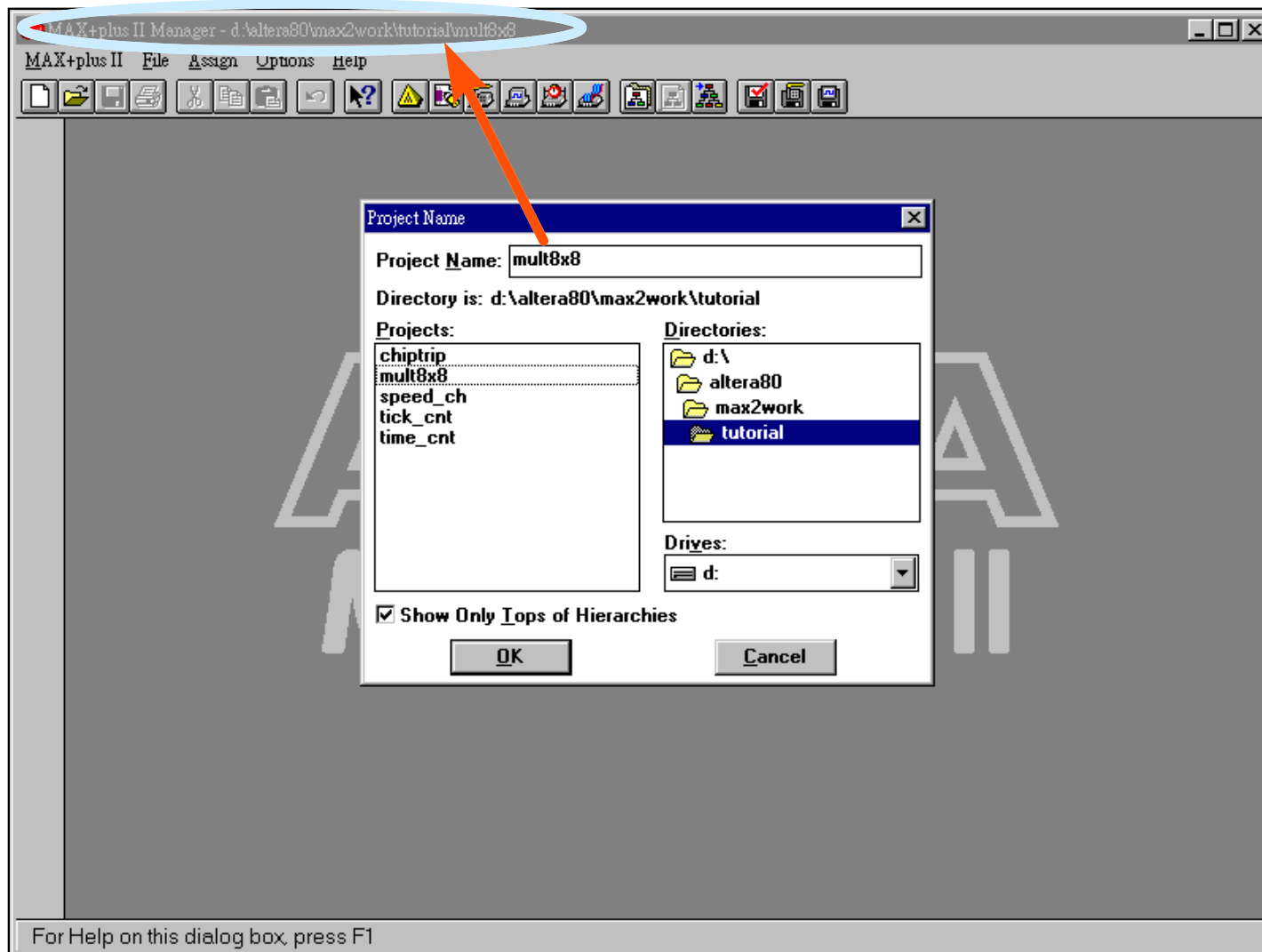
- A project consists of all files in a design hierarchy, including ancillary input and output files
- The project name is the name of the top-level design file, without the filename extension
- MAX+PLUS II performs compilation, simulation, timing analysis and programming on one project at a time

◆ To specify a project

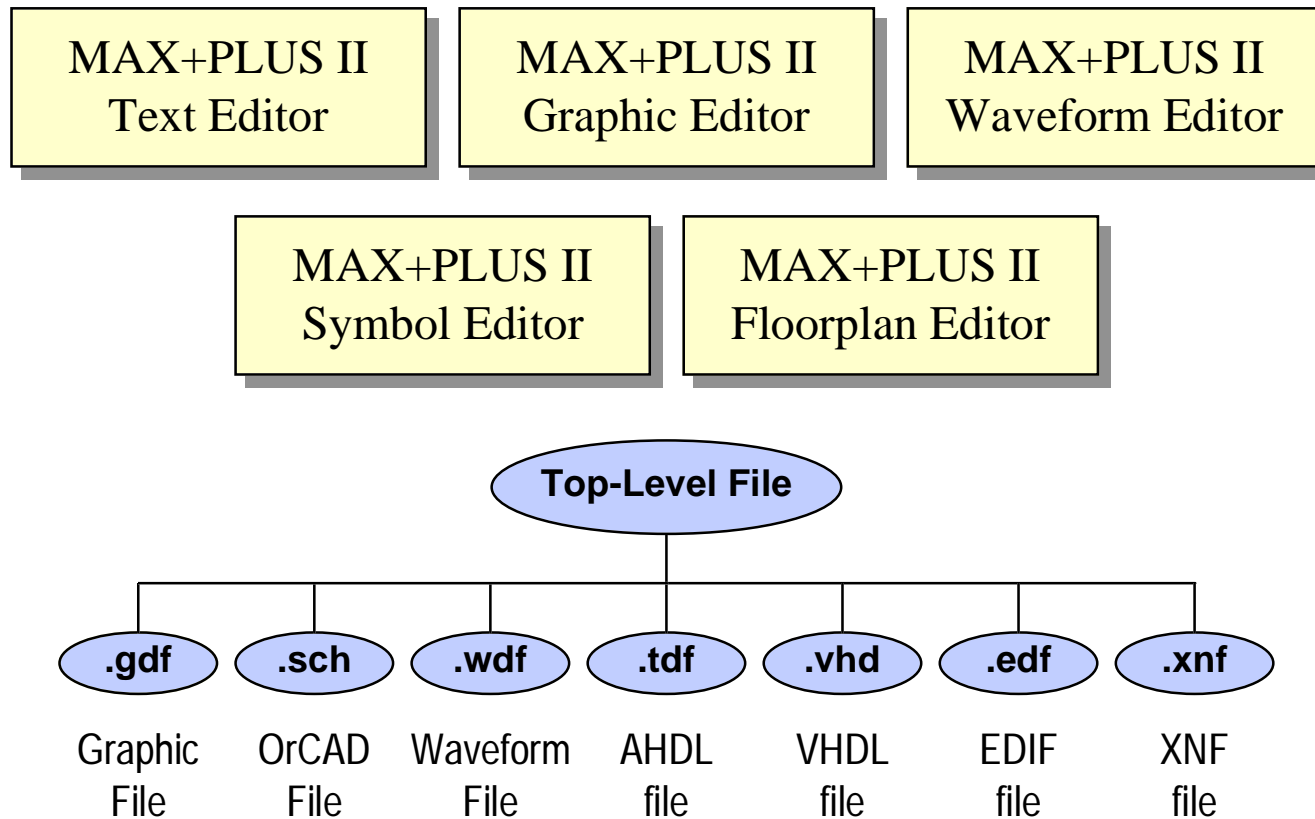
Menu: File -> Project -> Name... (To specify an existing or new design file)

Menu: File -> Project -> Set Project to Current File (To specify the current design file)

Specifying the Project Name



MAX+PLUS II Design Files

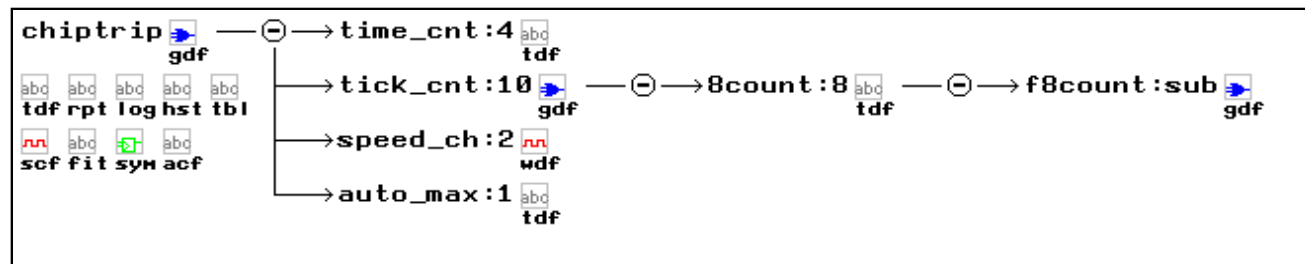


Hierarchy Display

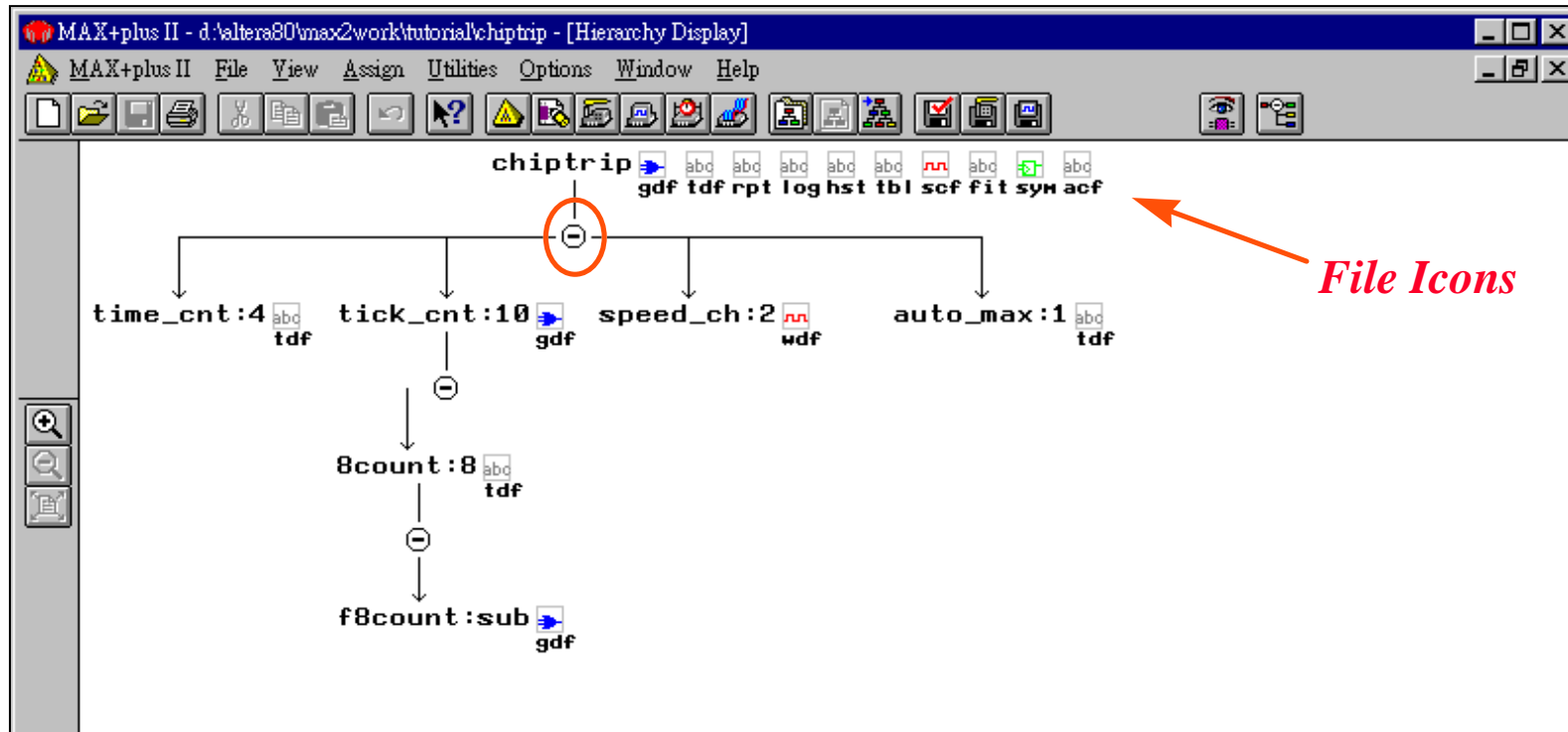
◆ MAX+PLUS II Hierarchy Display

- 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



Graphic Design Entry

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

MAX+PLUS II

Graphic Design Environment

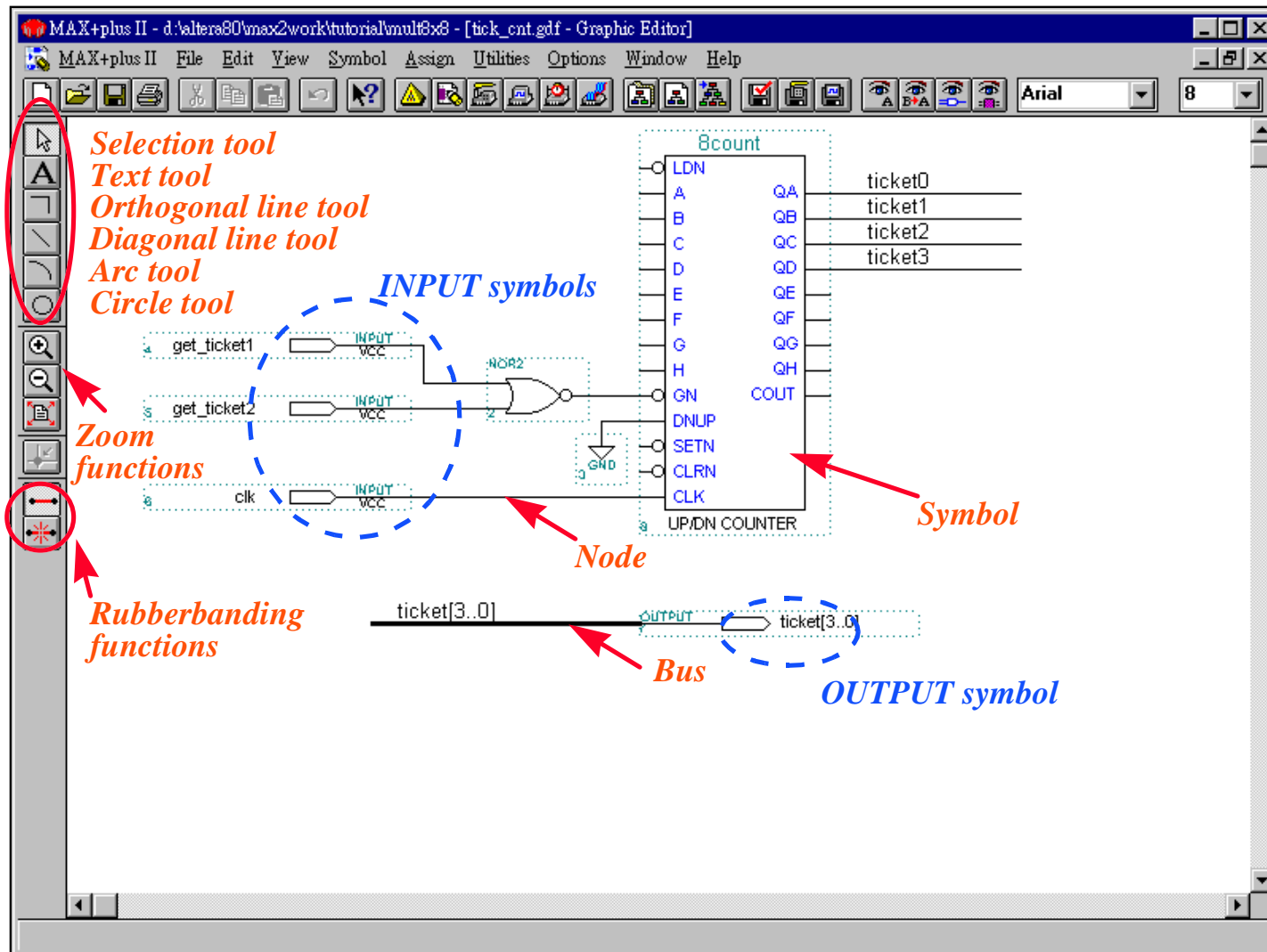
◆ MAX+PLUS II Graphic Editor

- To create Altera graphic design files (*.gdf)
- What-you-see-is-what-you-get design environment
- Sophisticated schematic capture program that allows you to enter even complex designs quickly & easily
 - User-friendly GUI
 - Complete on-line help
- The extensive primitive, megafunction, LPM and macrofunction libraries provide basic building blocks for constructing a design
- The symbol-generation capability lets you build your own libraries of custom functions. Symbols may represent any type of design file.

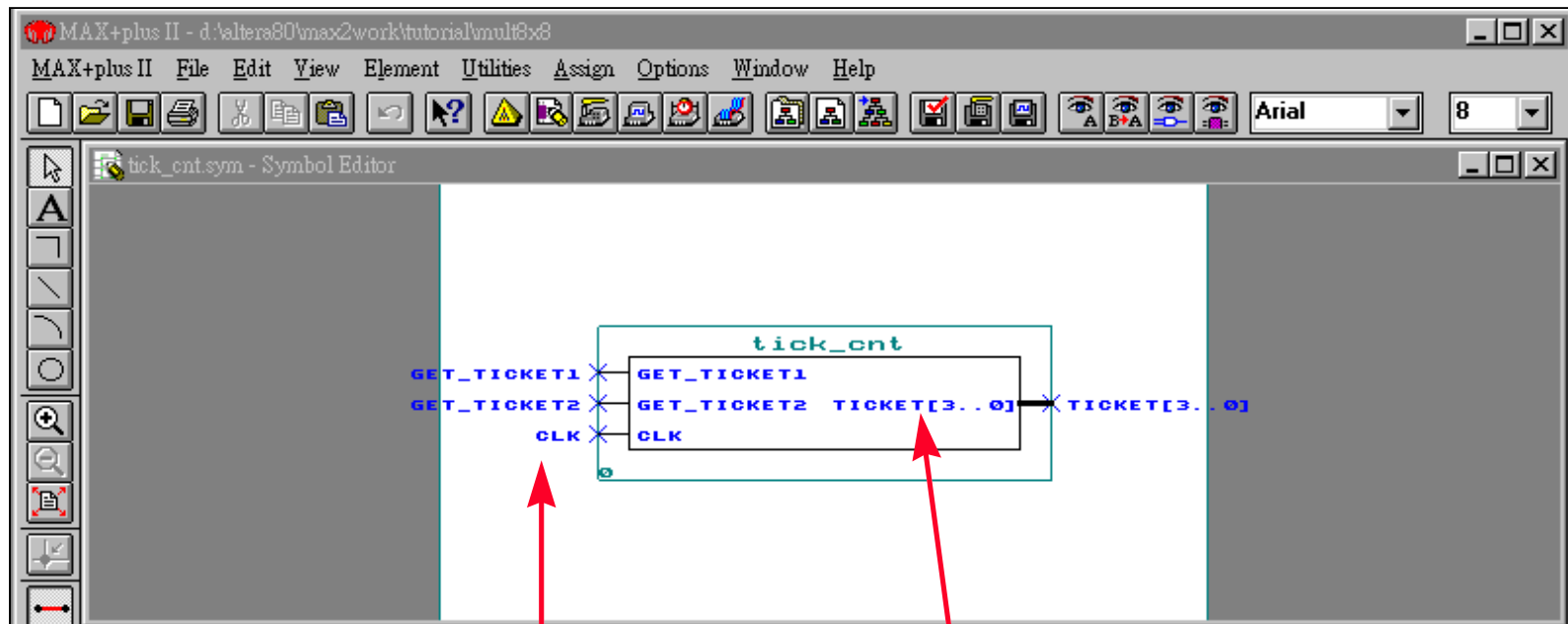
◆ MAX+PLUS II Symbol Editor

- To view, create, or edit a symbol that represents a logic circuit (*.sym)
- "Create Default Symbol" command is available from File menu of Graphic, Text, and Waveform Editors

Graphic Editor Window



Symbol Editor Window



Pinstub names inside the symbol border can be moved around and edited.

Duplicate pinstub names outside the symbol border show which name is associated with a pinstub

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

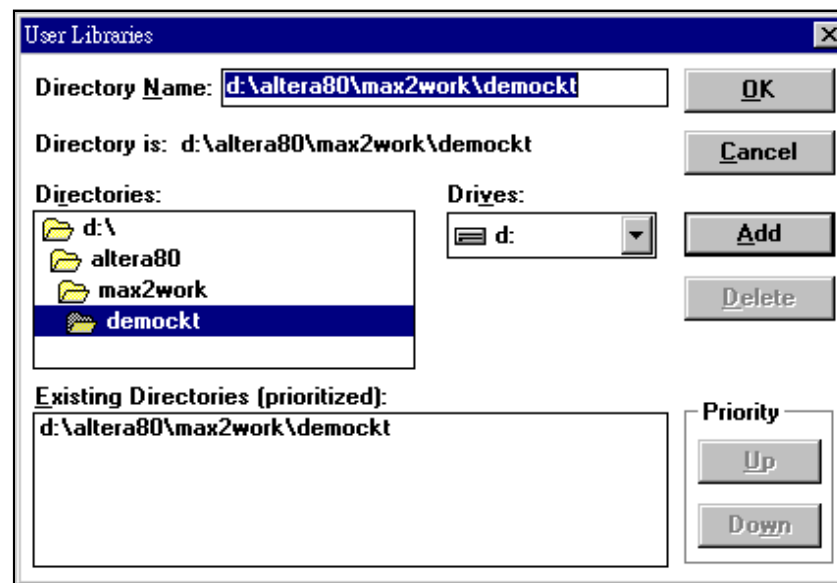
- Example: $A[31..0], B, C[3..0]$
- A sequential bus name consists of a series of node names and/or bus names, separated by commas (,). The first node or bus bit in the series is the MSB, the last node in the series is the LSB.

User Libraries

◆ To specify the user libraries

- User libraries contain Symbol Files and design files for your own megafunctions and macrofunctions
- The primary user library is under the project directory
- You can specify the path to another user library

Menu: Option -> User Libraries...



System Libraries

◆ Primitive library

- MAX+PLUS II provides many primitive functions, including input & output primitives, logic primitives, flip-flop & latch primitives and buffer primitives
- **Primitive Array**: A primitive array is a single primitive that represents multiple identical primitives.

◆ Macrofunction (old-style) library

- Over 300 74-series, bus, architecture-optimized, and application-specific macrofunctions

◆ Megafunction/LPM library

- **LPM**: Libraries of Parameterized Modules
- MAX+PLUS II provides standard LPM functions & other parameterized functions
 - Gates, arithmetic components, storage components, and other functions
- MegaCore megafunctions: pre-verified HDL design files for complex system-level functions that can be purchased from Altera

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

Using LPM Functions

◆ LPM features

- Architecture-independent design entry
- Efficient design mapping
- Tool-independent design entry
- Specification of a complete design

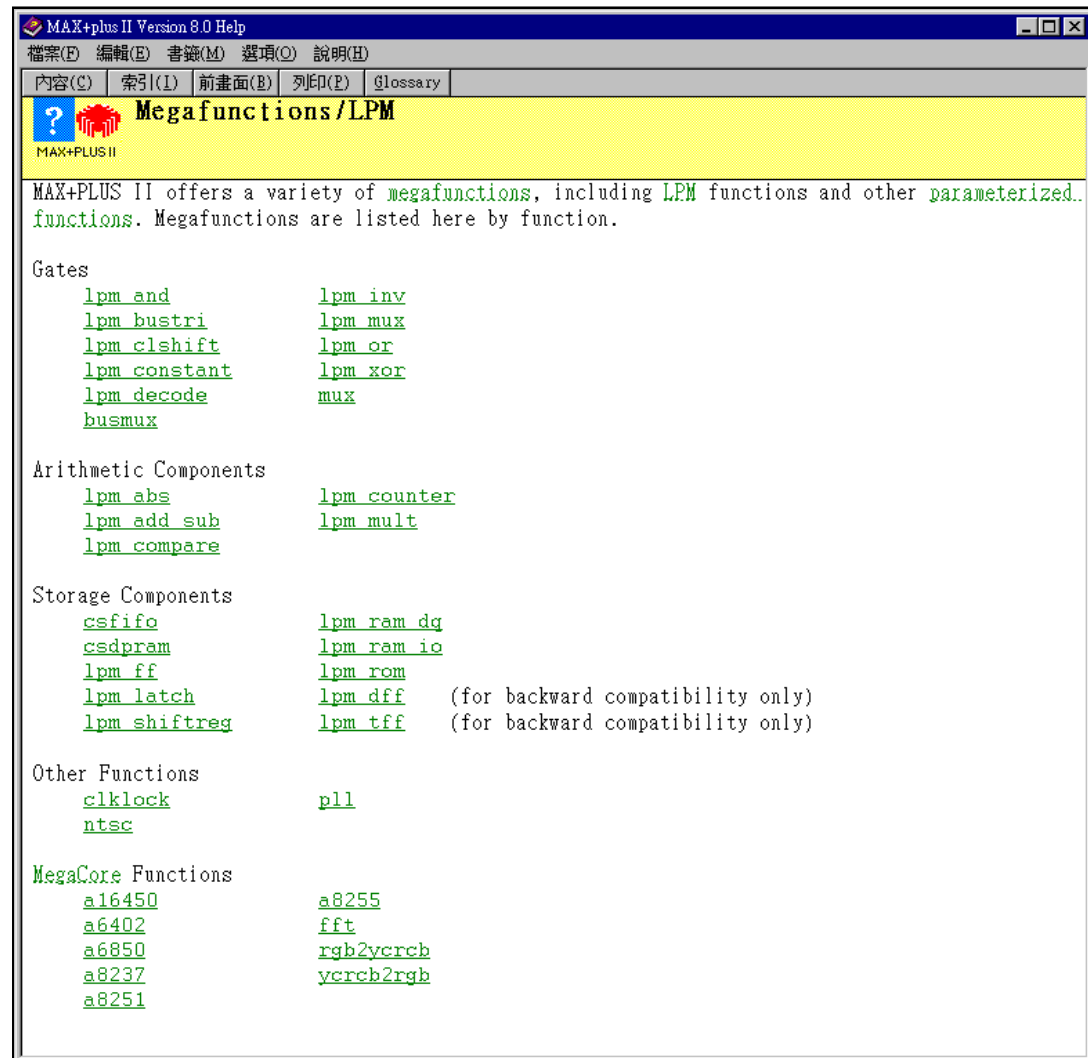
◆ Use LPM functions to reduce the development time

- Each function contains parameters that allow it to expand in many dimension, for example, the `LPM_COUNTER` function allows the user to create counters
 - with widths ranging from 1 to 256 bits
 - with up-down control
 - with asynchronous or synchronous loading function
 - ...
- Altera recommends using LPM functions instead of any other type of old-style macrofunction

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



Creating a New GDF File

- ◆ **Open a new graphic design file**

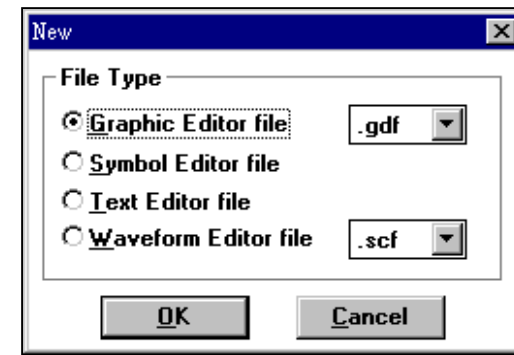
Menu: *File -> New... -> Graphic Editor file (.gdf)*

- ◆ **Save as a GDF file immediately**

Menu: *File -> Save As... ->*

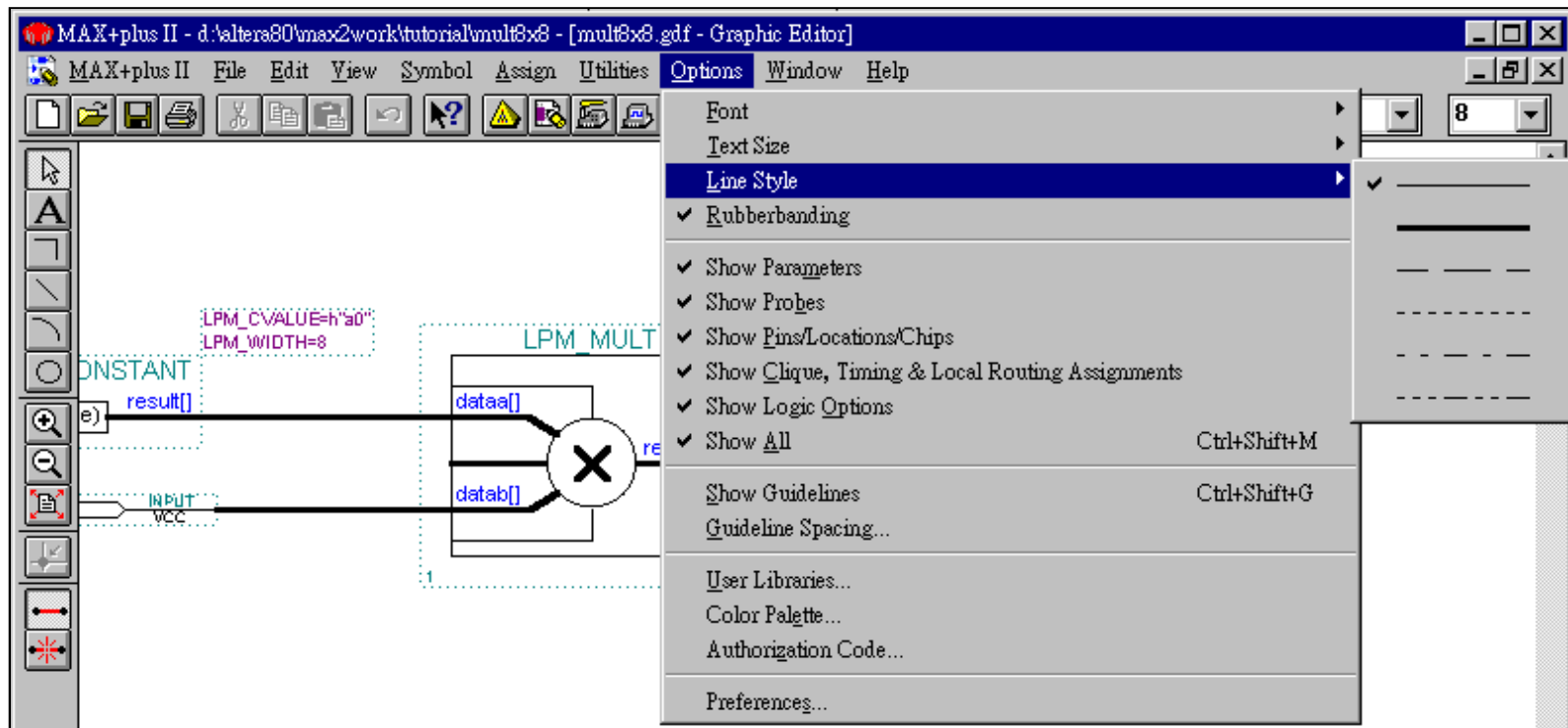
- ◆ **Set project to the current GDF file**

Menu: *File -> Project... -> Set Project to Current File*

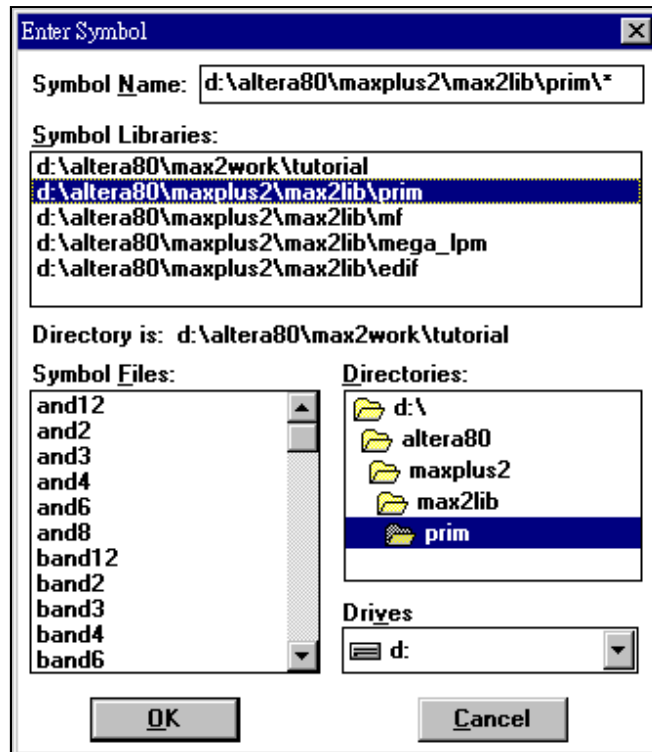


Setting Graphic Editor Options

- ◆ Font & text size options
- ◆ Line style option
- ◆ Guideline controls ...



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

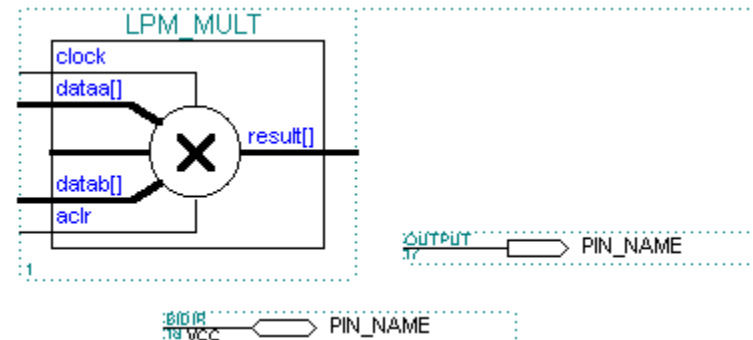
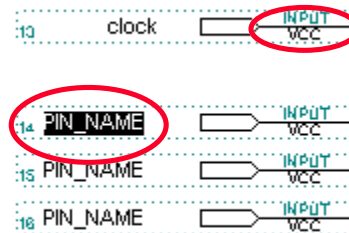
- Input pin/port: enter a INPUT symbol
- Output pin/port: enter a OUTPUT symbol
- Bidirectional pin/port: enter a BIDIR symbol

◆ 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

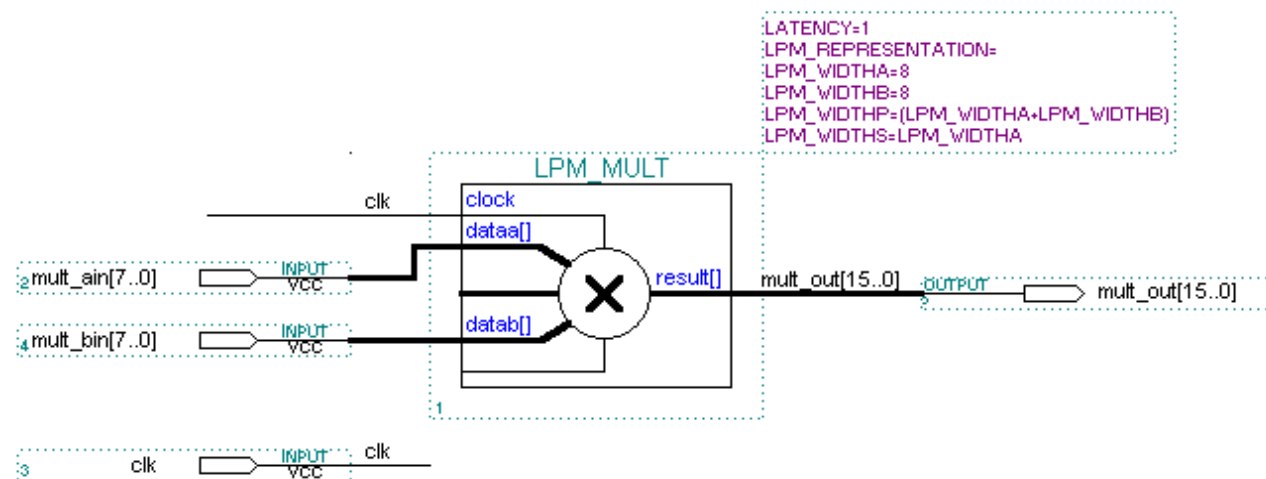


Connecting the Symbols

◆ To connect the symbols

- Use mouse to draw lines between symbols
 - Must choose appropriate line style from Options menu
- Or use "rubberbanding" function to connect symbols
- Or connect nodes & buses by name

◆ Name the nodes & buses



Editing Ports/Parameters of LPM

◆ Edit Ports/Parameters command

- To edit the port status of a megafunction/LPM

Menu: *Symbol -> Edit Ports/Parameters...*
(or press *ENTER* on the symbol)

The screenshot shows the 'Edit Ports/Parameters' dialog box for the LPM_MULT megafunction. The dialog is divided into two main sections: 'Ports' and 'Parameters'.

Ports Section:

- Function Name:** LPM_MULT
- Port Name:** aclr
- Port Status:** ☒ Used, ☐ Unused
- Inversion:** ☒ None, ☐ Pattern/Radix: [] hex
- Table:**

Name:	Status:	Inversion:
aclr	Used	None
clock	Used	None
dataa[LPM_WIDTHA-1..0]	Used	None
datab[LPM_WIDTHB-1..0]	Used	None
result[LPM_WIDTHP-1..0]	Used	None

Parameters Section:

- Parameter Name:** INPUT A IS CONSTANT
- Parameter Description:** Hint to help minimize the number of LCELLs
- Parameter Value:** "NO"
- Buttons:** Change, Clear
- Table:**

Name:	Value:
INPUT A IS CONSTANT	"NO"
INPUT B IS CONSTANT	"NO"
LPM_PIPELINE	3
LPM_REPRESENTATION	"UNSIGNED"
LPM_WIDTHA	8

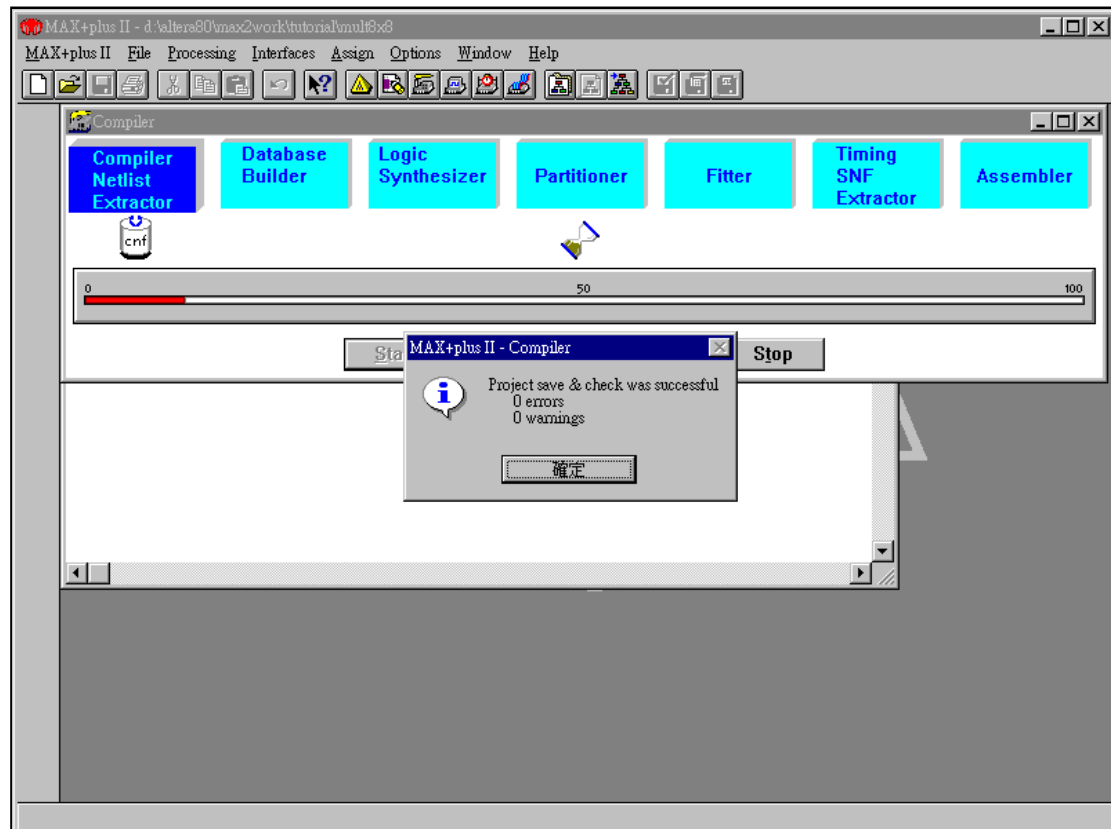
Buttons: OK, Cancel

Saving & Checking the Design

◆ Save & Check command

- MAX+PLUS II Compiler will be opened to check for basic errors

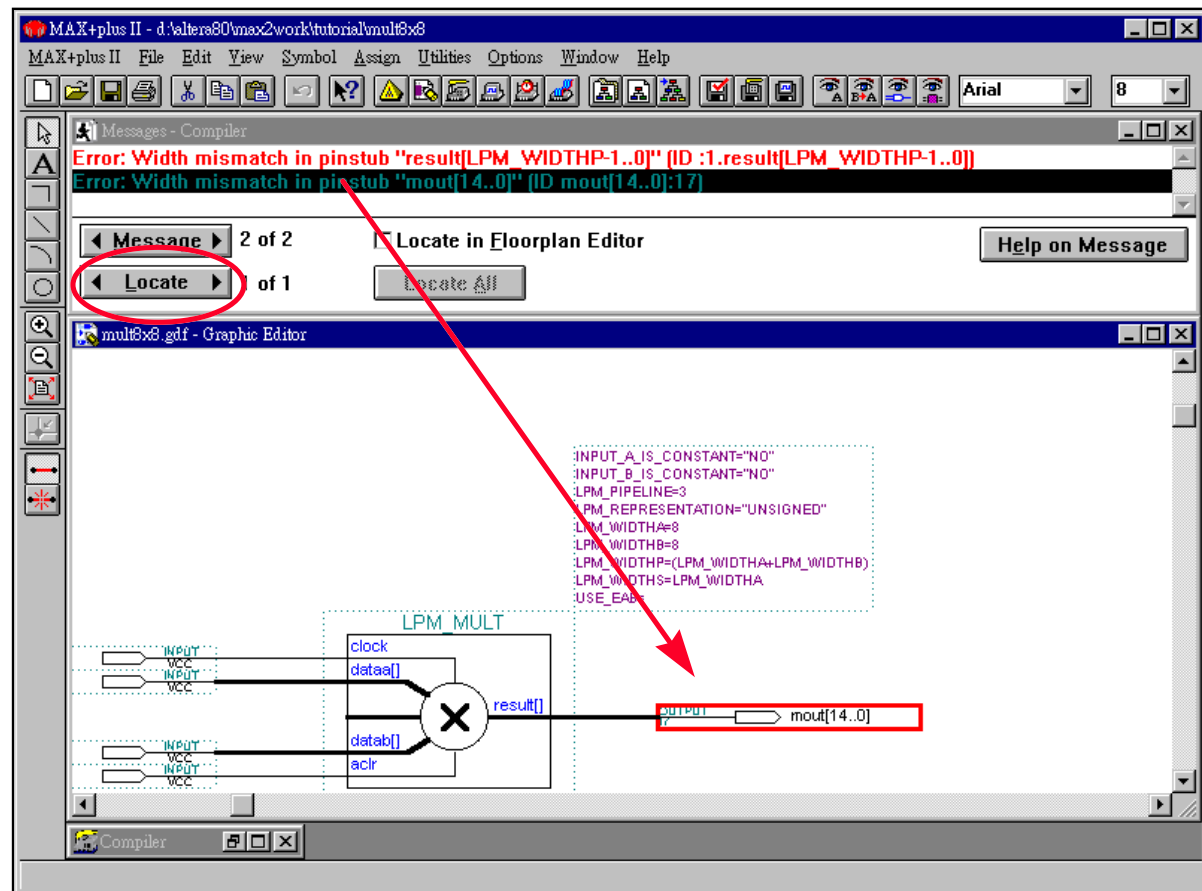
Menu: File -> Project -> Project Save & Check



Error Locating

◆ If an error occurs...

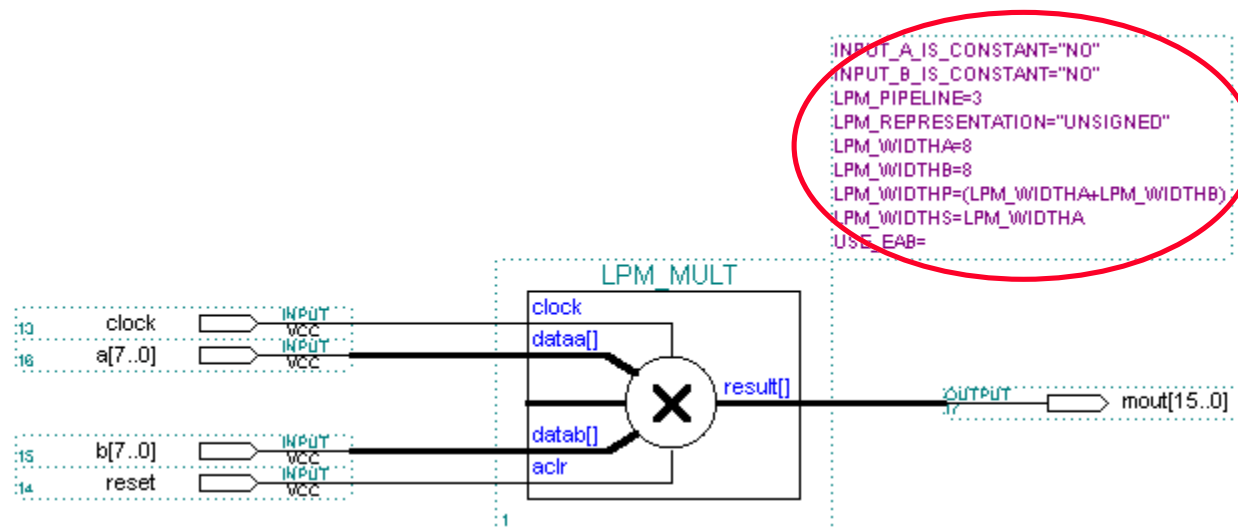
- MAX+PLUS II Message Processor is opened to display the error message
- To locate the error, simply select the error message, and click "Locate" button



Example: Multiplier

◆ Design a multiplier with `LPM_MULT`

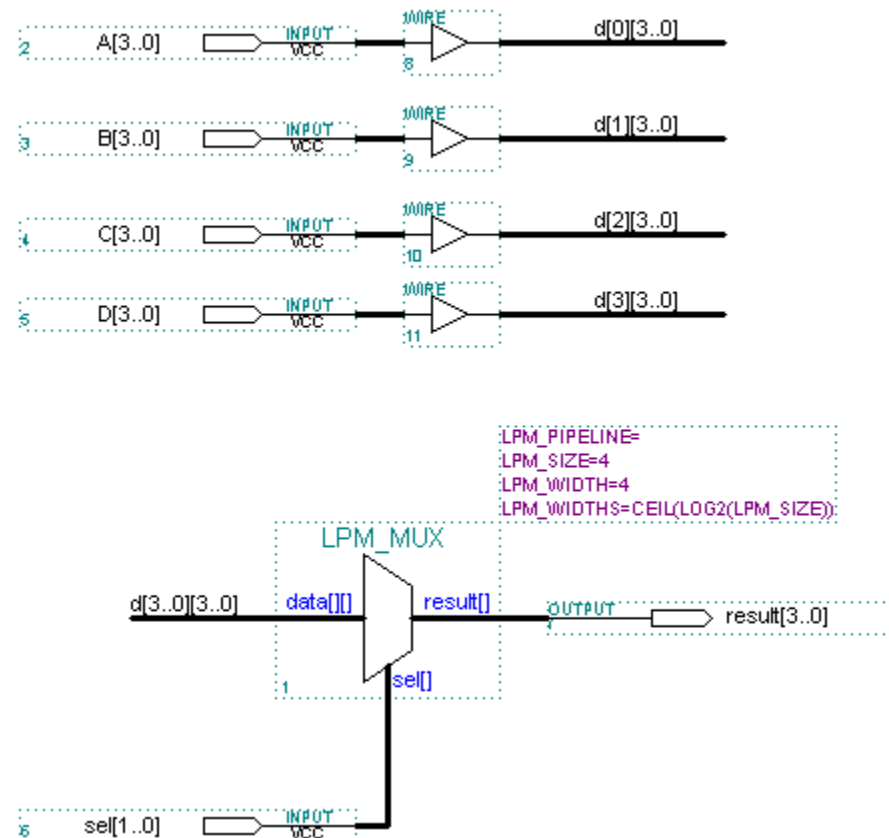
- The easiest way to create a multiplier is to use the `LPM_MULT` function
 - Can be unsigned or signed
 - Can be pipelined
 - Also can create a MAC(Multiplier-Accumulator) circuit



Example: Multiplexer

◆ Design a multiplexer with LPM_MUX

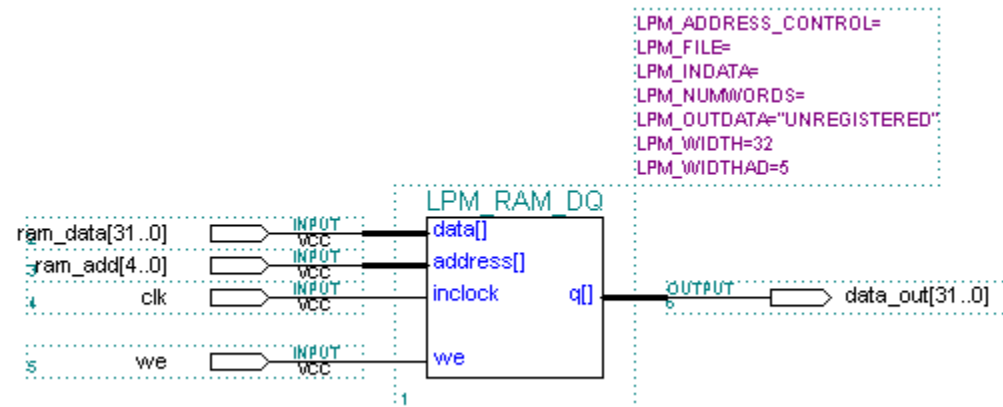
- Use **WIRE** primitive to rename a bus or node
- LPM_MUX data input is a dual range bus



Example: RAM

◆ Design RAM circuit with LPM

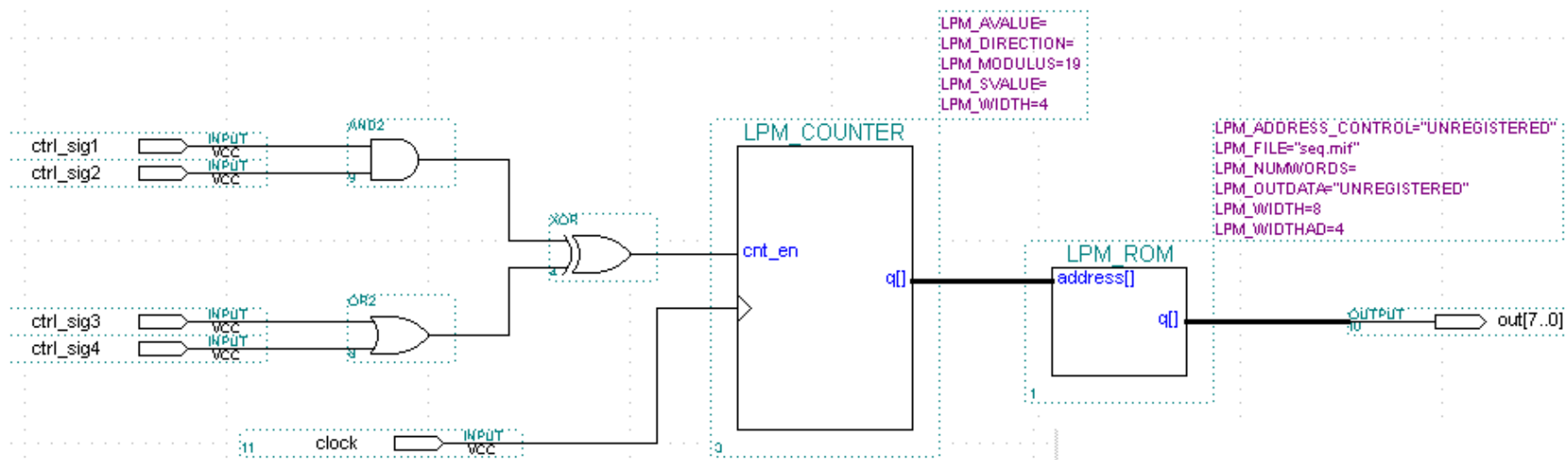
- Use LPM_RAM_IO to design RAM with a single input & output port
- Use LPM_RAM_DQ to design RAM with separate input & output ports



Example: Sequencer

◆ Design a sequencer with LPM_COUNTER & LPM_ROM

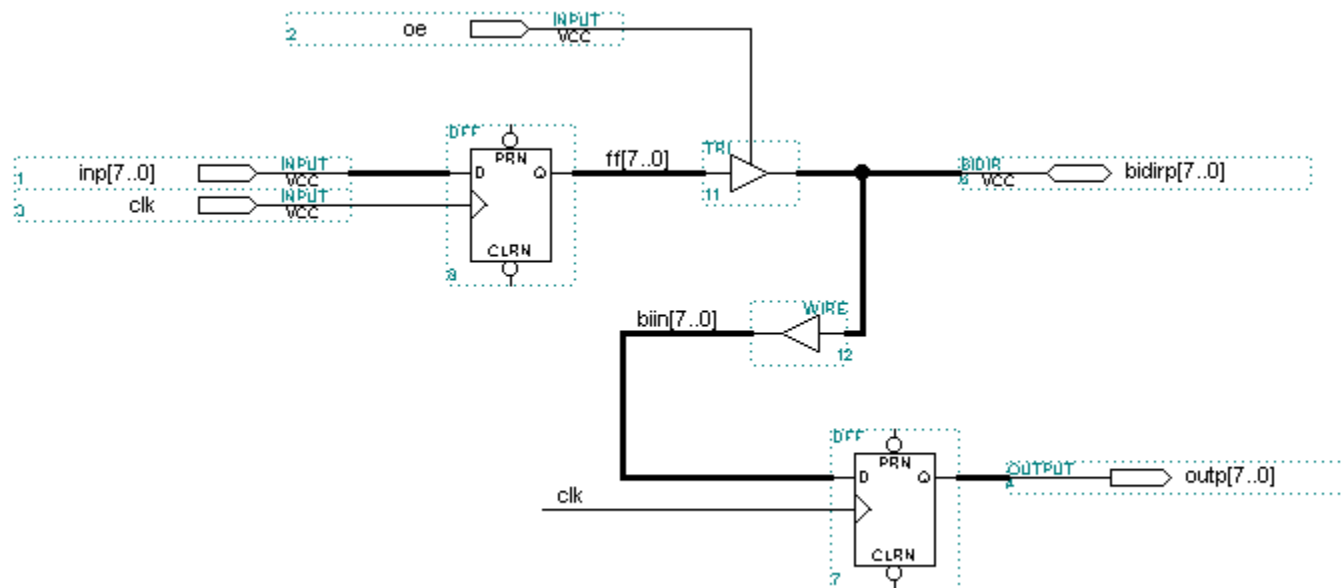
- ROM data is specified in a Memory Initialization File (.mif) or a Intel-Hex File (.hex)
- This example only sequences through 19 states so the modulus of lpm_counter is set to 19. It uses a small section of an EAB (19 out of 256-address locations)



Example: Bidirectional Pin

◆ Use TRI & BIDIR pin symbol

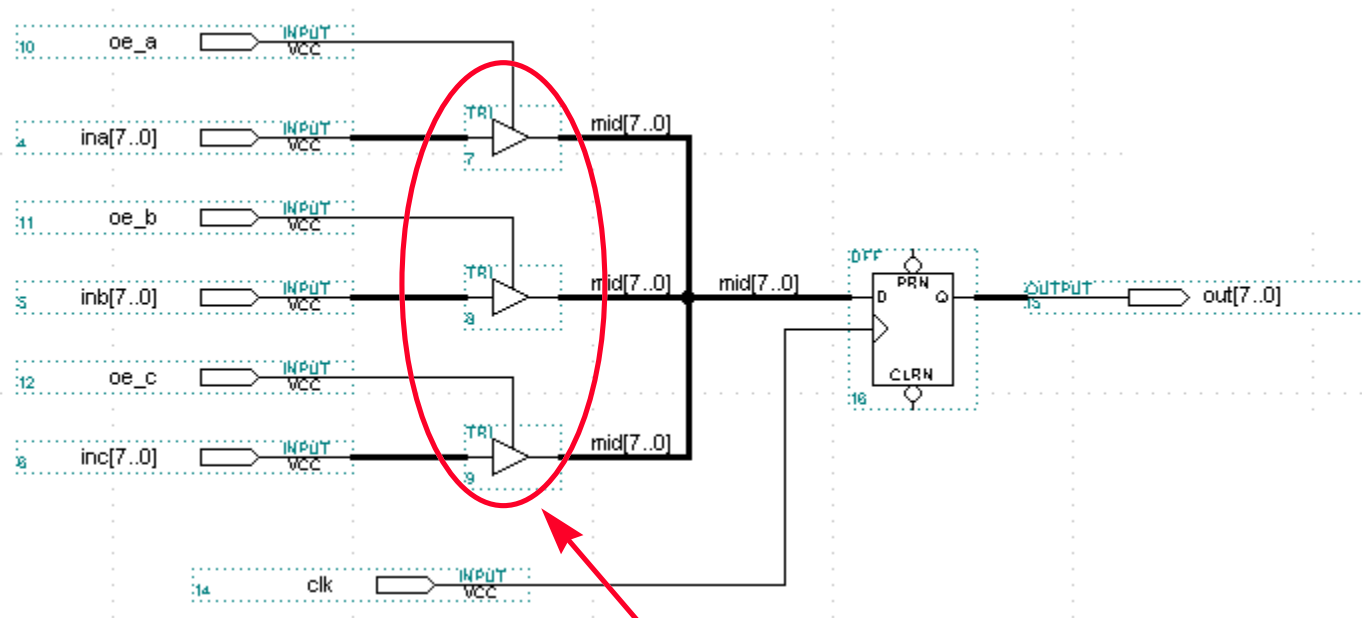
- If the TRI symbol feeds to a output or bidirectional pin, it will be implemented as tri-state buffer in the I/O cell



Example: Tri-State Buses - (1)

◆ 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

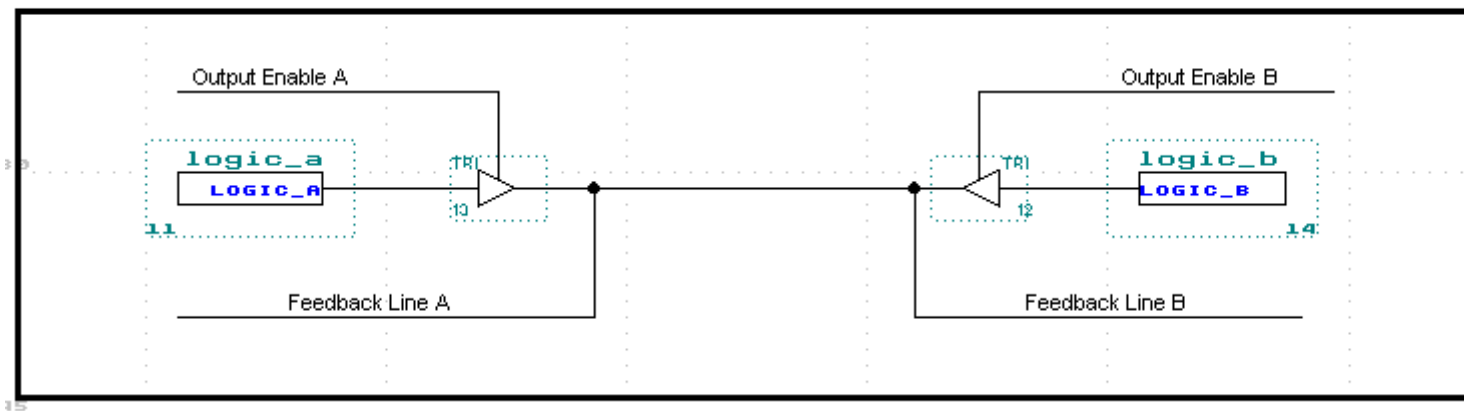


*MAX+PLUS II will automatically convert it into a multiplexer.
If the tri-state buffers feed a pin, a tri-state buffer will be available
after the multiplexer.*

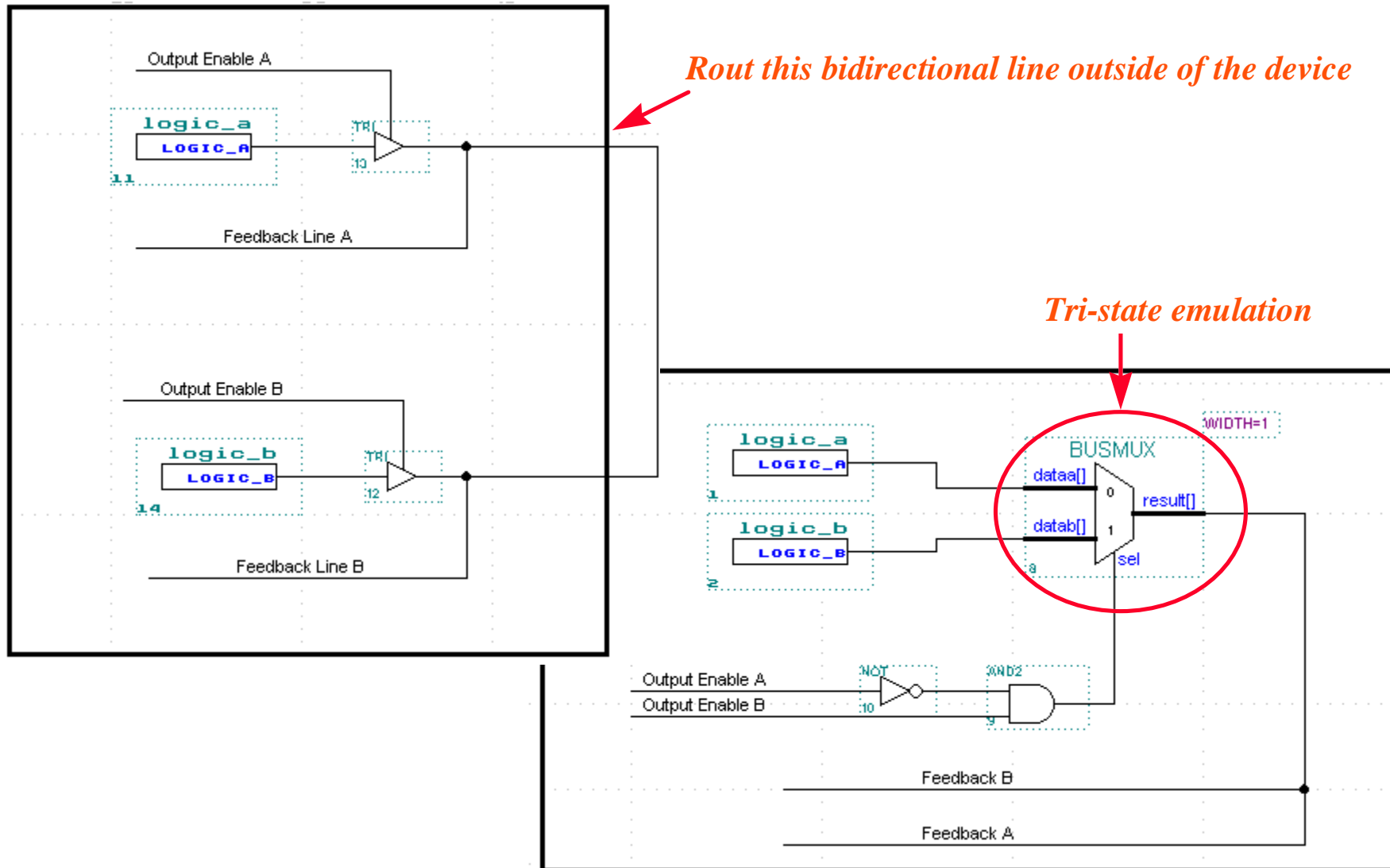
Example: Tri-State Buses - (2)

◆ 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 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 - (3)



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, 0, 0, 0, 0;
    H"8"    => 1, 1, 1, 1, 1, 1, 1;
    H"9"    => 1, 1, 1, 1, 0, 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)

◆ Logic section (required)

- To define the logical operations of the file
 - Defaults statement
 - Assert statement
 - Boolean equations
 - Boolean control equations
 - Case statement
 - If-Generate statement
 - If-Then statement
 - Truth table statement
 - In-line logic function reference

AHDL Basic Elements - (1)

◆ Numbers in AHDL

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

◆ Group

- Sequential group: (a, b, c)
- Single-range group: a[4..1] = (a4, a3, a2, a1)
- Dual-range group: d[2..0][1..0] = (d2_1, d2_0, d1_1, d1_0, d0_1, d0_0)
- Entire range group: 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";
```

AHDL Syntax - (2)

◆ Parameters statement

- To declare one or more parameters that control the implementation of a parameterized megafunction or 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);
```

AHDL Syntax - (5)

◆ Define statement

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

```
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;
```

AHDL Syntax - (6)

◆ 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;
```

AHDL Syntax - (8)

◆ 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:

```
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 **INPUT**, **OUTPUT**, **BIDIR**, **MACHINE INPUT**, or **MACHINE OUTPUT**
 - **MACHINE INPUT** & **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:

```
SUBDESIGN top  
(  
    foo, bar, clk1, clk2 : INPUT = VCC;  
    a0, a1, a2, a3, a4   : OUTPUT;  
    b[7..0]              : BIDIR;  
)
```

AHDL Syntax - (10)

◆ Variable section

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

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;
  b      : 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;
```


AHDL Syntax - (12)

◆ 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)
```

comp will have the following ports:

comp.a[], comp.b[], comp.less, comp.equal, comp.greater

adder will have the following ports:

adder.dataa[], adder.datab[], adder.result[]

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



AHDL Syntax - (15)

◆ 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);
```

ss_out is a state machine output

VARIABLE

```
ss : MACHINE;
```

```
BEGIN
```

```
ss = ss_def (sys_clk, reset, !hold);
```

```
IF ss == s0 THEN
```

```
:
```

```
ELSIF ss == s1 THEN
```

```
:
```

```
END;
```

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:

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:

```
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:

```
IF DEVICE_FAMILY == "FLEX8K" GENERATE  
    c[] = 8kadder(a[], b[], cin);  
ELSE GENERATE  
    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;
```

AHDL Syntax - (22)

◆ In-line logic function reference

- A Boolean equation that implements a logic function
- 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)  
    RETURNS (result[LPM_WIDTH-1..0], cout, overflow);
```

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

Positional port association

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

Named port association

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:

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

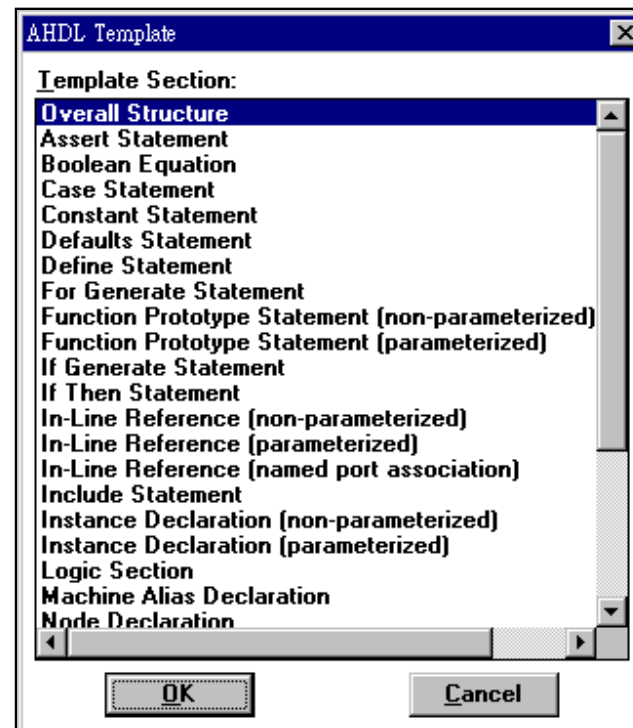
- AHDL templates & examples
- AHDL context-sensitive help
- Syntax coloring
- Error location
- Resource & device assignments

AHDL Templates

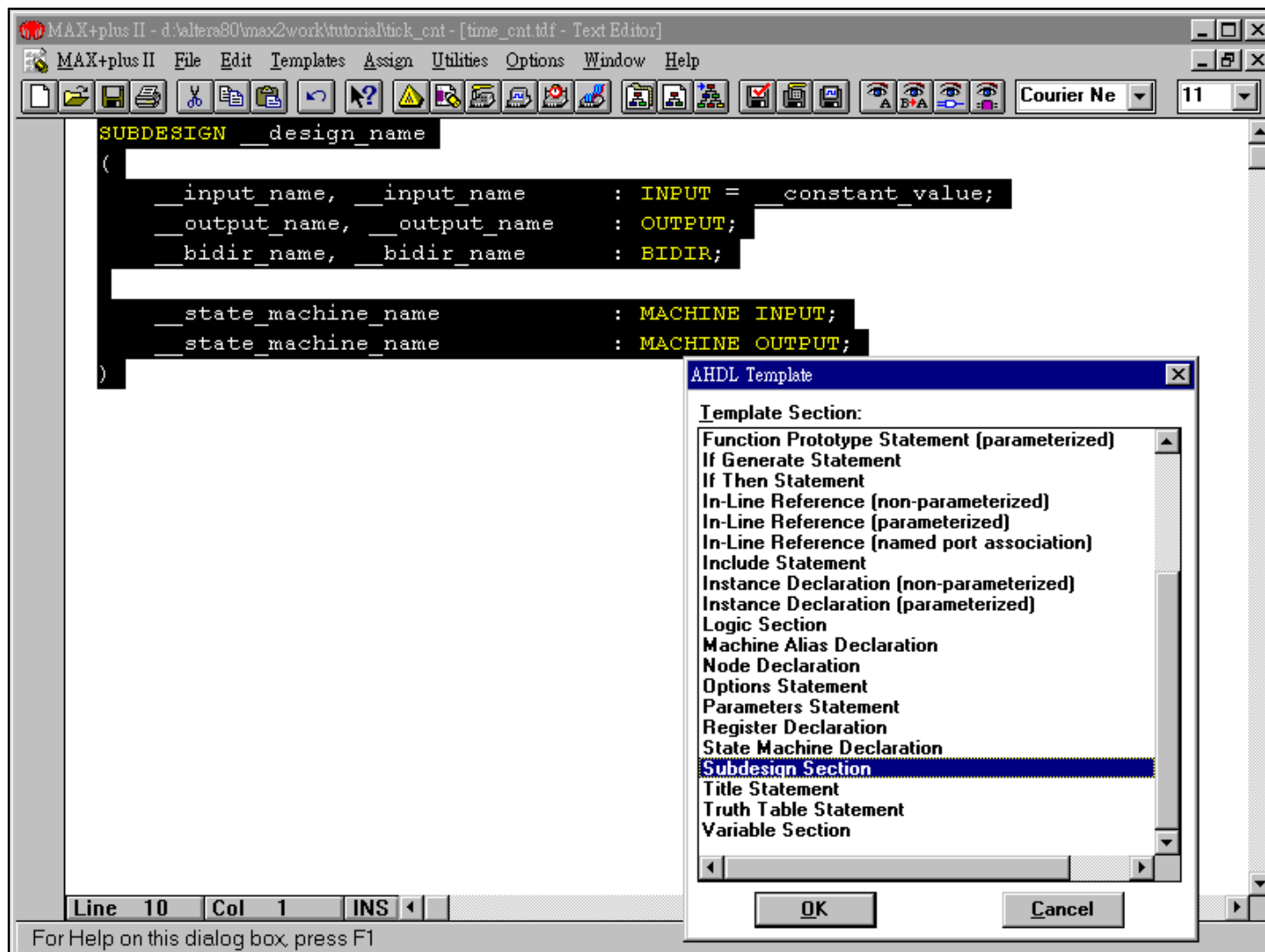
◆ AHDL templates make design easier

- 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

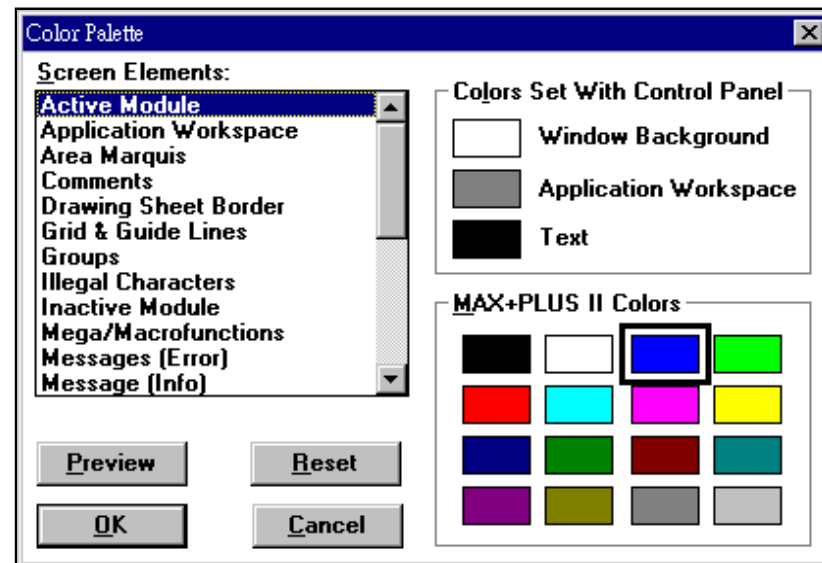
- To improve TDF readability & accuracy

Menu: *Options -> Syntax Coloring*

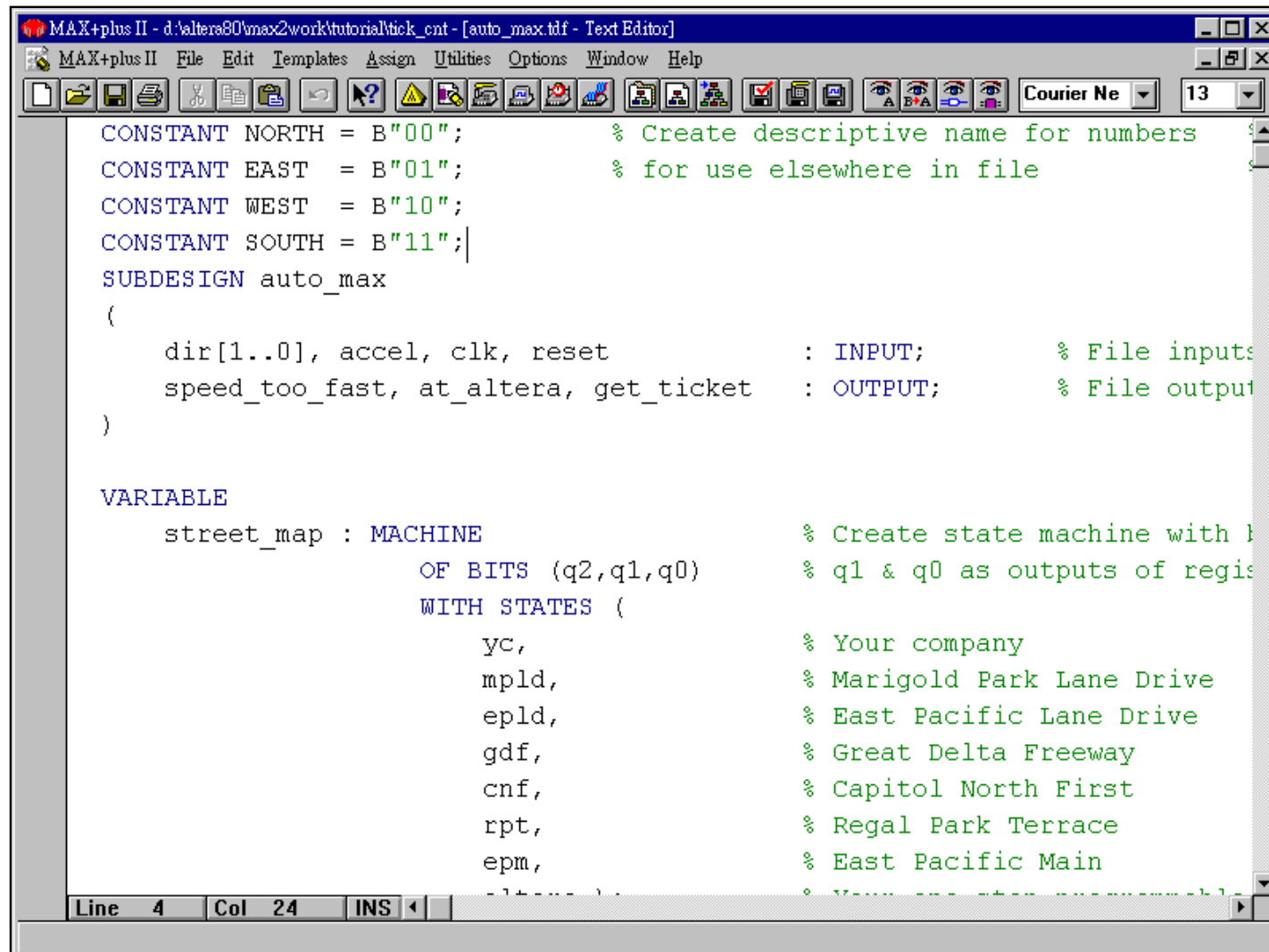
◆ To customize the color palette

Menu: *Options -> Color Palette...*

- The AHDL-relative options:
 - Comments
 - Illegal Characters
 - Megafunctions/Macrofunctions
 - Reserved Identifiers
 - Reserved Keywords
 - Strings
 - Text



Text Editor with Syntax Coloring



The screenshot shows the MAX+plus II Text Editor window. The title bar reads "MAX+plus II - d:\altera80\max2work\tutorial\tick_ent - [auto_max.tdf - Text Editor]". The menu bar includes "MAX+plus II", "File", "Edit", "Templates", "Assign", "Utilities", "Options", "Window", and "Help". The toolbar contains various icons for file operations and editing. The font is set to "Courier Ne" and the size is "13". The code is displayed with syntax coloring: keywords are blue, constants are green, and comments are green. The code includes constant declarations for directions, a subdesign declaration, input/output declarations, and a state machine definition.

```
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 outputs
)

VARIABLE
    street_map : MACHINE           % Create state machine with 1
                                % q1 & q0 as outputs of register
                                WITH STATES (
                                    yc,                 % Your company
                                    mp1d,               % Marigold Park Lane Drive
                                    ep1d,               % East Pacific Lane Drive
                                    gdf,                 % Great Delta Freeway
                                    cnf,                 % Capitol North First
                                    rpt,                 % Regal Park Terrace
                                    epm,                 % East Pacific Main
                                    altava,             % Your company name
                                )
```

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

```
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;
```

```
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

```
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;
```

```
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

```
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 LPM_MUX

```
FUNCTION lpm_mux (data[LPM_SIZE-1..0][LPM_WIDTH-1..0], sel[LPM_WIDTHHS-1..0])  
  WITH (LPM_WIDTH, LPM_SIZE, LPM_WIDTHHS, 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_WIDTHHS=2);  
END;
```

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

```
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)

```
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

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

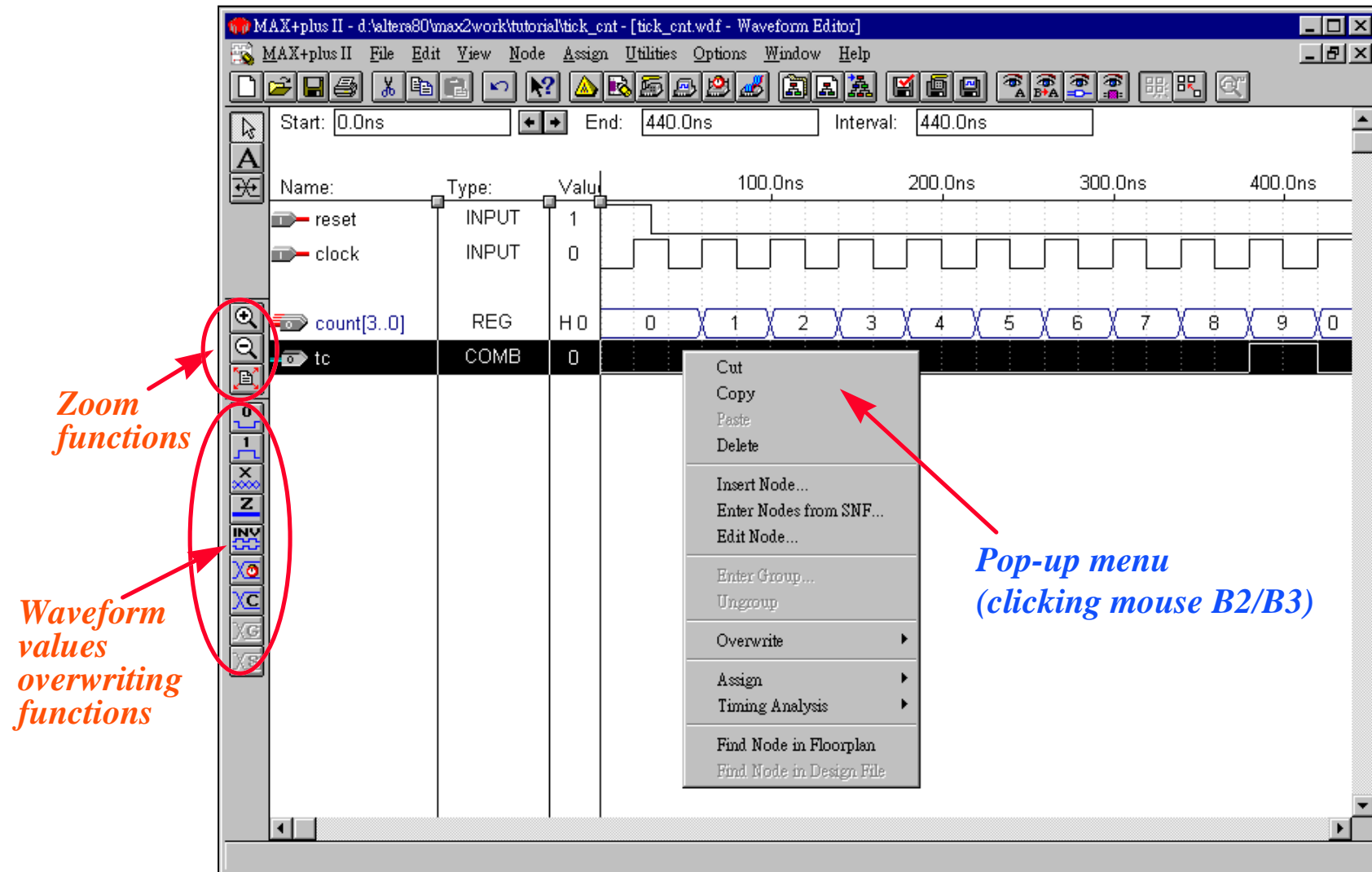
- Waveform design entry is best suited for circuits with well-defined sequential inputs & outputs, such as state machines, counters, and registers

◆ For design verification

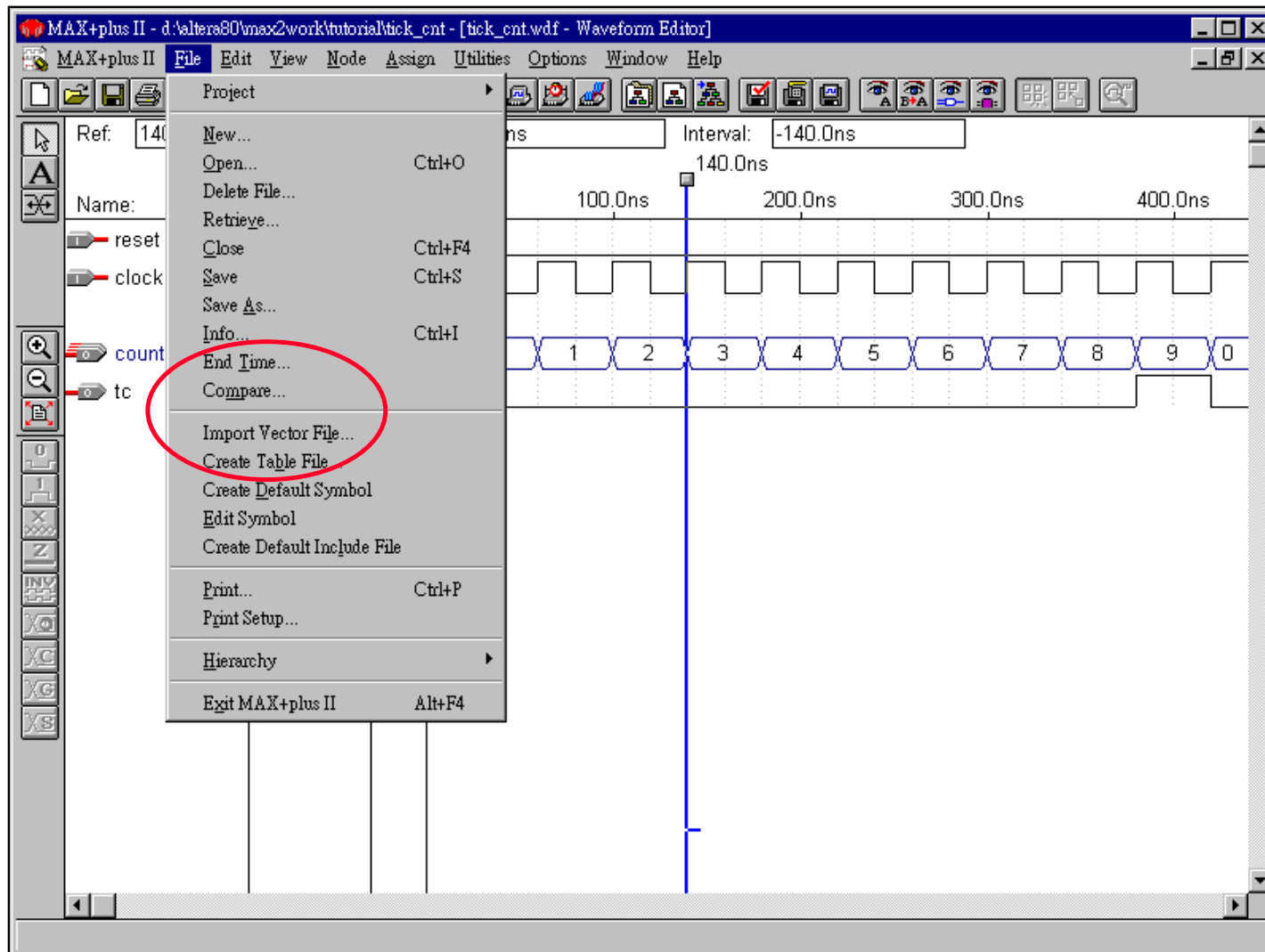
- Waveform Editor is a simulation pattern editor/viewer
- Waveform Editor is fully integrated with MAX+PLUS II Simulator & Programmer to provide full project verification flow

MAX+PLUS II

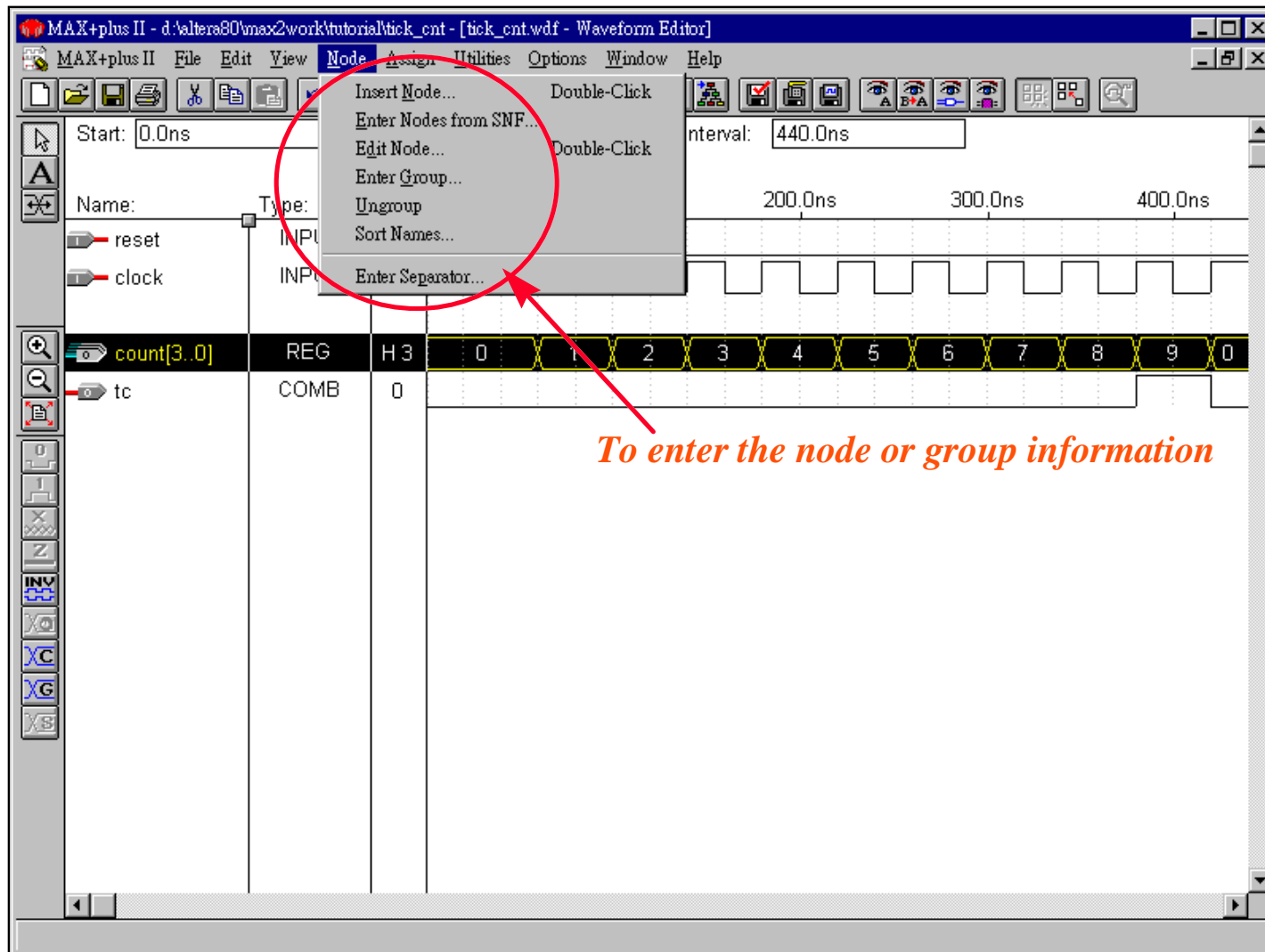
Waveform Design Environment



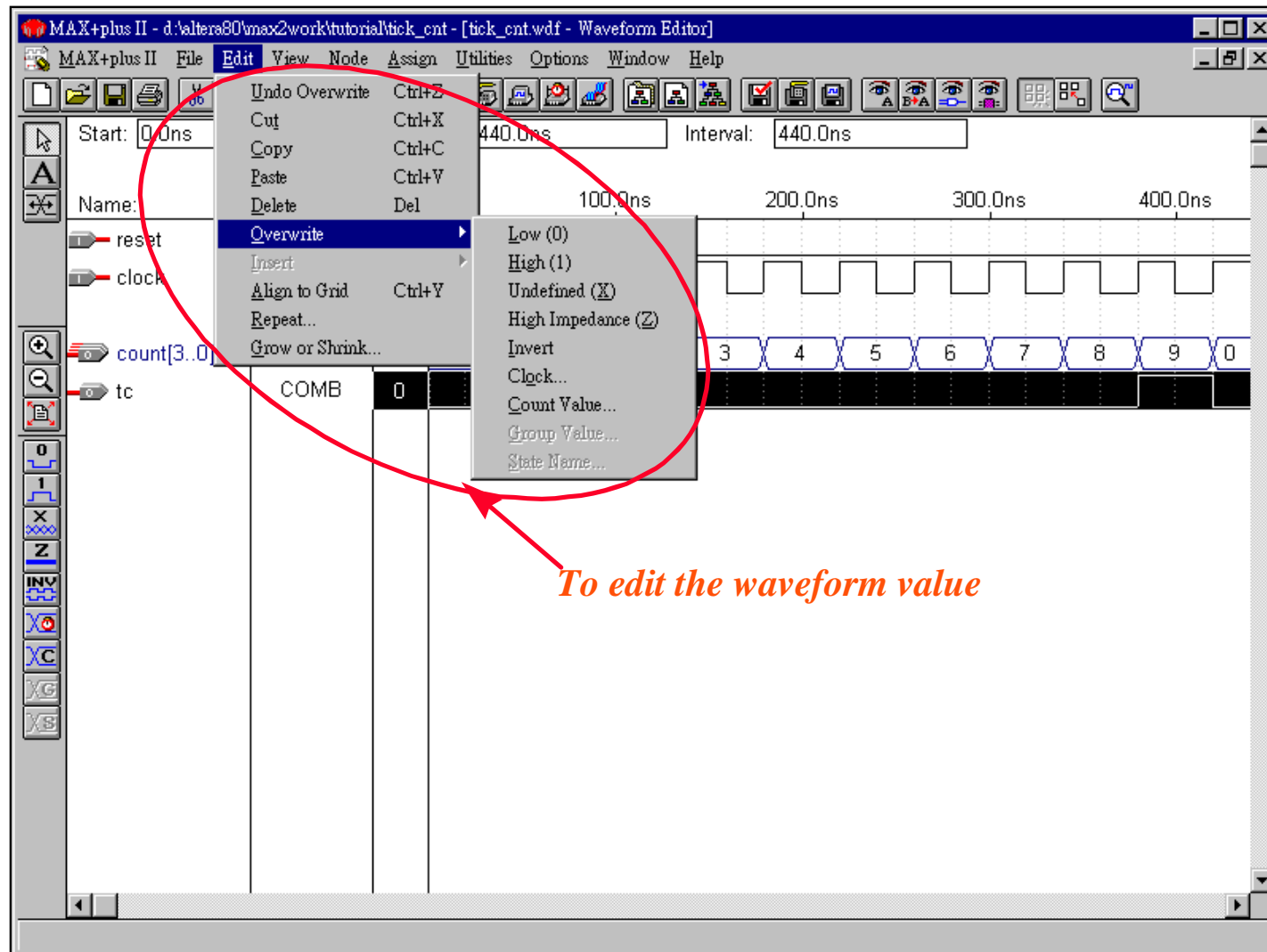
File Menu



Node Menu



Edit Menu



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*

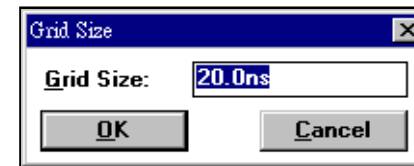
Setting Waveform Editor Options

◆ Set the grid size & show the grid

Menu: *Options -> Grid Size...*

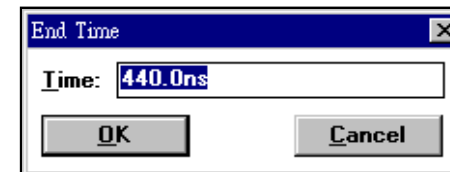
Menu: *Options -> Show Grid*

- 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

Insert Node

Node Name: count[3..0]

Default Value: 0

I/O Type

- ☐ Input Pin
- ☐ Output Pin
- ☒ Buried Node

OK Cancel

For Waveform Design File (WDF) Only

Node Type

- ☐ Pin Input
- ☒ Registered
- ☐ Combinatorial
- ☐ Machine

Secondary Inputs

Clock: clock

Reset: reset

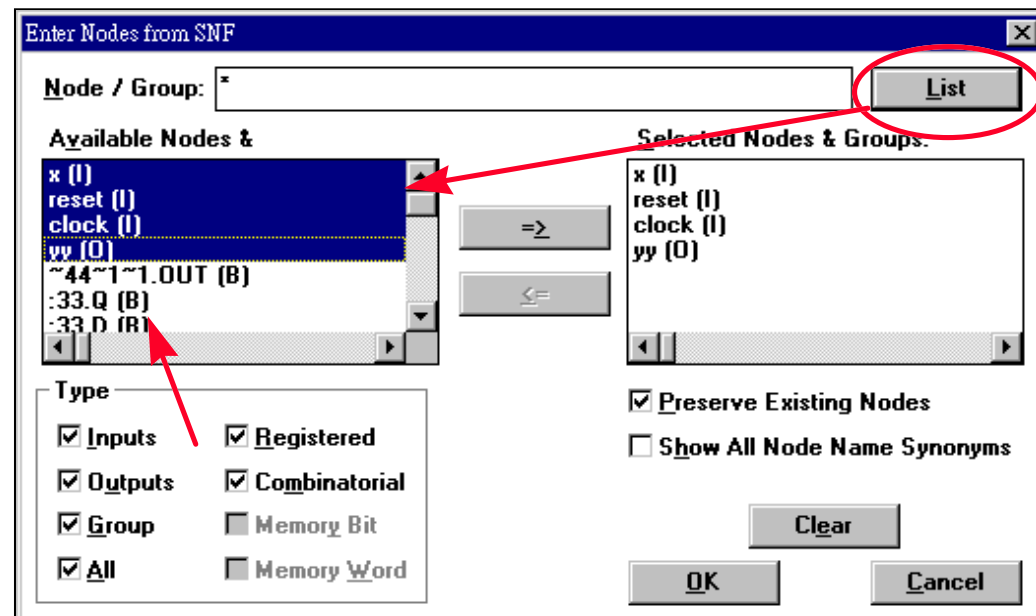
Preset: reset

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



Editing Waveforms - (1)

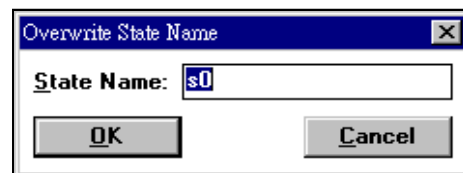
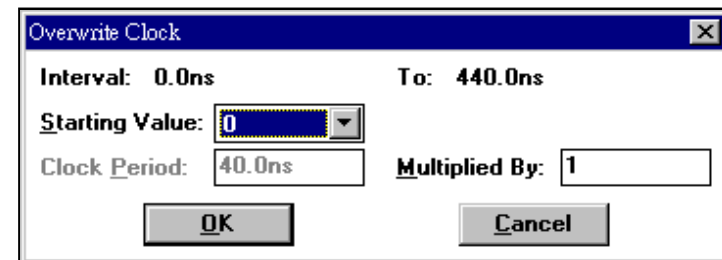
◆ Edit the waveforms

- 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...



Editing Waveforms - (2)

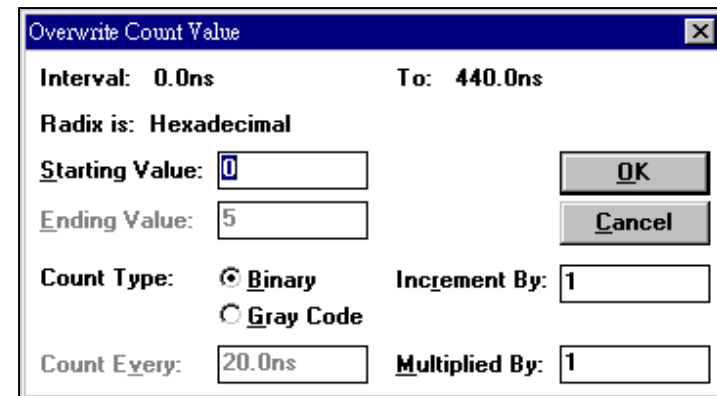
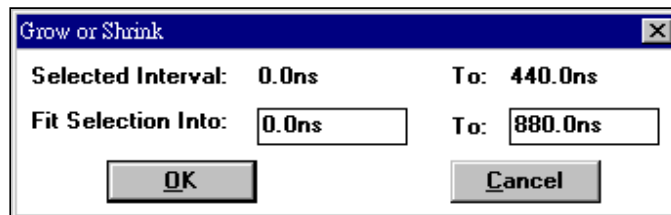
◆ Edit the waveforms

- 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...

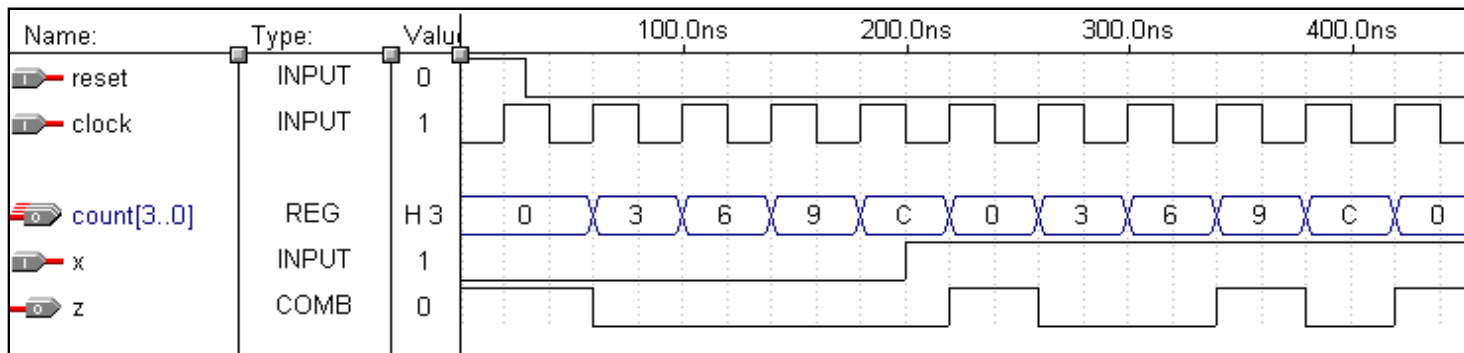
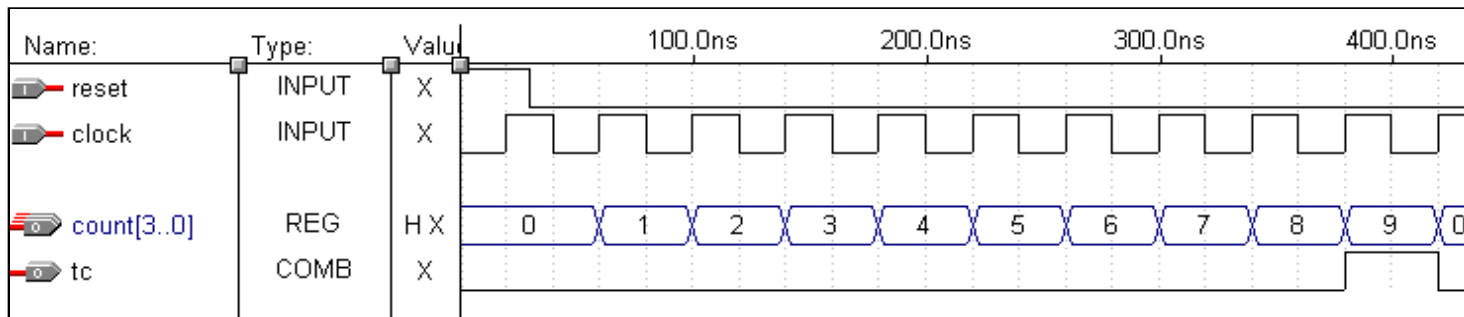
- Use "Overwrite Count Value" to help create all possible combinations of decoder input values, and then manually edit the output waveforms

Name:	Type:		50.0ns	100.0ns	150.0ns	200.0ns	250.0ns	300.0ns	350.0ns	400.0ns								
 in[3..0]	INPUT		F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
 decout[15..0]	COMB		8000	4000	2000	1000	0800	0400	0200	0100	0080	0040	0020	0010	0008	0004	0002	0001

Example: Counter

◆ When design a counter

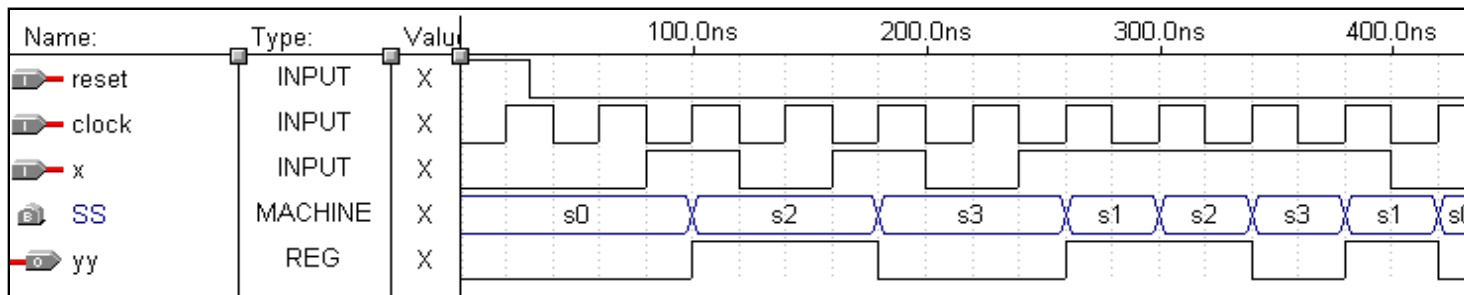
- Use "Overwrite Count Value" command to create a regular counter waveform



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 Implementation

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

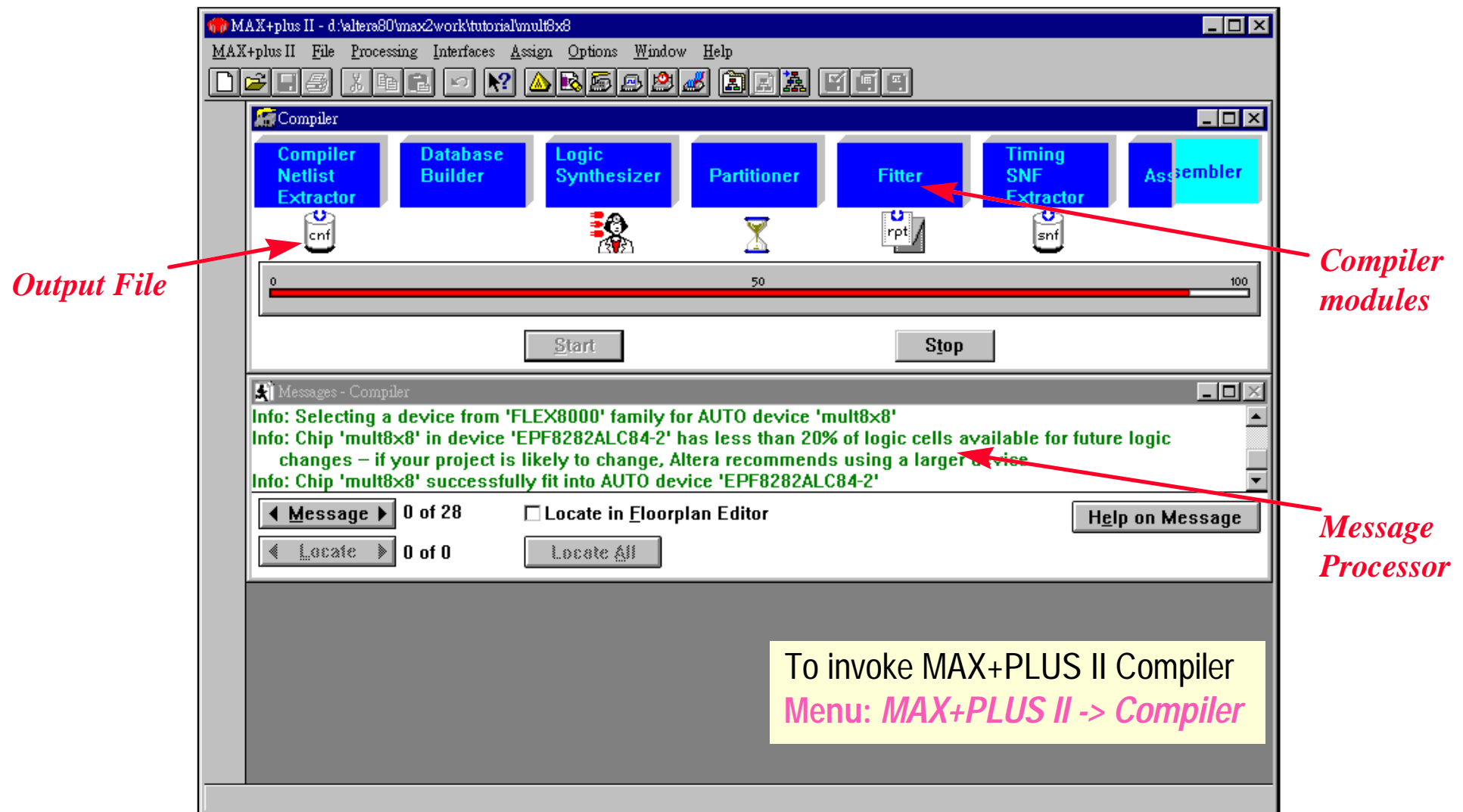
MAX+PLUS II Compiler

◆ MAX+PLUS II Compiler

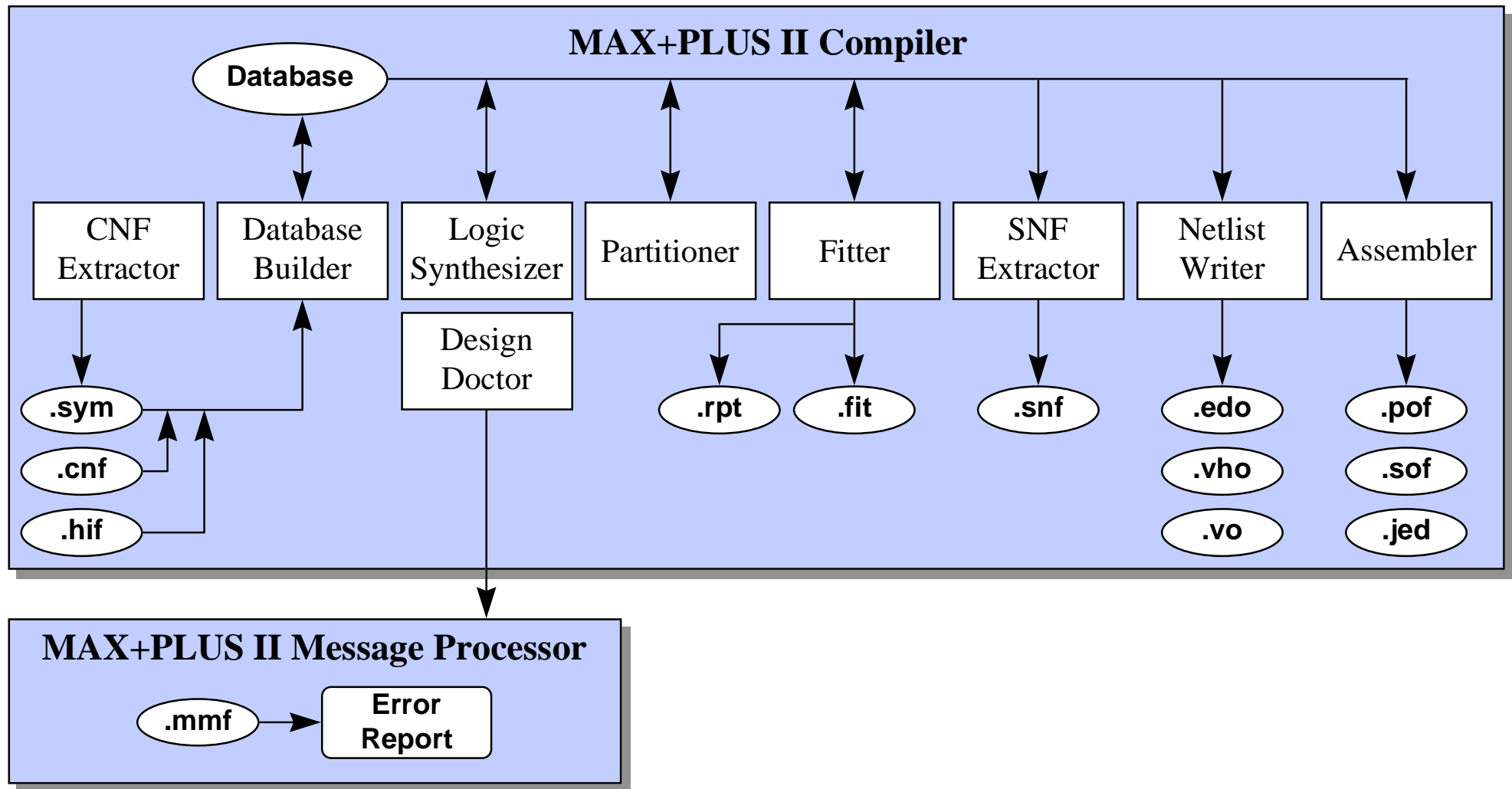
- MAX+PLUS II Compiler consists of a series of modules and a utility that check a project for errors, synthesize the logic, fit the project into one or more Altera devices, and generate output files for simulation, timing analysis, and device programming
- To invoke Compiler

Menu: MAX+PLUS II -> Compiler

MAX+PLUS II Compiler Window



MAX+PLUS II Compilation Flow



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

Compiler Modules - (7)

◆ 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

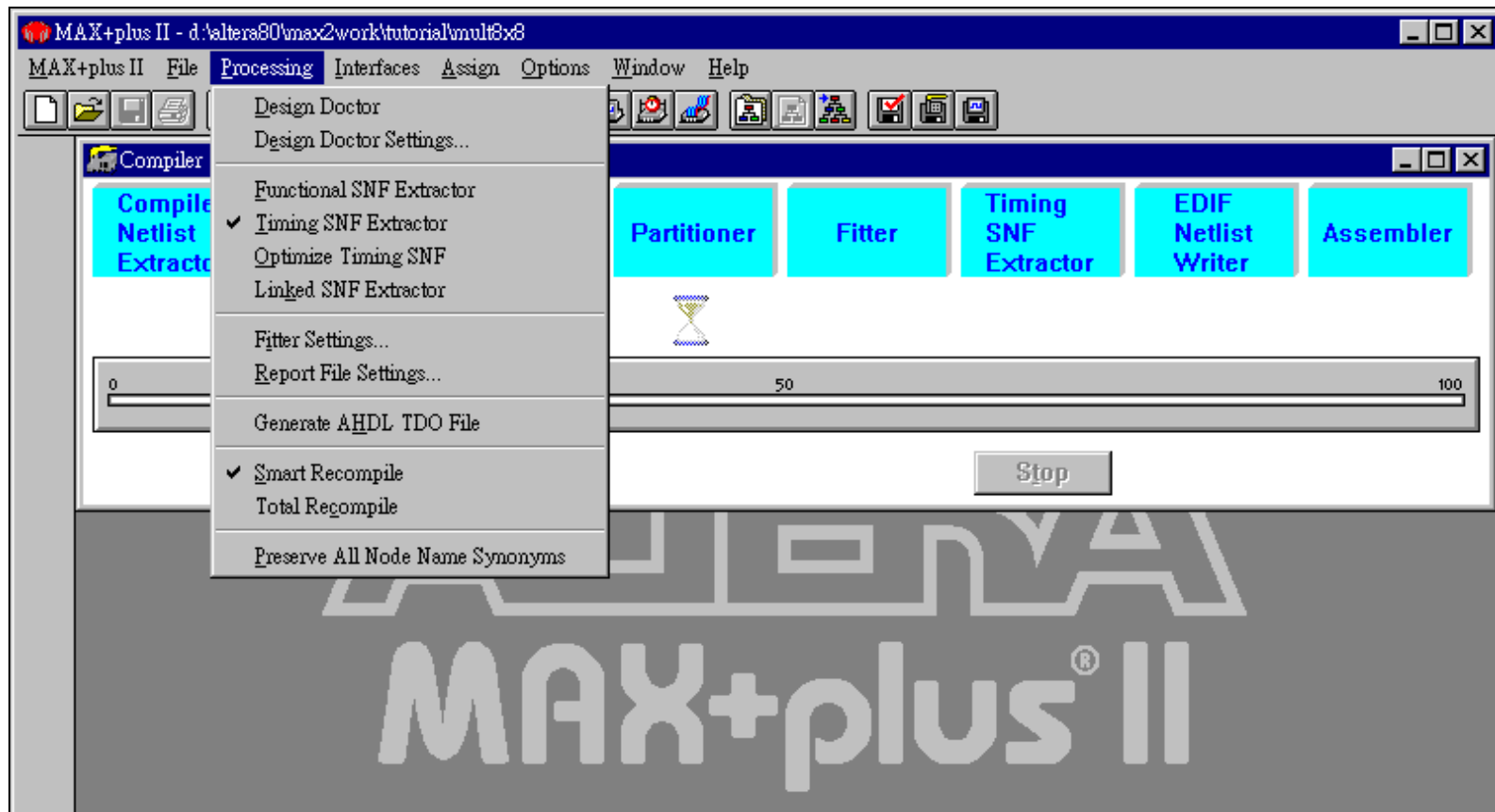
- The Compiler module that creates one or more programming files for programming or configuring the device(s) for a project
- The assembler generates one or more device programming files
 - POFs and JEDEC Files are always generated
 - SOFs, Hex Files, and TTFs are also generated if the project uses FLEX 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

Preparing for Compilation

◆ Preparing for Compilation

- Specify the project to the top level design file
- Customize design processing to your specification
 - Specify project logic synthesis style and settings
 - Specify timing requirements
 - Divide a large project into multiple devices
 - Set various device options
 - ...

Processing Menu

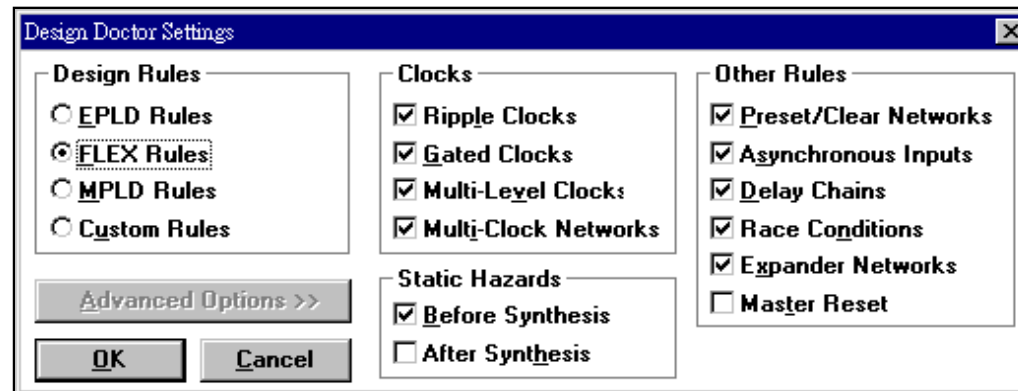


Compilation Process Settings - (1)

◆ Customize the Design Doctor utility

- Design Doctor option can be turned on if you want check each design file for logic that may cause system-level reliability problems
 - Design Doctor option is not turned on by default
- You can choose one of predefined sets of design rules (EPLD, FLEX or MPLD rules) or create a custom set of design rules

Menu: *Processing -> Design Doctor Setting...*

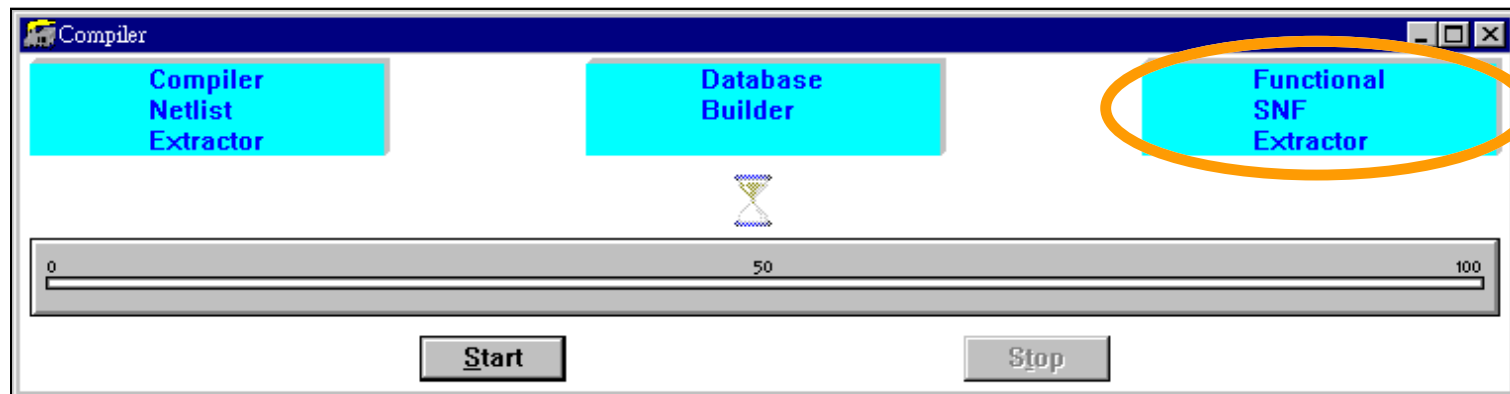


Compilation Process Settings - (2)

◆ Functional SNF extraction

- To create the functional SNF file required for functional simulation
- Only the Compiler Netlist Extractor, Database Builder, and Functional SNF Extractor modules run, shortening the processing time

Menu: Processing -> Functional SNF Extractor



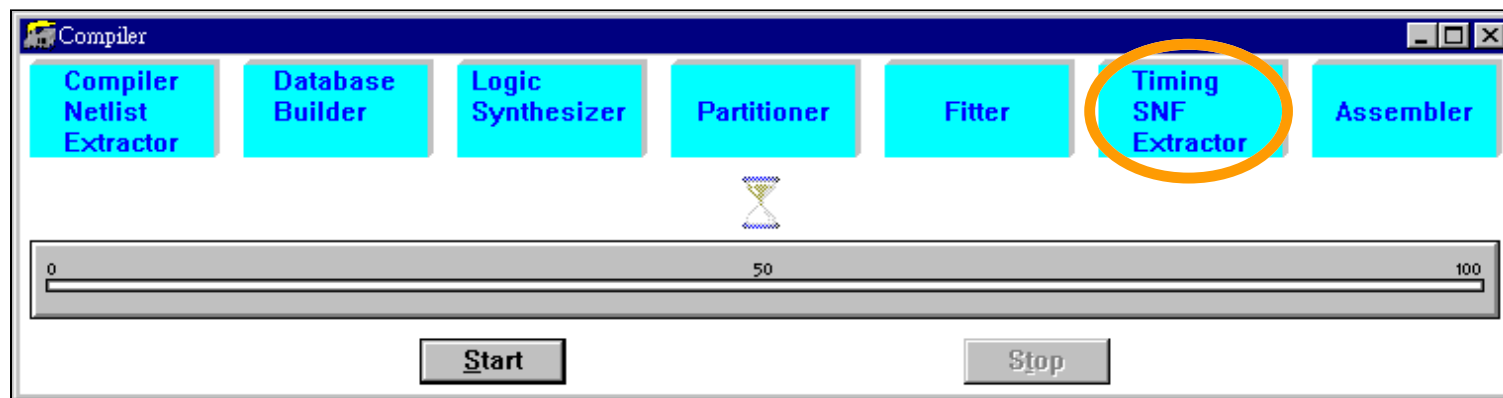
Compilation Process Settings - (3)

◆ Timing SNF extraction

- "Timing SNF Extractor" option must be turned on for further timing analysis & timing simulation
- You can instruct the Compiler to generate an optimized SNF. It will increase the compilation time, but save your time during timing simulation & timing analysis.

Menu: *Processing -> Timing SNF Extractor*

Menu: *Processing -> Optimize Timing SNF*

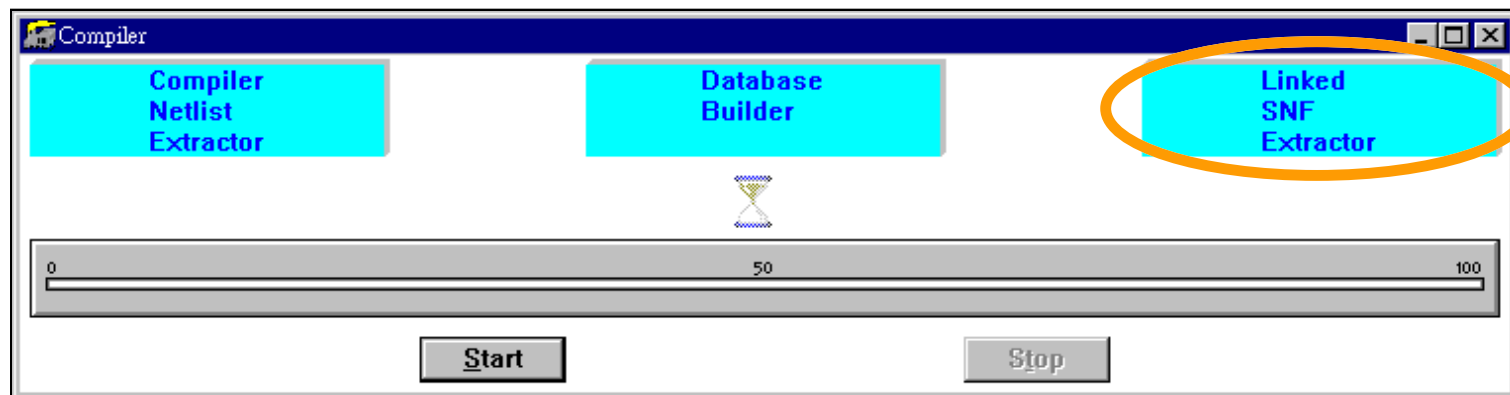


Compilation Process Settings - (4)

◆ Linked SNF extraction

- You can create a super-project consisting of a top-level design file that contains symbols or instances representing multiple individual sub-projects and then compile the super-project to create a linked SNF for simulation and timing analysis
- Only the Compiler Netlist Extractor, Database Builder, and Linked SNF Extractor modules run, shortening the processing time

Menu: *Processing -> Linked SNF Extractor*

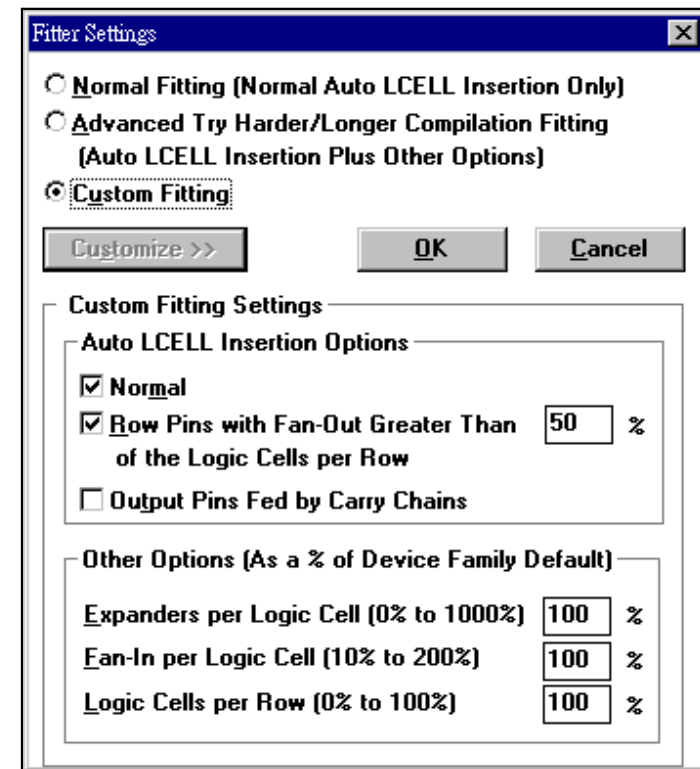


Compilation Process Settings - (5)

◆ Customize the Fitter settings

- To customize the fitting techniques to help achieve a fit, for example, by automatically inserting logic cells or limiting fan-in
 - Default is “Normal Fitting”
- Many of the Fitter settings improve project fitting through automatic LCELL insertion. However, inserting LCELL buffers in a project increases its logic cell usage and slows its performance.

Menu: *Processing -> Fitter Settings...*

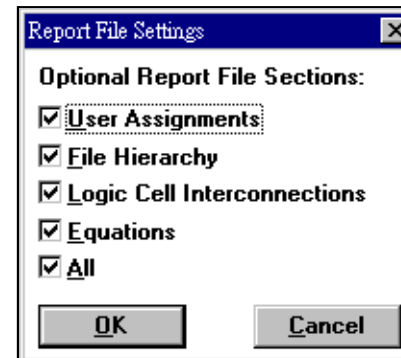


Compilation Process Settings - (6)

◆ Customize the report file settings

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

Menu: *Processing -> Report File Settings...*



Compilation Process Settings - (7)

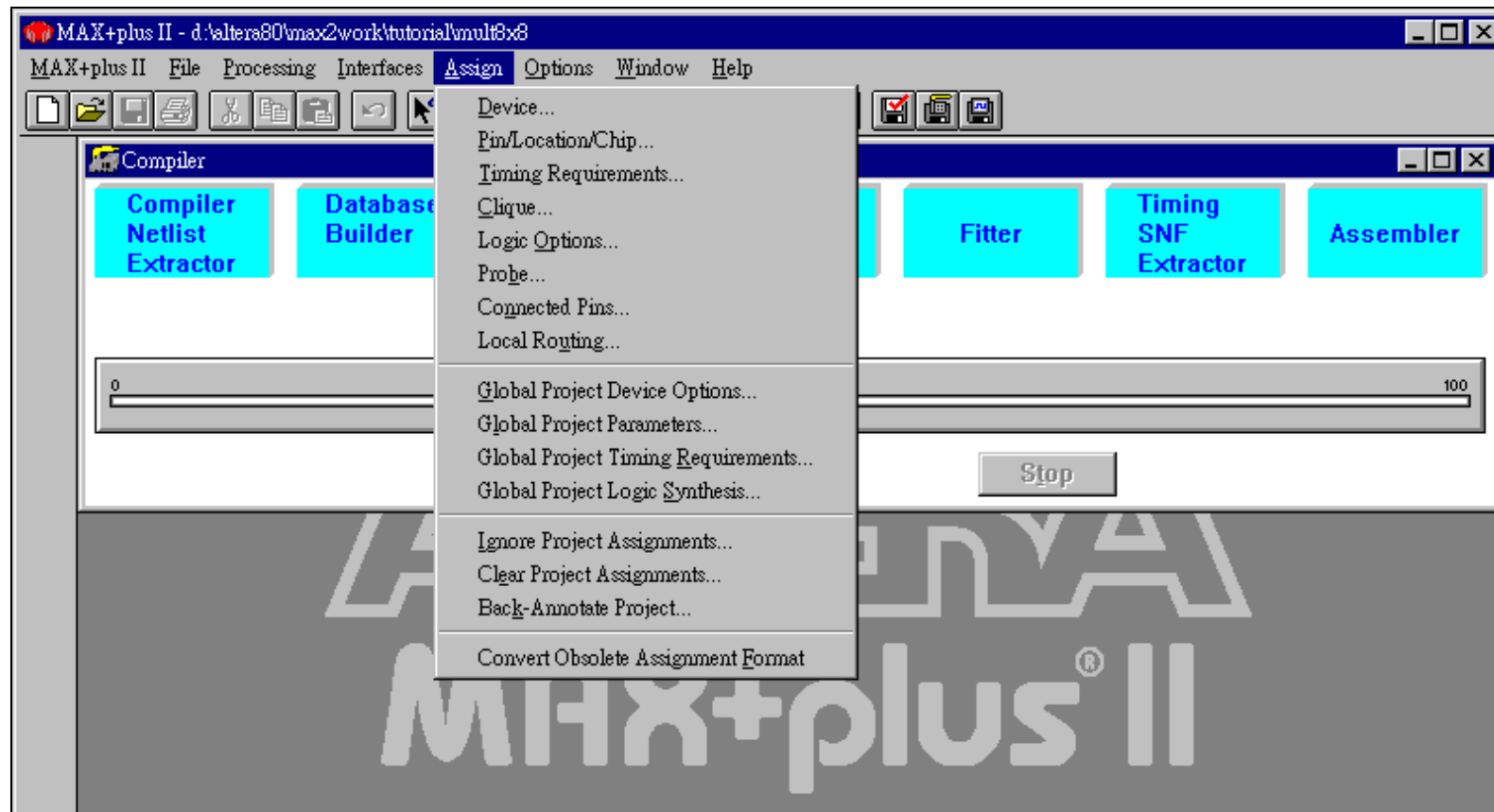
◆ “Smart Recompile” & “Total Recompile”

- 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



MAX+PLUS II Assignments - (1)

◆ Device assignment

- To assign project logic to a device.
- In a multi-device project, device assignments map chip assignments to specific devices

◆ Pin assignment

- To assign the input or output of a single logic function to a specific pin, row, or column within a chip

◆ Location assignment

- To assign a single logic function to a specific location within a chip, such as a logic cell, I/O cell, embedded cell, LAB, EAB, row, or column.

◆ Chip assignment

- To specify which logic functions must be implemented in the same device when a project is partitioned into multiple devices

MAX+PLUS II Assignments - (2)

◆ Clique assignment

- To specify which logic functions must remain together
- Grouping logic functions into a clique helps ensure that they are implemented in the same LAB, EAB, row, or device

◆ Connected pin assignment

- To specify how two or more pins are connected externally on your board
- This information is also useful for timing simulation and multi-project simulation

◆ Probe assignment

- To assign an easy-to-remember, unique name to an input or output of a logic function

MAX+PLUS II Assignments - (3)

◆ Timing assignment

- To guide logic synthesis & fitting on individual logic functions to achieve the desired performance
 - t_{PD} : input to non-registered output delay
 - t_{CO} : clock to output delay
 - t_{SU} : clock setup time
 - f_{MAX} : clock frequency

◆ Logic option assignment

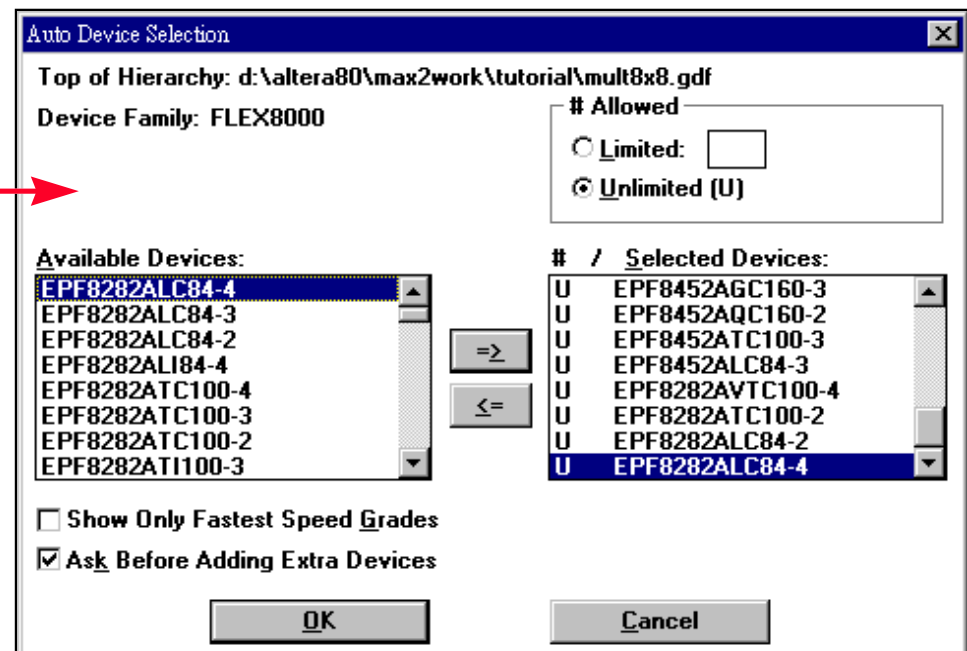
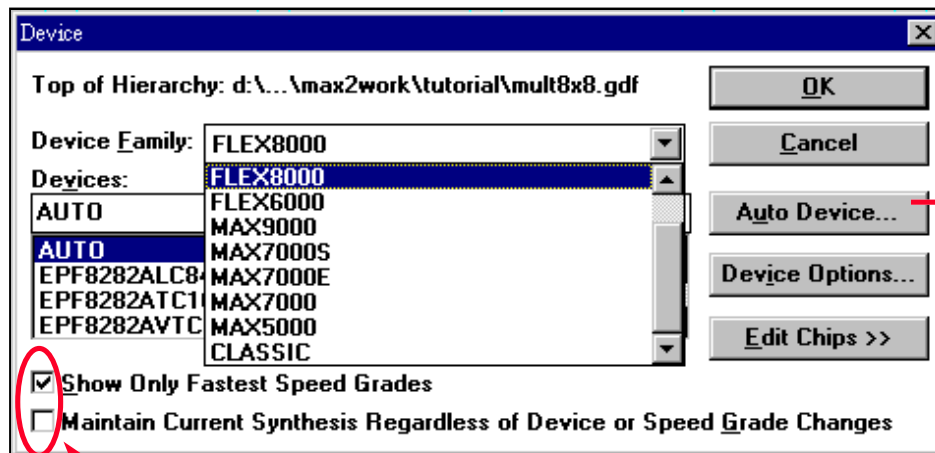
- To guide logic synthesis on individual logic functions during compilation with logic synthesis style and/or individual logic synthesis options

Device Assignments

◆ To select device family

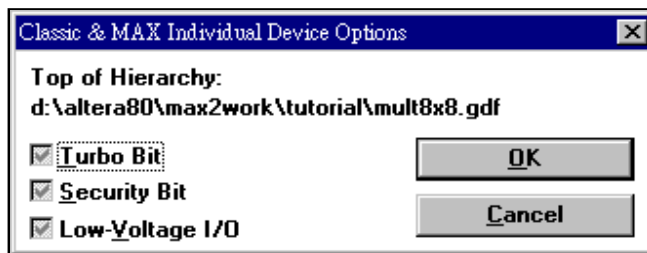
- You can select a specific device for your project
- Or you can let Compiler select device(s) automatically (Auto Device)
- You can specify individual device options or global device options

Menu: *Assign -> Device...*

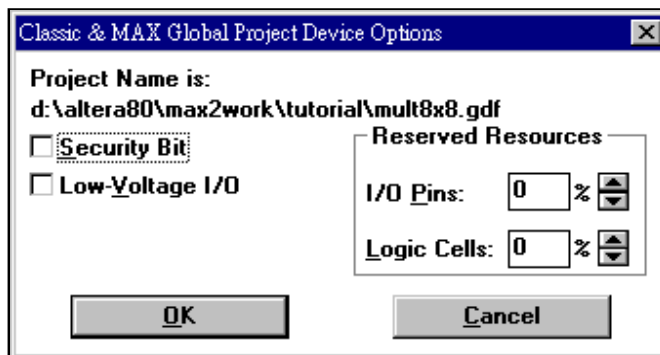


Device Options - MAX 7000/E & MAX 9000

Menu: *Assign -> Device... -> Device Options...*

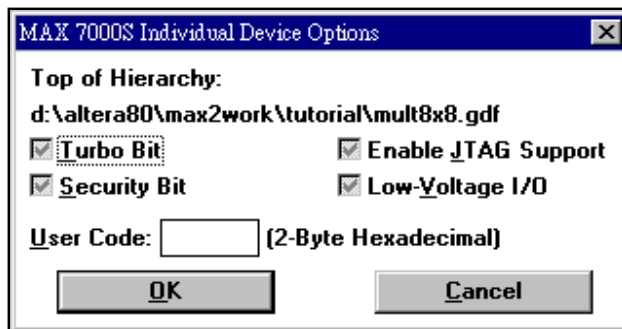


Menu: *Assign -> Global Project Device Options...*

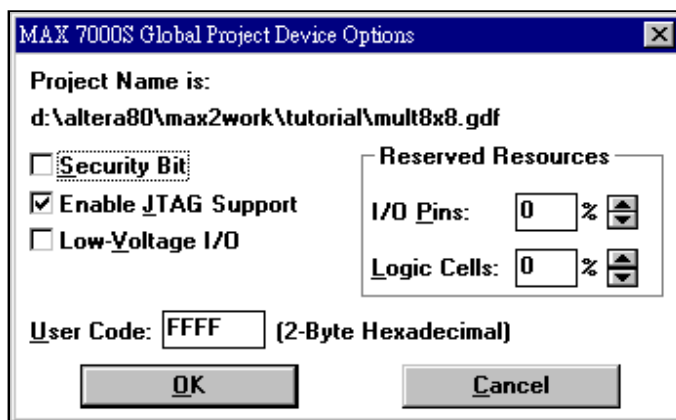


Device Options - MAX 7000S

Menu: *Assign -> Device... -> Device Options...*



Menu: *Assign -> Global Project Device Options...*



Device Options - FLEX 8000A

Menu: *Assign -> Device... -> Device Options...*

Menu: *Assign -> Global Project Device Options...*

FLEX 8000 Individual Device Options

Top of Hierarchy: d:\altera80\max2work\tutorial\mult8x8.gdf

Device Options

☒ User-Supplied Start-Up Clock (CLKUSR)

☒ Auto-Restart Configuration on Frame Error

☒ Release Clears Before Tri-States

☒ Enable DCLK Output in User Mode

☒ Disable Start-Up Time-Out

☒ Low-Voltage I/O

☒ Enable JTAG Support

Dual-Purpose Configuration Pins

Configuration Scheme: Use Global Project Setting

Pin:	Reserve:	Tri-State:	Pin:	Reserve:	Tri-State:
Data0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add[0..12]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nWS, nRS, nCS, CS	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add13	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Data[1..7]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add14	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
RDYnBUSY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add15	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Rdclk	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add16	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SDOUT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add17	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Not Affected By Configuration Scheme:			CLKUSR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

FLEX 8000 Global Project Device Options

Project Name is: d:\altera80\max2work\tutorial\mult8x8.gdf

Device Options

☐ User-Supplied Start-Up Clock (CLKUSR)

☐ Auto-Restart Configuration on Frame Error

☐ Release Clears Before Tri-States

☐ Enable DCLK Output in User Mode

☐ Disable Start-Up Time-Out

☐ Low-Voltage I/O

☐ Enable JTAG Support

Reserved Resources

I/O Pins: 0 %

Logic Cells: 0 %

Dual-Purpose Configuration Pins

Configuration Scheme: Active Serial (can use Configuration EPROM)

Pin:	Reserve:	Tri-State:	Pin:	Reserve:	Tri-State:
Data0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Add[0..12]	<input type="checkbox"/>	<input type="checkbox"/>
nWS, nRS, nCS, CS	<input type="checkbox"/>	<input type="checkbox"/>	Add13	<input type="checkbox"/>	<input type="checkbox"/>
Data[1..7]	<input type="checkbox"/>	<input type="checkbox"/>	Add14	<input type="checkbox"/>	<input type="checkbox"/>
RDYnBUSY	<input type="checkbox"/>	<input type="checkbox"/>	Add15	<input type="checkbox"/>	<input type="checkbox"/>
Rdclk	<input type="checkbox"/>	<input type="checkbox"/>	Add16	<input type="checkbox"/>	<input type="checkbox"/>
SDOUT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Add17	<input type="checkbox"/>	<input type="checkbox"/>
Not Affected By Configuration Scheme:			CLKUSR	<input type="checkbox"/>	<input type="checkbox"/>

Device Options - FLEX 10K/A

Menu: *Assign -> Device... -> Device Options...*

Menu: *Assign -> Global Project Device Options...*

FLEX 10K Individual Device Options

Top of Hierarchy: d:\altera80\max2work\tutorial\mult8x8.gdf

Device Options

- ☒ User-Supplied Start-Up Clock (CLKUSR)
- ☒ Auto-Restart Configuration on Frame Error
- ☒ Release Clears Before Tri-States
- ☒ Enable Chip-Wide Reset (DEV_CLRn)
- ☒ Enable Chip-Wide Output Enable (DEV_OE)
- ☒ Enable INIT_DONE Output
- ☒ Enable LOCK Output
- ☒ Low-Voltage I/O
- ☒ Use Low-Voltage Configuration EPROM

Dual-Purpose Configuration Pins

Configuration Scheme: Use Global Project Setting

Pin:	Reserve:	Tri-State:	Pin:	Reserve:	Tri-State:
nWS, nRS, nCS, CS	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RDYnBUSY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Data[1..7]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
Not Affected By Configuration Scheme:			CLKUSR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

JTAG User Code: (00 to 7F Hexadecimal)

OK Cancel

FLEX 10K Global Project Device Options

Project Name is: d:\altera80\max2work\tutorial\mult8x8.gdf

Device Options

- ☐ User-Supplied Start-Up Clock (CLKUSR)
- ☐ Auto-Restart Configuration on Frame Error
- ☐ Release Clears Before Tri-States
- ☐ Enable Chip-Wide Reset (DEV_CLRn)
- ☐ Enable Chip-Wide Output Enable (DEV_OE)
- ☐ Enable INIT_DONE Output
- ☐ Enable LOCK Output
- ☐ Low-Voltage I/O
- ☐ Use Low-Voltage Configuration EPROM

Reserved Resources

I/O Pins: %

Logic Cells: %

Dual-Purpose Configuration Pins

Configuration Scheme: Passive Serial (can use Configuration EPROM)

Pin:	Reserve:	Tri-State:	Pin:	Reserve:	Tri-State:
nWS, nRS, nCS, CS	<input type="checkbox"/>	<input type="checkbox"/>	RDYnBUSY	<input type="checkbox"/>	<input type="checkbox"/>
Data[1..7]	<input type="checkbox"/>	<input type="checkbox"/>			
Not Affected By Configuration Scheme:			CLKUSR	<input type="checkbox"/>	<input type="checkbox"/>

JTAG User Code: 7F (00 to 7F Hexadecimal)

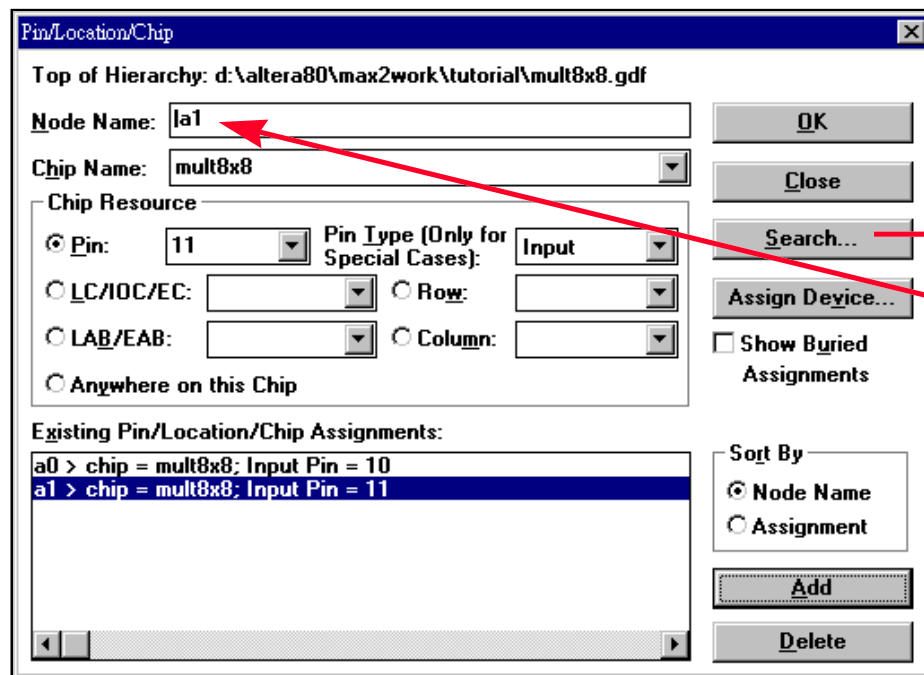
OK Cancel

Pin/Location/Chip Assignments

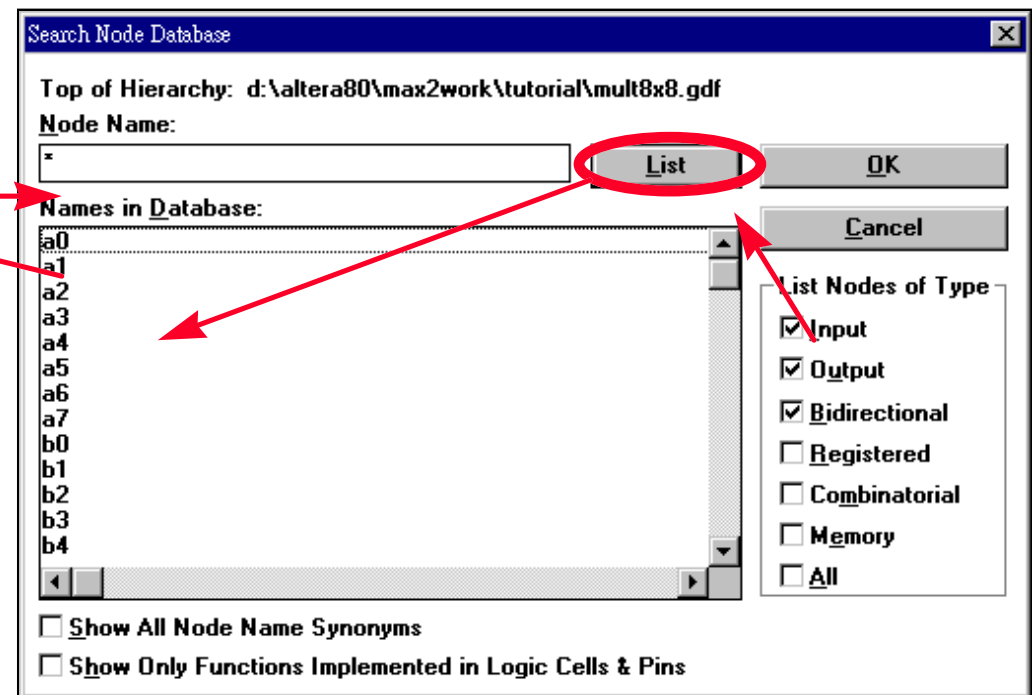
◆ To specify pin/location/chip assignments

- You must specify a specific device before specifying pin/location/chip assignments
- Click the "Search" button to help you list the nodes of your design

Menu: *Assign -> Pin/Location/Chip...*



The Pin/Location/Chip dialog box is shown. It has a title bar 'Pin/Location/Chip'. The 'Top of Hierarchy' is 'd:\altera80\max2work\tutorial\mult8x8.gdf'. The 'Node Name' field contains 'la1'. The 'Chip Name' dropdown shows 'mult8x8'. Under 'Chip Resource', 'Pin' is selected with value '11', and 'Pin Type' is 'Input'. There are buttons for 'OK', 'Close', 'Search...', and 'Assign Device...'. A 'Show Buried Assignments' checkbox is present. At the bottom, there is a list of 'Existing Pin/Location/Chip Assignments' with two entries: 'a0 > chip = mult8x8; Input Pin = 10' and 'a1 > chip = mult8x8; Input Pin = 11'. There are 'Sort By' options for 'Node Name' and 'Assignment', and 'Add' and 'Delete' buttons.



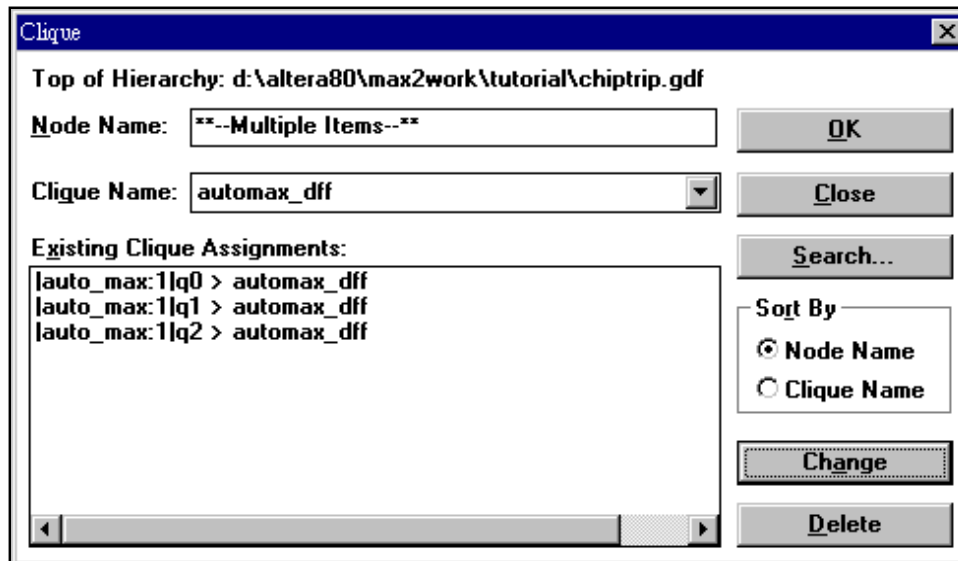
The Search Node Database dialog box is shown. It has a title bar 'Search Node Database'. The 'Top of Hierarchy' is 'd:\altera80\max2work\tutorial\mult8x8.gdf'. The 'Node Name' field contains '*'. The 'List' button is circled in red. Below the 'Node Name' field is a list of 'Names in Database' containing 'a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'b0', 'b1', 'b2', 'b3', 'b4'. To the right of the list is a 'List Nodes of Type' section with checkboxes for 'Input', 'Output', 'Bidirectional', 'Registered', 'Combinatorial', 'Memory', and 'All'. At the bottom, there are checkboxes for 'Show All Node Name Synonyms' and 'Show Only Functions Implemented in Logic Cells & Pins'. Buttons for 'OK' and 'Cancel' are on the right.

Clique Assignments

◆ To specify clique assignments

- You can define one or more selected logic functions as part of a single unit to be fitted into the same LAB, row, or chip
 - You cannot assign pins to a clique

Menu: *Assign -> Clique...*



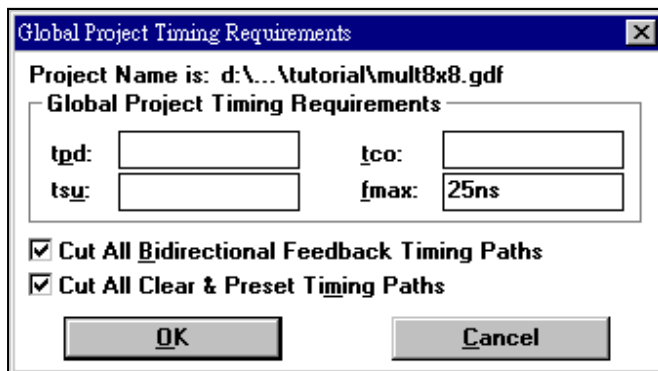
Individual Timing Assignments

◆ To specify timing requirements for FLEX designs

- 4 types of timing requirements: t_{PD} , t_{CO} , t_{SU} , f_{MAX}
- Do not give unreasonable global timing constraint

Menu: *Assign -> Global Project Timing Requirements...*

Menu: *Assign -> Timing Requirements...*



Global Project Timing Requirements

Project Name is: d:\...\tutorial\mult8x8.gdf

Global Project Timing Requirements

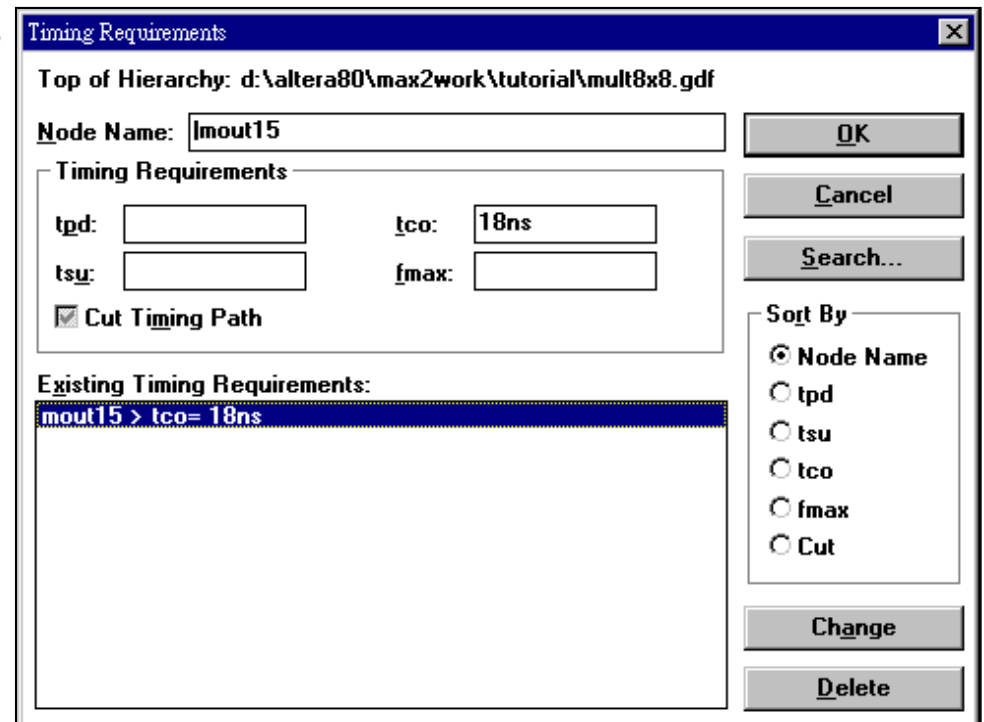
tpd: tco:

tsu: fmax: 25ns

☒ Cut All Bidirectional Feedback Timing Paths

☒ Cut All Clear & Preset Timing Paths

OK Cancel



Timing Requirements

Top of Hierarchy: d:\altera80\max2work\tutorial\mult8x8.gdf

Node Name: mout15

Timing Requirements

tpd: tco: 18ns

tsu: fmax:

☒ Cut Timing Path

Existing Timing Requirements:

mout15 > tco= 18ns

Sort By

☒ Node Name

☐ tpd

☐ tsu

☐ tco

☐ fmax

☐ Cut

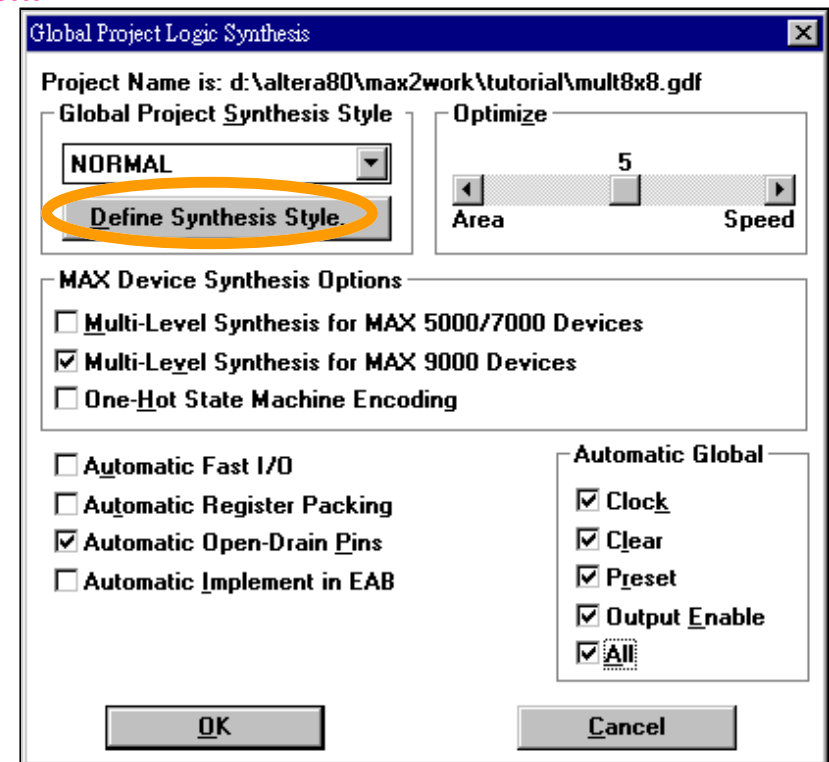
Change Delete

Global Synthesis Style Settings

◆ To select global synthesis options

- You can specify a speed / area optimization preference
- You can specify pre-defined or customized synthesis style

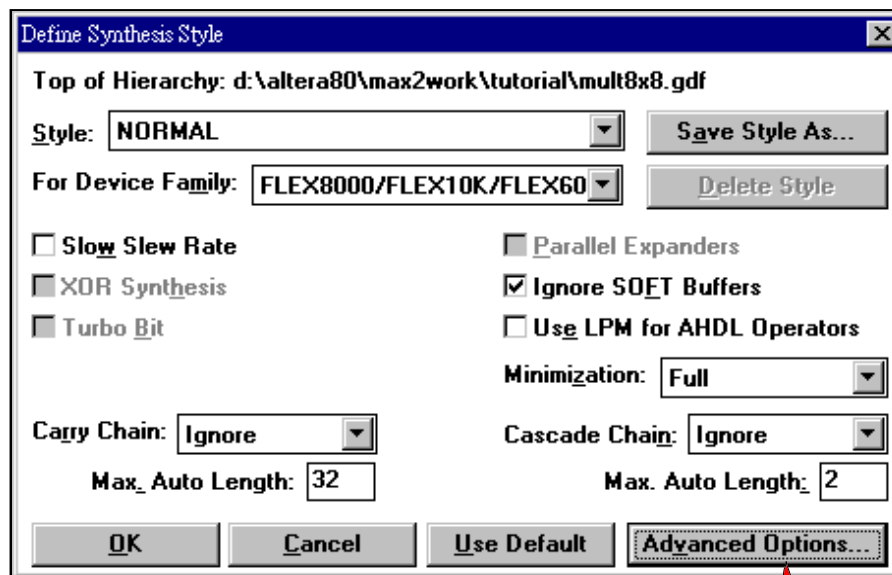
Menu: Assign -> Global Project Logic Synthesis...



Pre-Defined Synthesis Style - Normal

◆ “Normal” synthesis style

- Optimize your project logic for minimum resource usage



Define Synthesis Style

Top of Hierarchy: d:\altera80\max2work\tutorial\mult8x8.gdf

Style: **NORMAL** [Save Style As...]

For Device Family: FLEX8000/FLEX10K/FLEX60 [Delete Style]

☐ Slow Slew Rate ☐ Parallel Expanders

☐ XOR Synthesis ☒ Ignore SOFT Buffers

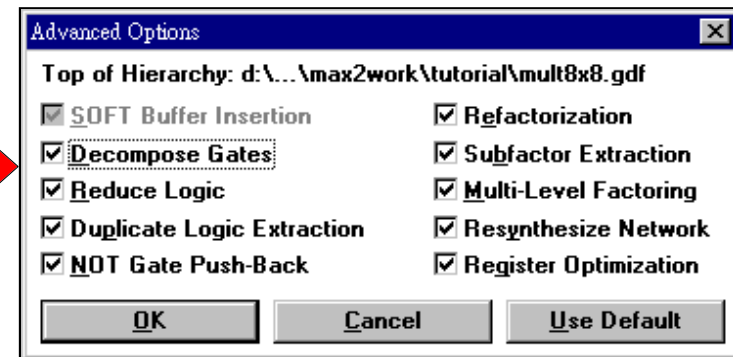
☐ Turbo Bit ☐ Use LPM for AHDL Operators

Minimization: Full

Carry Chain: Ignore Cascade Chain: Ignore

Max Auto Length: 32 Max Auto Length: 2

[OK] [Cancel] [Use Default] [Advanced Options...]



Advanced Options

Top of Hierarchy: d:\... \max2work\tutorial\mult8x8.gdf

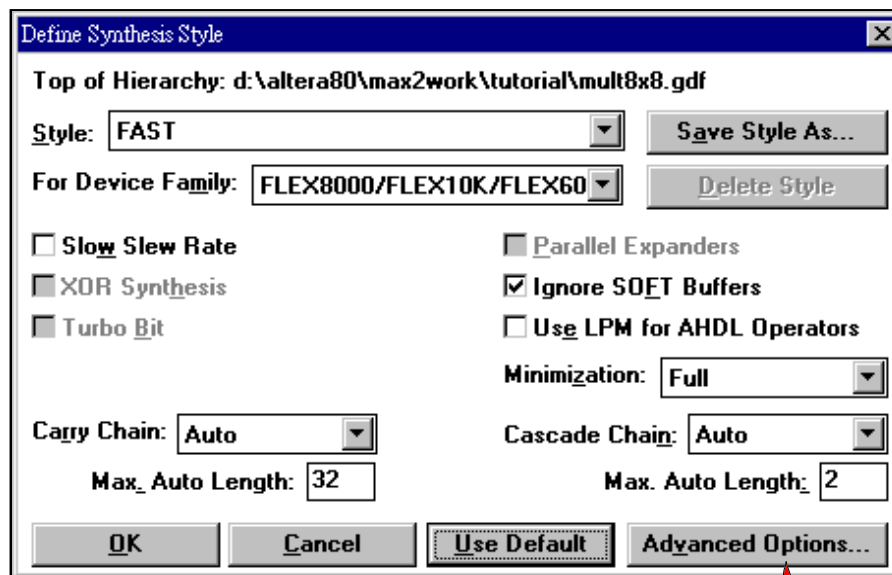
<input checked="" type="checkbox"/> SOFT Buffer Insertion	<input checked="" type="checkbox"/> Refactorization
<input checked="" type="checkbox"/> Decompose Gates	<input checked="" type="checkbox"/> Subfactor Extraction
<input checked="" type="checkbox"/> Reduce Logic	<input checked="" type="checkbox"/> Multi-Level Factoring
<input checked="" type="checkbox"/> Duplicate Logic Extraction	<input checked="" type="checkbox"/> Resynthesize Network
<input checked="" type="checkbox"/> NOT Gate Push-Back	<input checked="" type="checkbox"/> Register Optimization

[OK] [Cancel] [Use Default]

Pre-Defined Synthesis Style - Fast

◆ “Fast” synthesis style

- Optimize your project logic for maximum speed



Define Synthesis Style

Top of Hierarchy: d:\altera80\max2work\tutorial\mult8x8.gdf

Style: FAST

For Device Family: FLEX8000/FLEX10K/FLEX60

☐ Slow Slew Rate

☐ XOR Synthesis

☐ Turbo Bit

☐ Parallel Expanders

☒ Ignore SOFT Buffers

☐ Use LPM for AHDL Operators

Minimization: Full

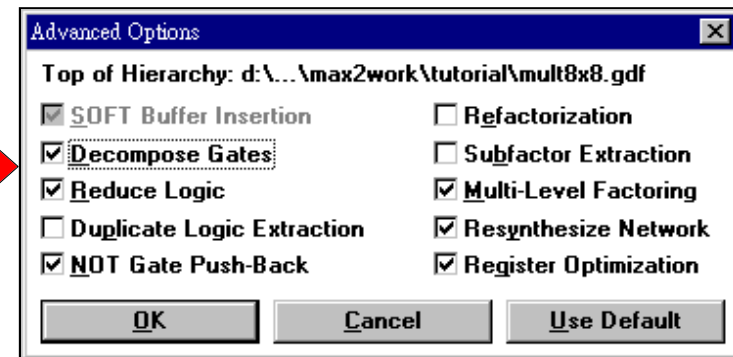
Carry Chain: Auto

Cascade Chain: Auto

Max Auto Length: 32

Max Auto Length: 2

OK Cancel Use Default Advanced Options...



Advanced Options

Top of Hierarchy: d:\...\max2work\tutorial\mult8x8.gdf

☒ SOFT Buffer Insertion

☒ Decompose Gates

☒ Reduce Logic

☐ Duplicate Logic Extraction

☒ NOT Gate Push-Back

☐ Refactorization

☐ Subfactor Extraction

☒ Multi-Level Factoring

☒ Resynthesize Network

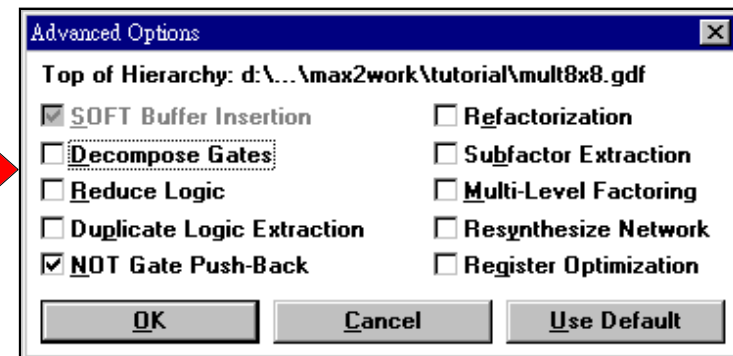
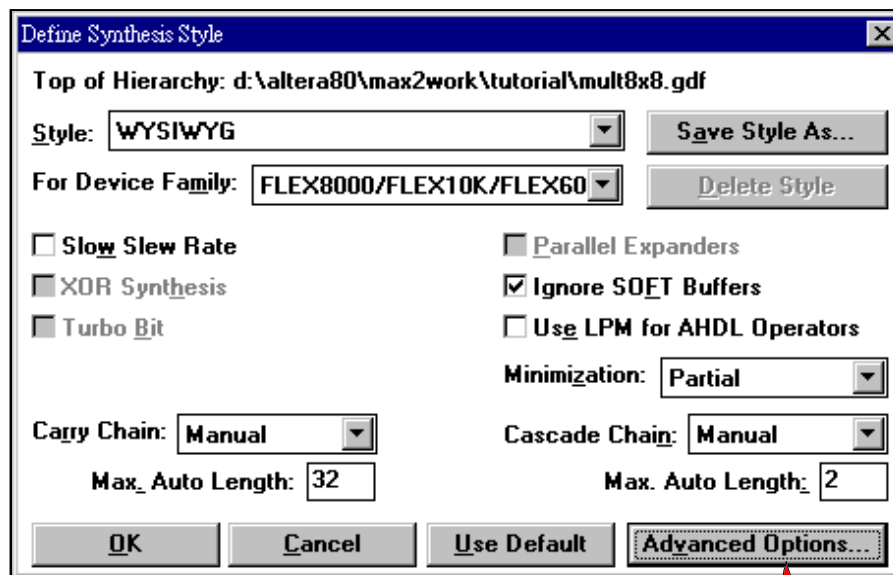
☒ Register Optimization

OK Cancel Use Default

Pre-Defined Synthesis Style - WYSIWYG

◆ “WYSIWYG” synthesis style

- “What You See Is What You Get” style -- do not remove or insert logic

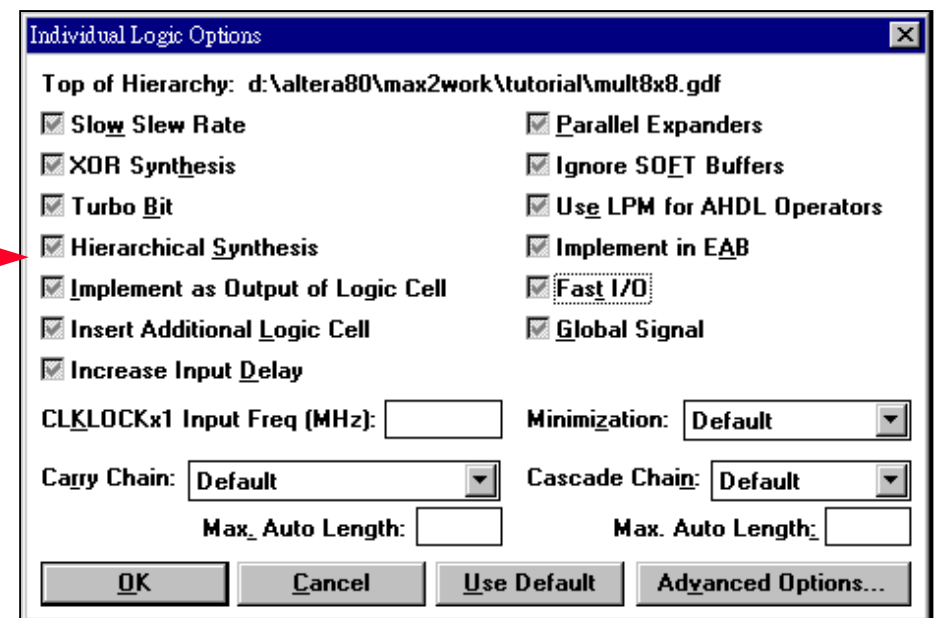
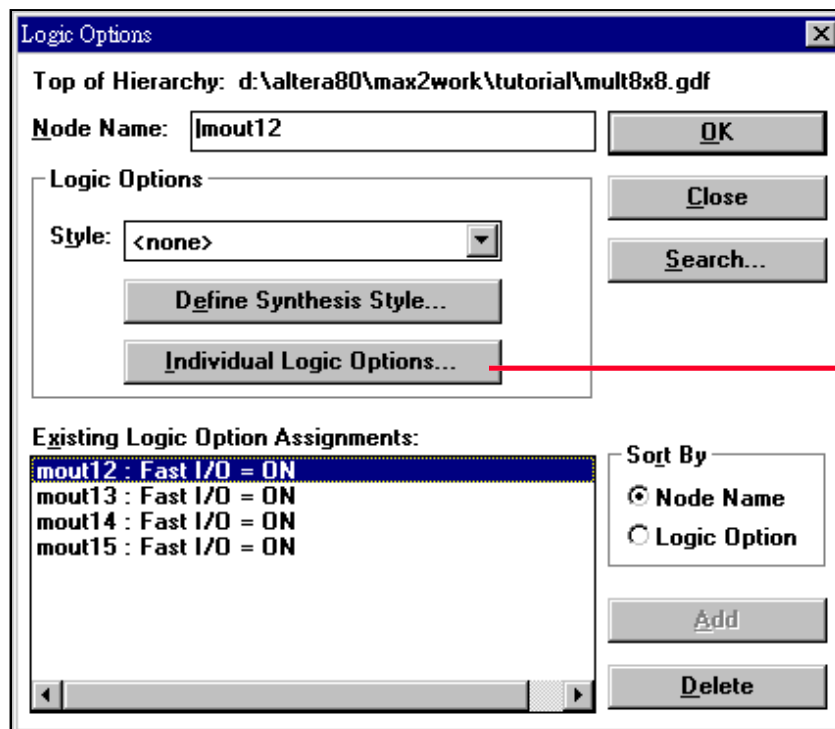


Individual Logic Options

◆ To specify individual logic options

- Assigns a logic synthesis style and/or logic options that control how the Compiler synthesizes logic for one or more nodes, buses, and/or symbols in the current hierarchy tree

Menu: *Assign -> Logic Options...*



Compiling the Project

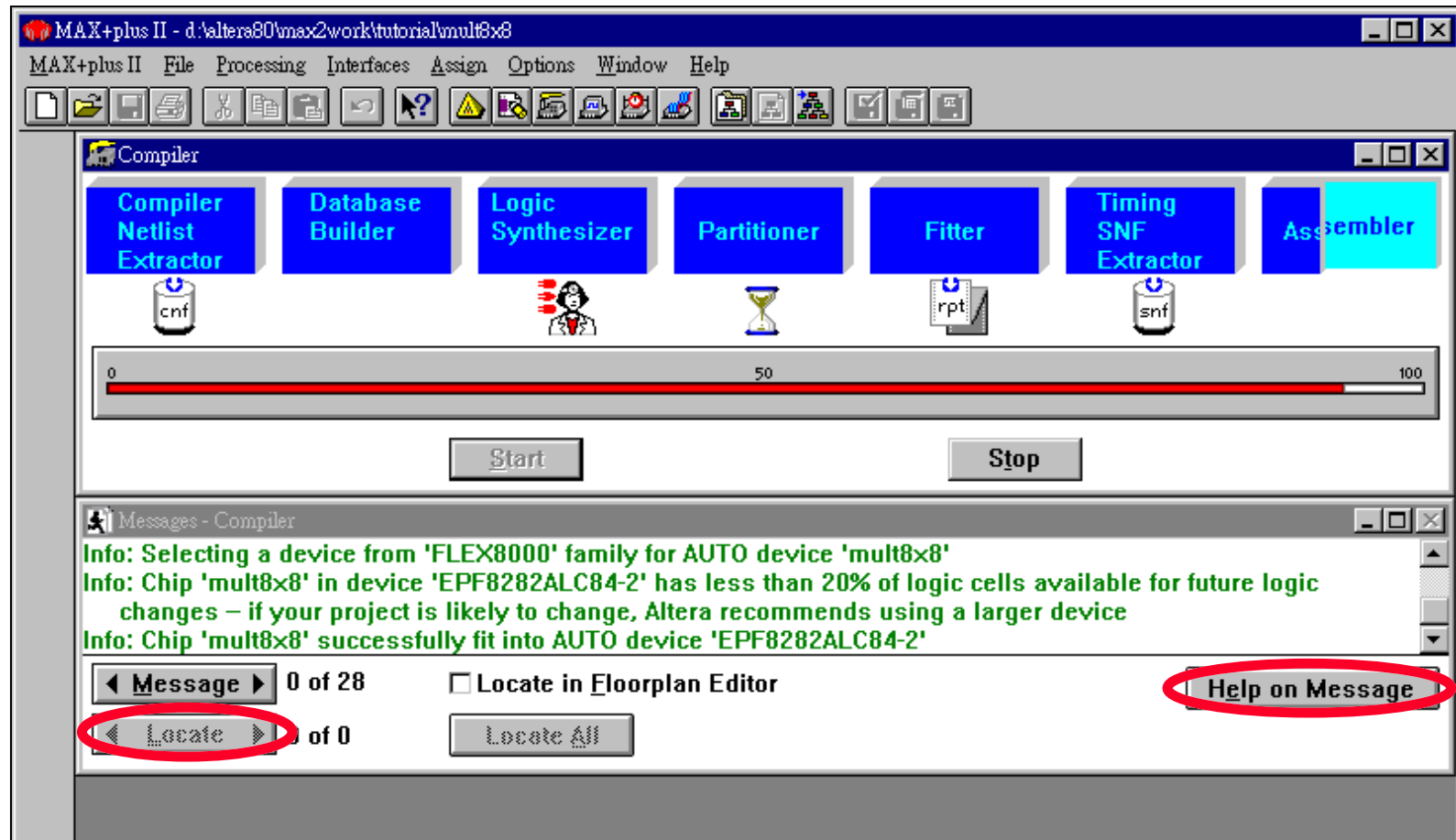
◆ Start the compilation

- Automatic compilation: just click on start button
- The Message Processor appears, see every warnings and errors if occurred

◆ Message Processor

- When the compiler begins running, the Message Processor appears to show error, warning, and information messages on the status of your project
 - Info
 - Warning
 - Error
- Get help on a message : select a message and click Help on Message

Compiling...



Checking the Messages

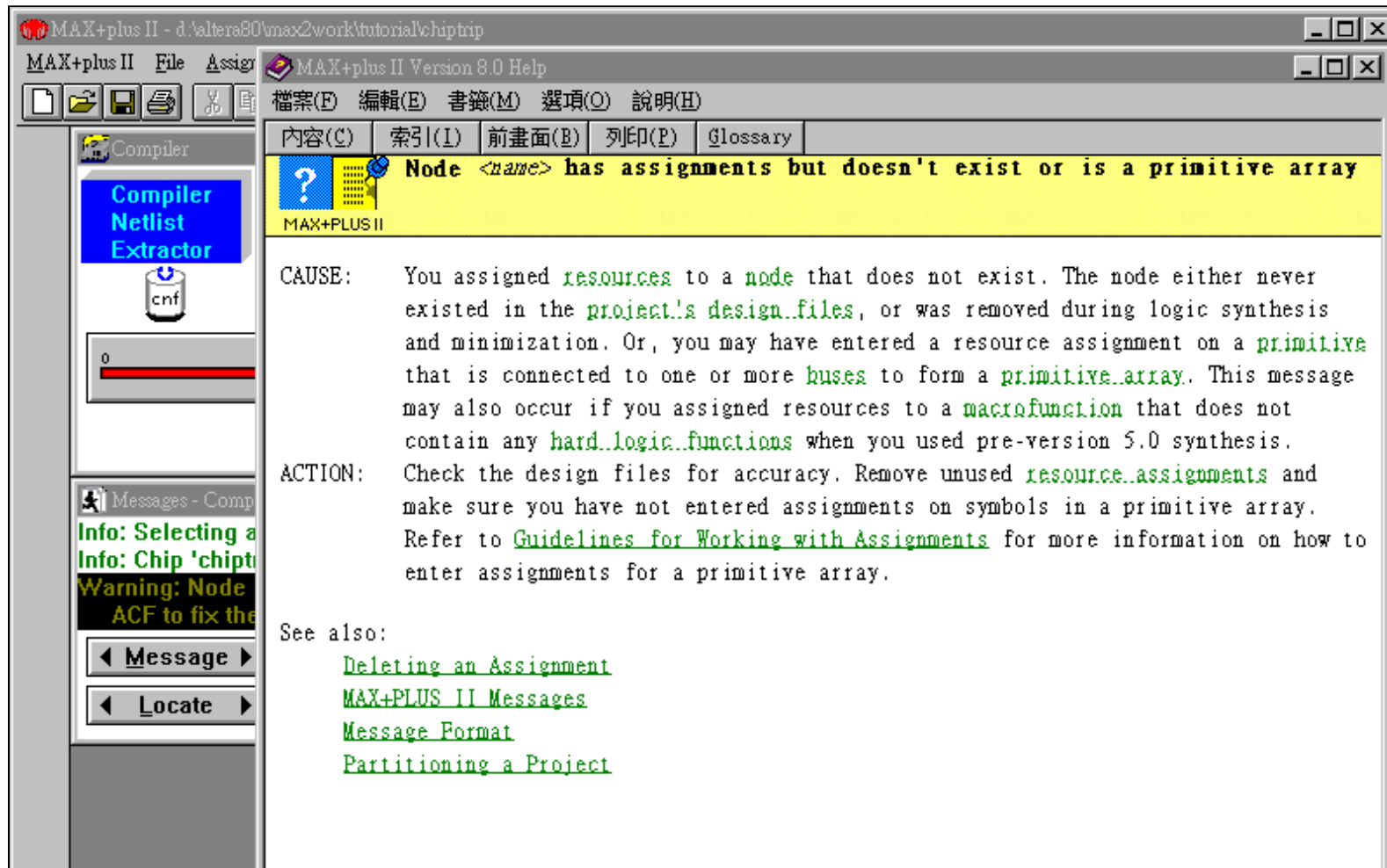
◆ Check the messages in Message Processor

- 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

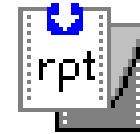
Help on Message



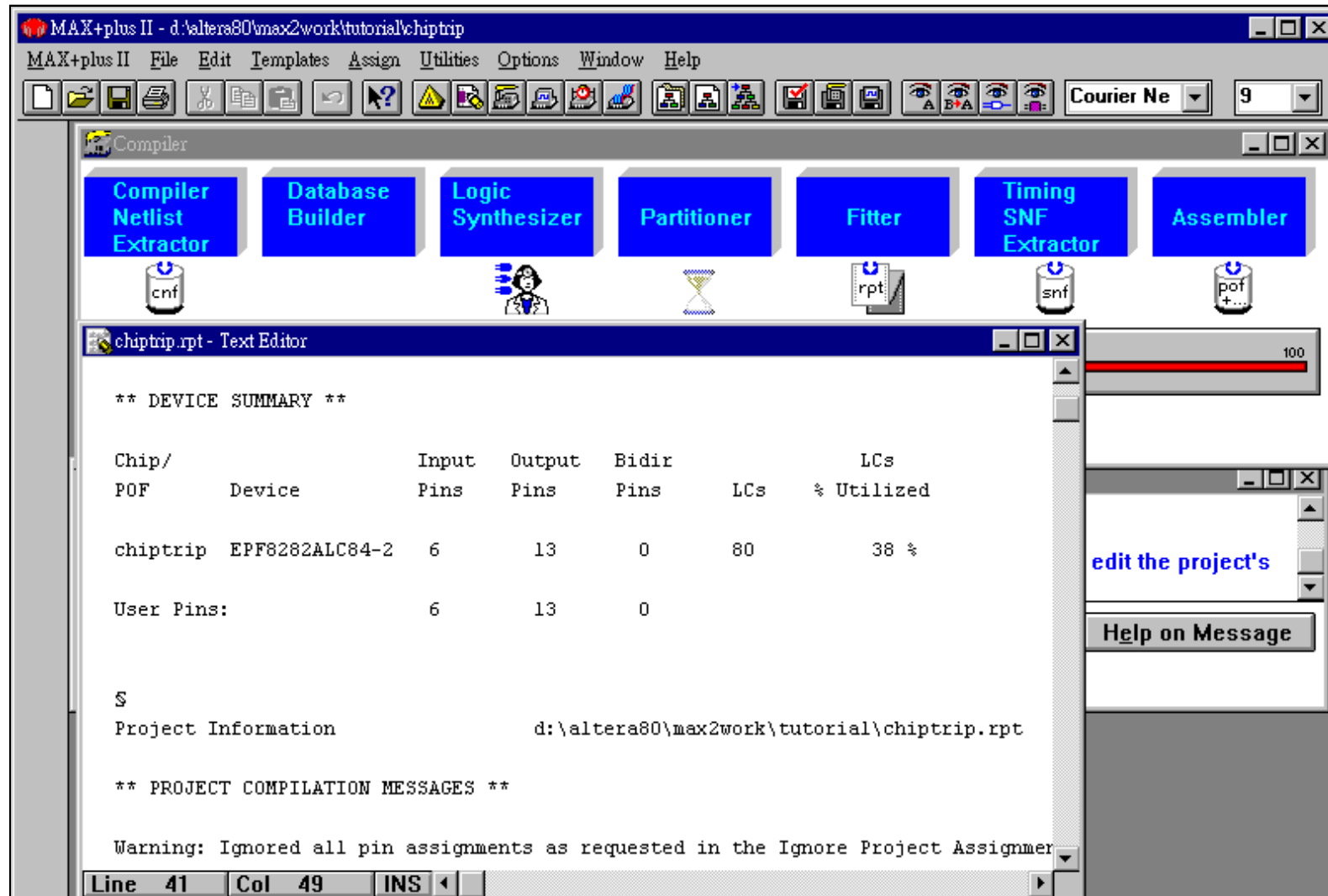
Checking the Reports

◆ 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, ...



Viewing Report File



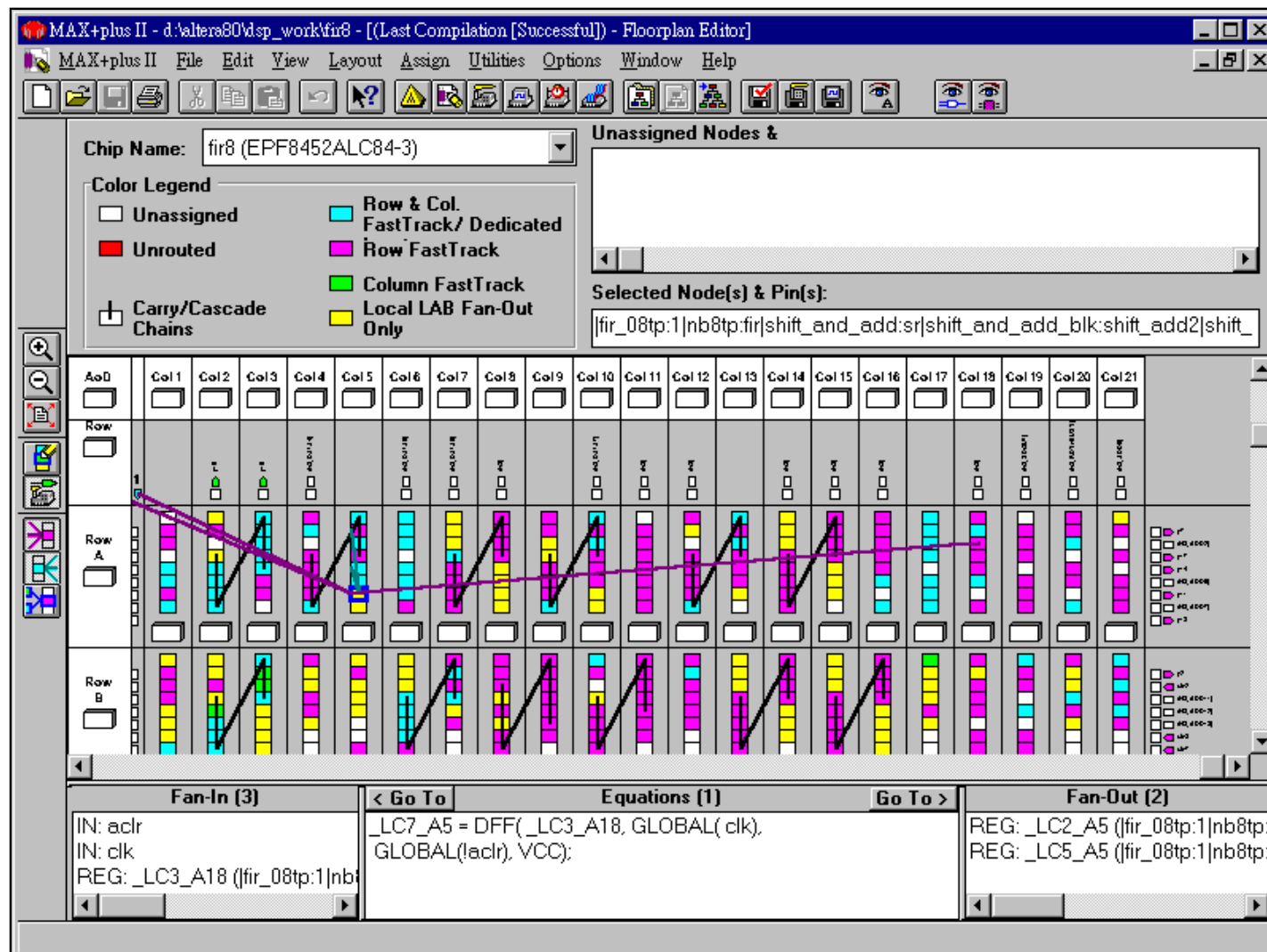
Checking the Floorplan

◆ Invoke Floorplan Editor to view the compilation results

- You can use the MAX+PLUS II Floorplan Editor to assign physical device resources or to view Compiler partitioning and fitting results
 - You can easily 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
 - You can use the Floorplan Editor to view the Compiler's logic placement and back-annotate the results of compilation if necessary
- To invoke Floorplan Editor

Menu: MAX+PLUS II -> Floorplan Editor

Floorplan Editor Window



Floorplan Viewers

◆ LAB View & Device View

- The Device View shows all pins on a device package and their function
- The LAB View shows the interior of the device, including all LABs and the individual logic cells within each LAB. In FLEX and MAX 9000 devices, I/O cell locations are also displayed.
- A color legend clearly indicates unassigned and assigned pins, logic cells, and I/O cells; unrouted items; and nonassignable items such as VCC, GND, and reserved pins
 - To toggle between Device View and LAB View

Menu: Layout -> Device View

Menu: Layout -> LAB View

◆ Report File Equation Viewer

- To display report file(*.rpt) routing and equation information for pin, logic cell, and embedded cell assignments
 - Toggle the Report File Equation Viewer

Menu: Layout -> Report File Equation Viewer

Floorplan Editor Options

◆ To show node fan-in, fan-out and routing information

- You can display fan-in and/or fan-out routing information for one or more selected embedded cells, logic cells, I/O cells, and/or pins in the LAB View display

- To show node fan-in & fan-out

Menu: *Options -> Show Node Fan-In*

Menu: *Options -> Show Node Fan-Out*

- To show node routing information, select a node...

Menu: *Options -> Routing Statistics...*

◆ To show paths between nodes

- You can display the path between selected logic cells, I/O cells, embedded cells, and/or pins that feed one another

- To show node paths

Menu: *Options -> Show Path*

LAB View

Chip Name:

Color Legend

- Unassigned
- Unrouted
- Moved
- Carry/Cascade Chains
- Row & Col. FastTrack/ Dedicated
- Row FastTrack
- Column FastTrack
- Local LAB Fan-Out Only

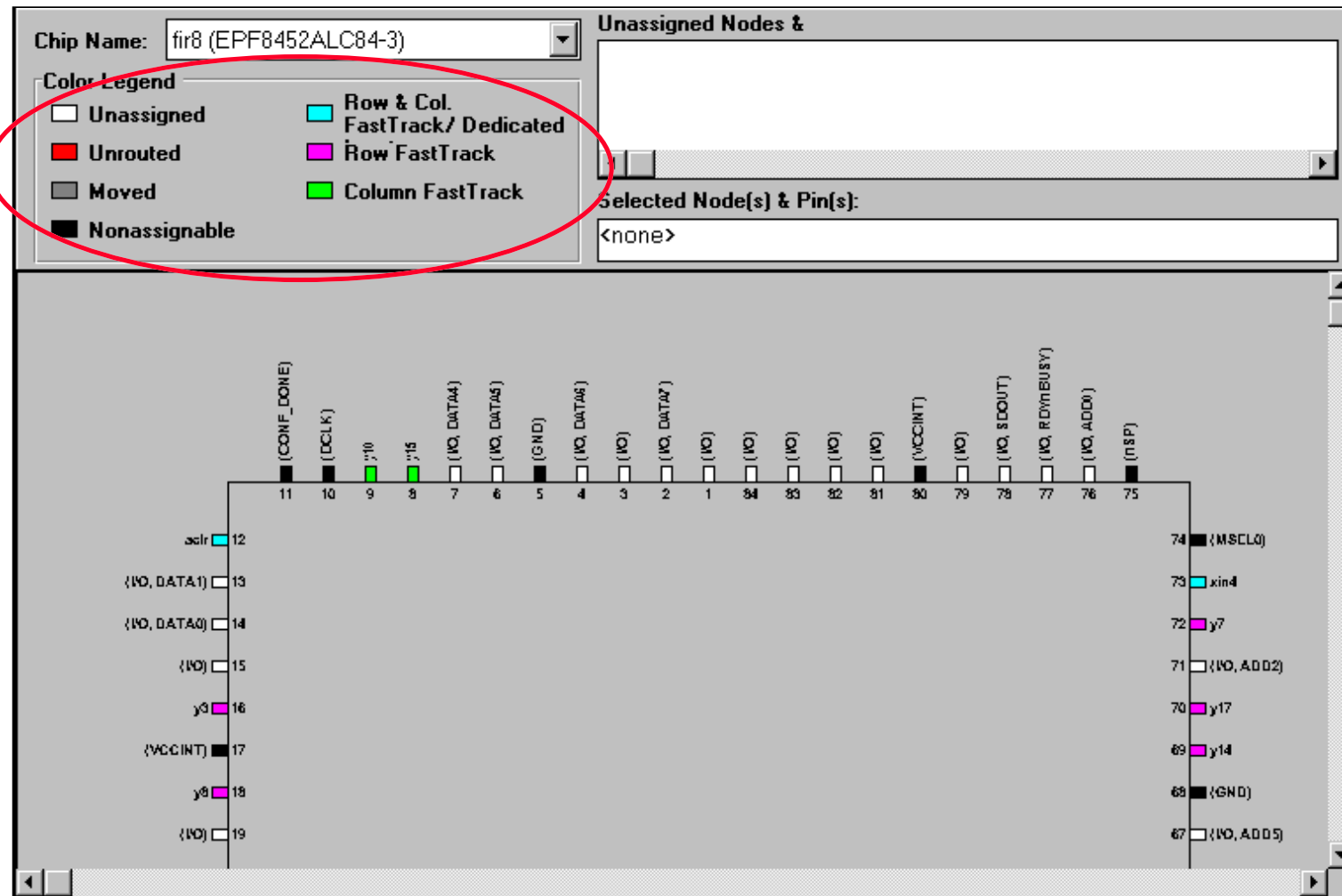
Unassigned Nodes & Pin(s):

Selected Node(s) & Pin(s):

<none>

AoD	Any Col	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13
Any Row	stim1 clk stim4 sclr		x10 y15		(IO, DATA4) y15		(IO, DATA5) y15	(IO, DATA6) y15	(IO) y15		(IO, DATA7) y15	(IO) y15	(IO) y15	
Row A	(IO, DATA1) (IO, DATA0) (IO) y3 y6 (IO) (IO) y12													
Row	xin9 y6													

Device View



Report File Equation Viewer

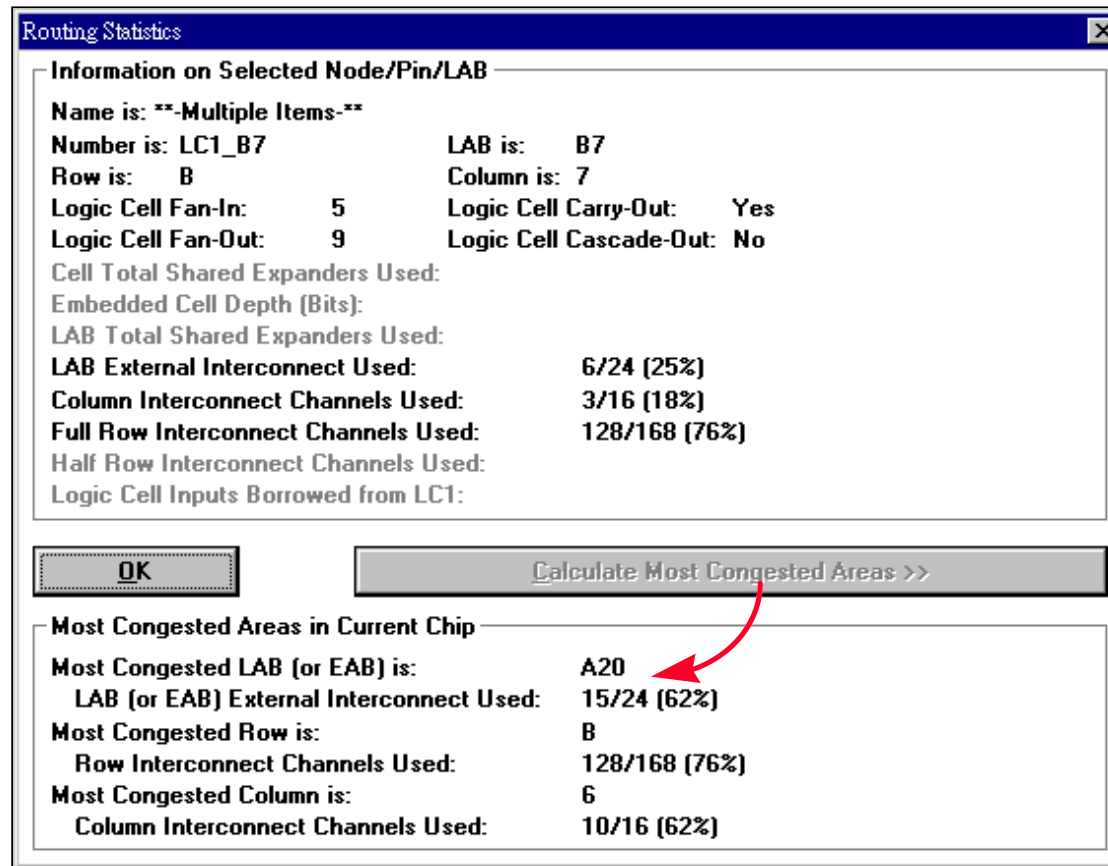
The screenshot shows the Report File Equation Viewer interface. The main window displays a grid of equations for various rows and columns. A red arrow points to the 'Go To' button in the bottom panel, which is used to navigate to a specific equation in the grid.

AoD	Any Col	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13
Any Row														
Row A	<input type="checkbox"/> {IO, DATA1} <input type="checkbox"/> {IO, DATA0} <input type="checkbox"/> {IO} <input type="checkbox"/> y0 <input type="checkbox"/> y8 <input type="checkbox"/> {IO} <input type="checkbox"/> {IO} <input type="checkbox"/> y12													
Row B	<input type="checkbox"/> xin8 <input type="checkbox"/> y6 <input type="checkbox"/> y9 <input type="checkbox"/> {IO} <input type="checkbox"/> {IO}													

Bottom Panel:

- Fan-In (3)**: IN: aclr, IN: clk, REG: _LC2_A18 (fif_08tp:1|nb8tp:1)
- Equations (1)**: `_LC1_B6 = DFF(_LC2_A18, GLOBAL(clk), GLOBAL(laclr), VCC);`
- Go To >**: REG: _LC8_B6 (fif_08tp:1|nb8tp:1)

Routing Statistics



The image shows a 'Routing Statistics' dialog box. It has a title bar with a close button. The main area is divided into two sections. The top section, 'Information on Selected Node/Pin/LAB', lists details for a selected node: Name is '**-Multiple Items-**', Number is 'LC1_B7', LAB is 'B7', Row is 'B', Column is '7', Logic Cell Fan-In is '5', Logic Cell Carry-Out is 'Yes', Logic Cell Fan-Out is '9', and Logic Cell Cascade-Out is 'No'. It also shows resource usage: 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, and Logic Cell Inputs Borrowed from LC1. The bottom section, 'Most Congested Areas in Current Chip', lists: 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', and Column Interconnect Channels Used (10/16, 62%). A red arrow points from the 'Calculate Most Congested Areas >>' button to the 'Most Congested LAB (or EAB) is: A20' text. At the bottom are 'OK' and 'Calculate Most Congested Areas >>' buttons.

Routing Statistics

Information on Selected Node/Pin/LAB

Name is: ****Multiple Items****
Number is: **LC1_B7** LAB is: **B7**
Row is: **B** Column is: **7**
Logic Cell Fan-In: **5** Logic Cell Carry-Out: **Yes**
Logic Cell Fan-Out: **9** Logic Cell Cascade-Out: **No**

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:

OK **Calculate Most Congested Areas >>**

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, ...

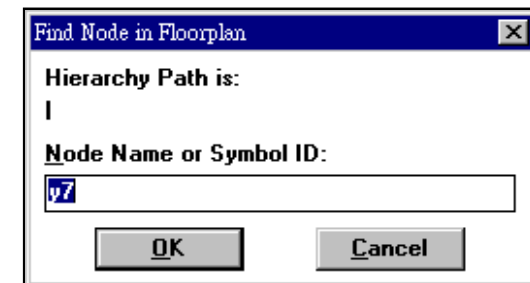
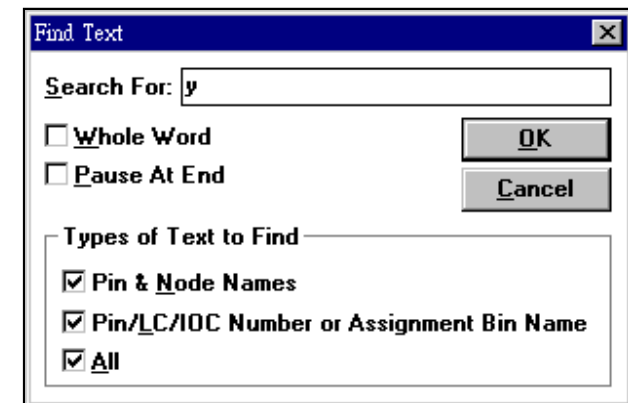
- “Find Text” command: to search the current chip for the first occurrence of the specified text
- “Find Node” command: to find one or more nodes or other logic function(s) in the design file or in the floorplan

◆ To help running timing analysis

- You can specify source and destination nodes in the floorplan to run timing analysis

Find <u>T</u> ext...	Ctrl+F
Find <u>N</u> ode in Design File...	Ctrl+B
Find <u>N</u> ode in Floorplan...	
Find <u>N</u> ext	Ctrl+N
Find <u>P</u> revious	Ctrl+Shift+N
Find <u>L</u> ast Edit	
<hr/>	
Timing Analysis <u>S</u> ource	Ctrl+Alt+S
Timing Analysis <u>D</u> estination	Ctrl+Alt+D
Timing Analysis <u>C</u> utoff	Ctrl+Alt+C
<u>A</u> nalyze Timing	
<u>C</u> lear All Timing Analysis Tags	

Floorplan Editor Utilities Menu



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

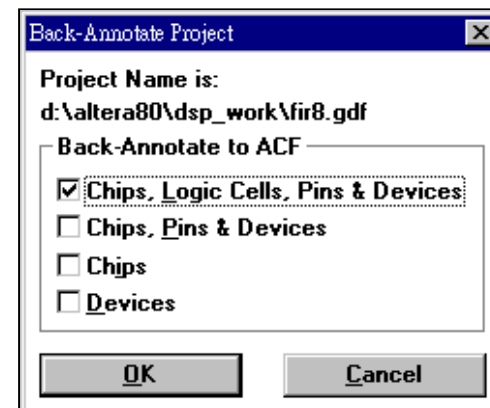
Menu: *Layout -> Current Assignments Floorplan*

Menu: *Layout -> Last Compilation Floorplan*

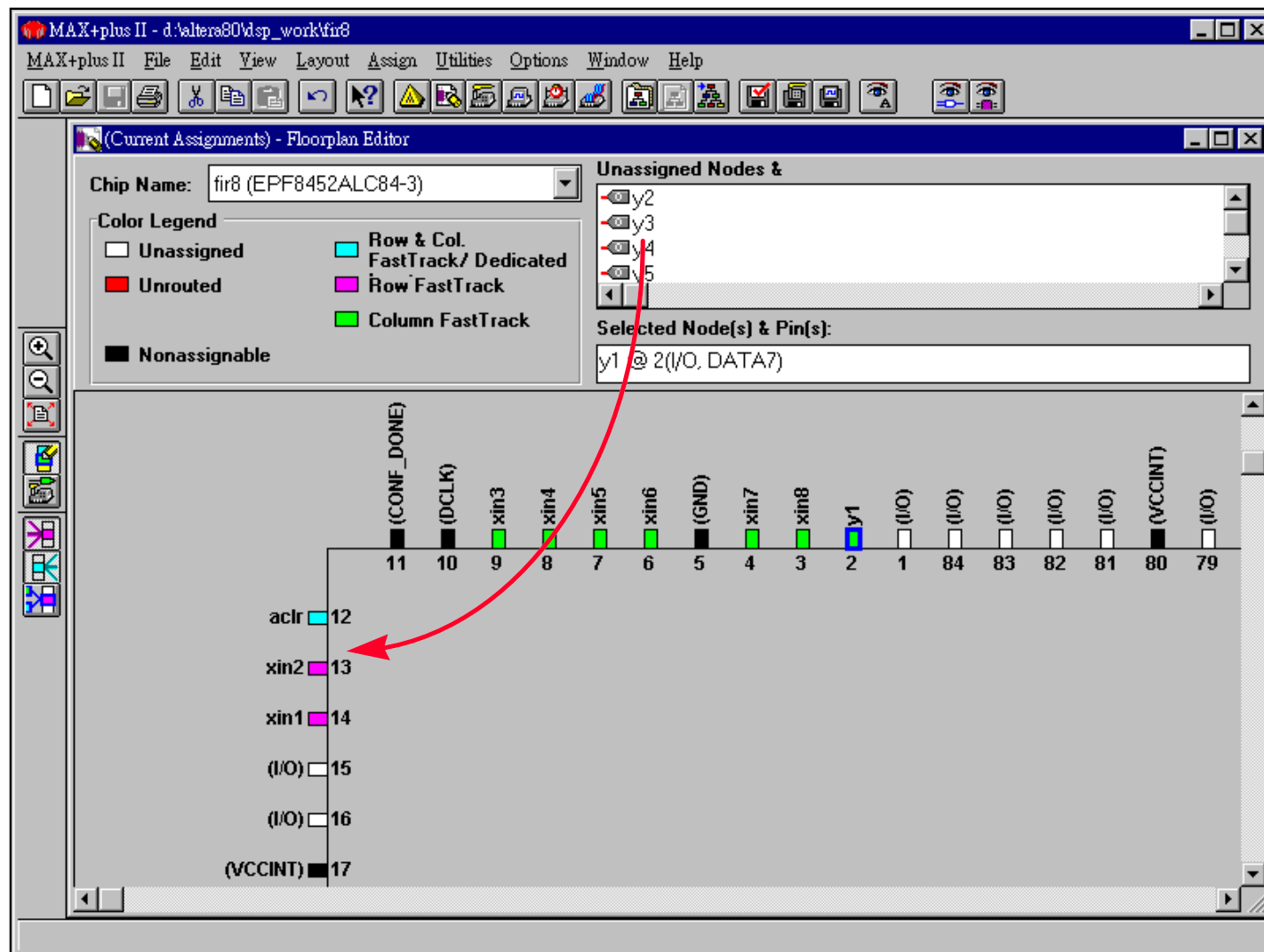
◆ Back-annotate the floorplan for subsequent compilation

- 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-Annotate Project...*



Current Pin Assignment Floorplan



Current LAB Assignment Floorplan

MAX+plus II - d:\altera80\dsp_work\fir8

MAX+plus II File Edit View Layout Assign Utilities Options Window Help

(Current Assignments) - Floorplan Editor

Chip Name: fir8 (EPF8452ALC84-3)

Color Legend

- Unassigned
- Unrouted
- Carry/Cascade Chains
- Row & Col. FastTrack/ Dedicated
- Row FastTrack
- Column FastTrack
- Local LAB Fan-Out Only

Unassigned Nodes &

- [fir_08tp:1|nb8tp.fir|shift_and_add:sr|shift_and_add_blk:shift_add
- [fir_08tp:1|nb8tp.fir|shift_and_add:sr|shift_and_add_blk:shift_add
- [fir_08tp:1|nb8tp.fir|shift_and_add:sr|shift_and_add_blk:shift_add
- [fir_08tp:1|nb8tp.fir|shift_and_add:sr|shift_and_add_blk:shift_add

Selected Node(s) & Pin(s):

[fir_08tp:1|nb8tp.fir|shift_and_add:sr|shift_and_add_blk:shift_add3|pip

Anywhere on this Column

Anywhere on Device

Anywhere on this Row

AoD

Any Col

Col 1

Col 2

Col 3

Col 4

Col 5

Col 6

Col 7

Col 8

Col 9

Any Row

(Ded. Input)

(Ded. Input)

clk

acir

xin2

xin1

(I/O)

(I/O)

(I/O)

(I/O)

Row A

8

xin3

xin4

xin5

xin6

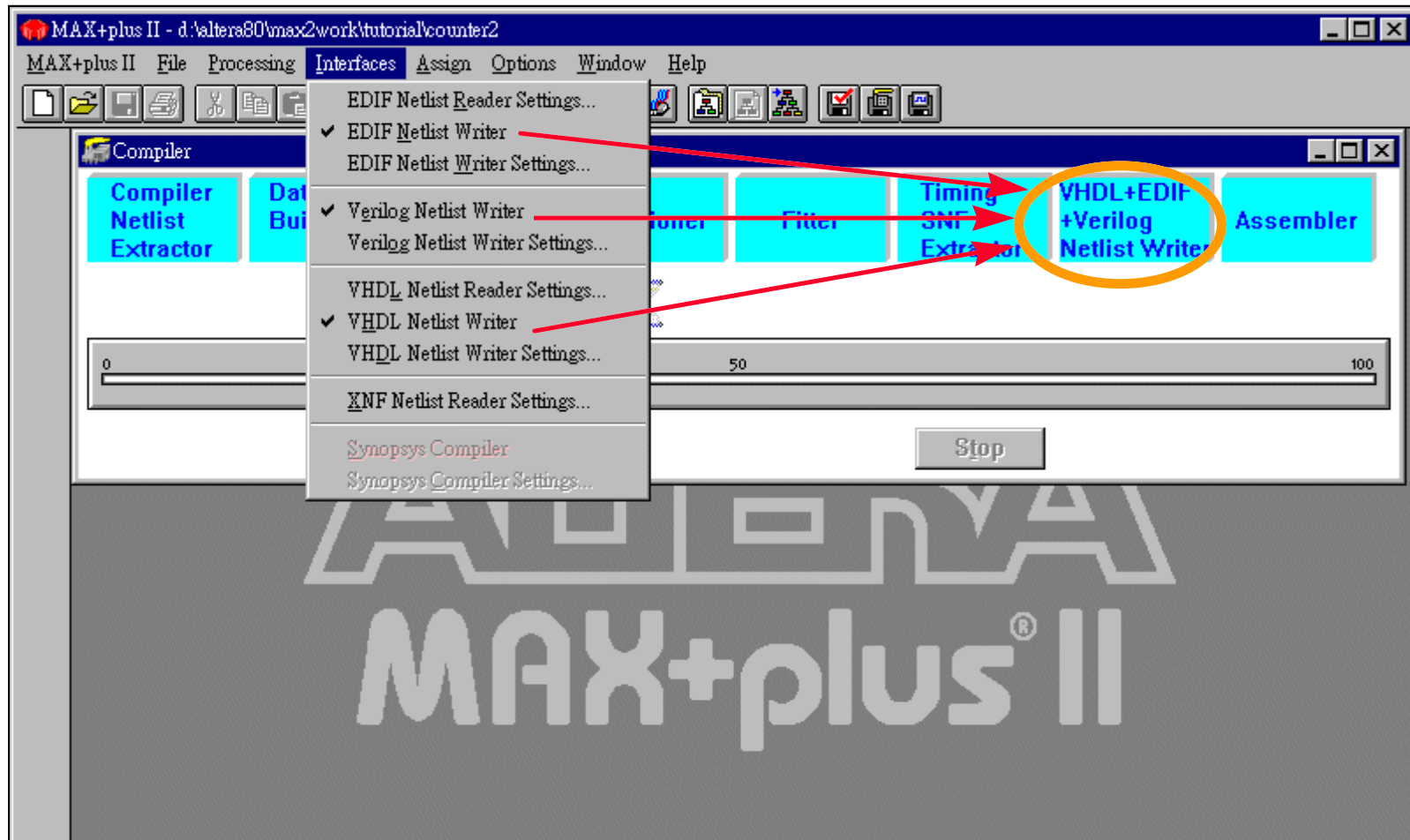
xin7

xin8

Appendix: Interfacing with 3rd-Party Tools

- ◆ To interface with other 3rd-party CAE tools, MAX+PLUS II reads the EDIF file created by 3rd-party CAE tools and writes EDIF, VHDL or Verilog file with timing after project compilation
 - You must explicitly specify how MAX+PLUS II interfaces with 3rd-party tools by using Interface Menu commands under MAX+PLUS II Compiler

Interface Menu



Interface Settings - (1)

◆ EDIF netlist reader settings (only used to interface with third-party tools)

- You need to specify the vendor that generated the project EDIF file
 - Altera uses a Library Mapping File (*.lmf) to map cells in EDIF input files to corresponding MAX+PLUS II primitives, megafunctions and macrofunctions

Menu: *Interfaces -> EDIF Netlist Reader Settings...*

EDIF Netlist Reader Settings

Vendor: **Synopsys**

☐ Show LMF Mapping Messages

OK Cancel

Signal: **VCC: VDD** **GND: GND**

Library Mapping Files

☒ LMF #1: **d:\altera80\maxplus2\lmf\altsyn.lmf**

☐ LMF #2: ***.lmf**

Directory is: **d:\altera80\max2work\tutorial**

Files: ***.lmf**

Directories:

- max2work
- tutorial**
- fusion
- user.lib

Drives:

Interface Settings - (2)

◆ EDIF netlist writer settings

- “EDIF Netlist Writer” option can be turned on to export the post-route netlist to an industry-standard workstation or PC environment for simulation or resynthesis
- You can direct the Compiler to include the delay constructs in the EDIF output file (*.edo) or write the delay constructs to a separate SDF output file (*.sdo)

Menu: *Interfaces -> EDIF Netlist Writer*

Menu: *Interfaces -> EDIF Netlist Writer Settings...*

The screenshot shows the "EDIF Netlist Writer Settings" dialog box. It has a title bar with a close button. The "Vendor" is set to "Synopsys". The "EDIF Version" has two radio buttons: "EDIF 2 0 0" (selected) and "EDIF 3 0 0". There are "Customize >>", "OK", and "Cancel" buttons. The "Signal Names" section has "VCC:" and "GND:" labels with text boxes containing "VCC" and "GND" respectively. The "Write Delay Constructs To" section has three radio buttons: "EDIF Output File [.edo]" (selected), "SDF Output File [.sdo] Ver 2.1", and "SDF Output File [.sdo] Ver 1.0". To the right of this are checkboxes for "Map Illegal EDIF Characters", "Force 0 ns Delays", "Time Scale:" (set to "0.1ns"), "Flatten Bus (EDIF 2 0 0 only)", and "Bus Delimiters:" (set to "()"). Below these are checkboxes for "Include Special Primitives" (checked), "Truncate Long Hierarchy Paths", and "EDIF Command File:" (set to "*.edc"). The "Directory is:" field shows "d:\altera80\max2work\tutorial". The "Files:" field shows "*.edc". The "Directories:" list shows "max2work", "tutorial" (selected), "fusion", and "user.lib". The "Drives:" field is empty.

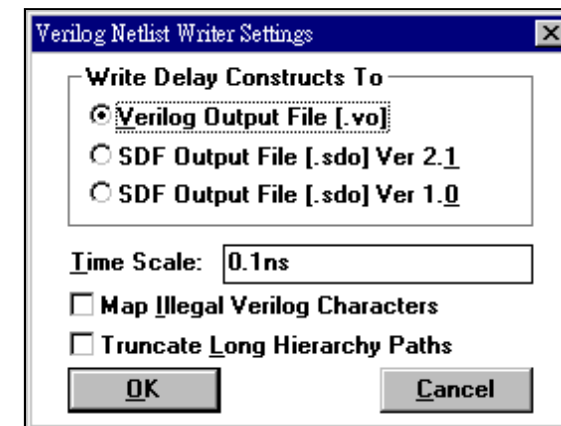
Interface Settings - (3)

◆ Verilog netlist writer settings

- “Verilog Netlist Writer” option must be turned on for the future Verilog timing simulation
- You can direct the Compiler to include the delay constructs in the EDIF output file (*.edo) or write the delay constructs to a separate SDF output file (*.sdo)

Menu: *Interfaces -> Verilog Netlist Writer*

Menu: *Interfaces -> Verilog Netlist Writer Settings...*



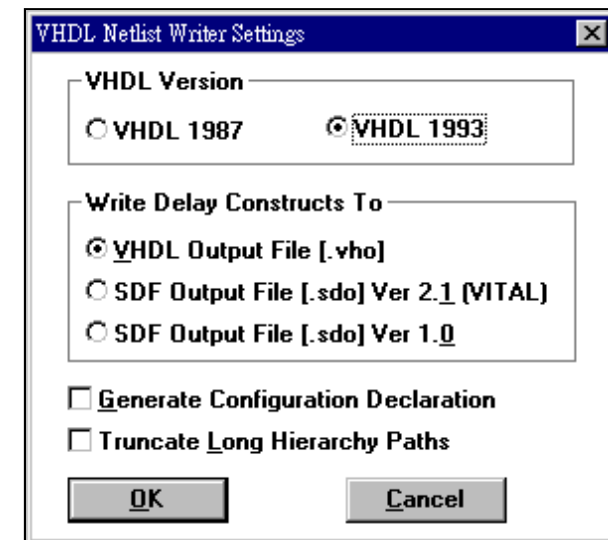
Interface Settings - (4)

◆ VHDL netlist writer settings

- “VHDL Netlist Writer” option must be turned on for the future VHDL timing simulation
- You can direct the Compiler to include the delay constructs in the EDIF output file (*.edo) or write the delay constructs to a separate SDF output file (*.sdo)

Menu: *Interfaces -> VHDL Netlist Writer*

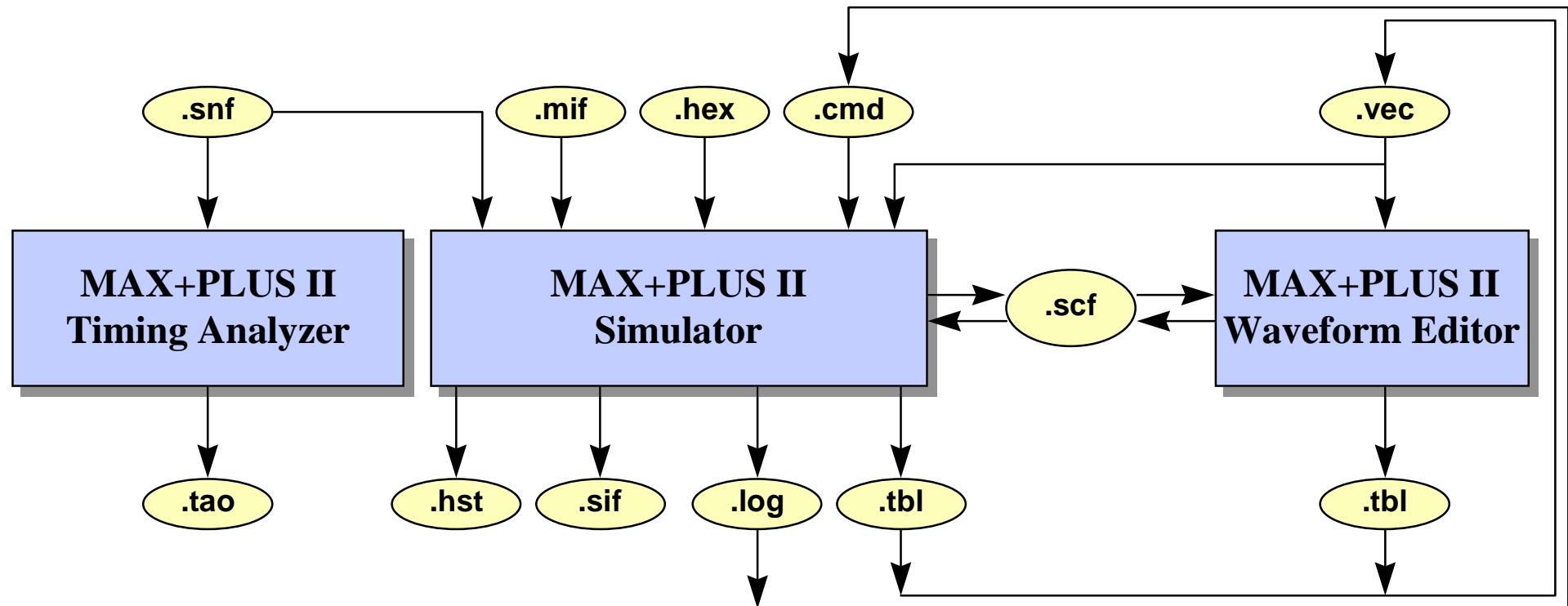
Menu: *Interfaces -> VHDL Netlist Writer Settings...*



Project Verification

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

Project Verification Methodology

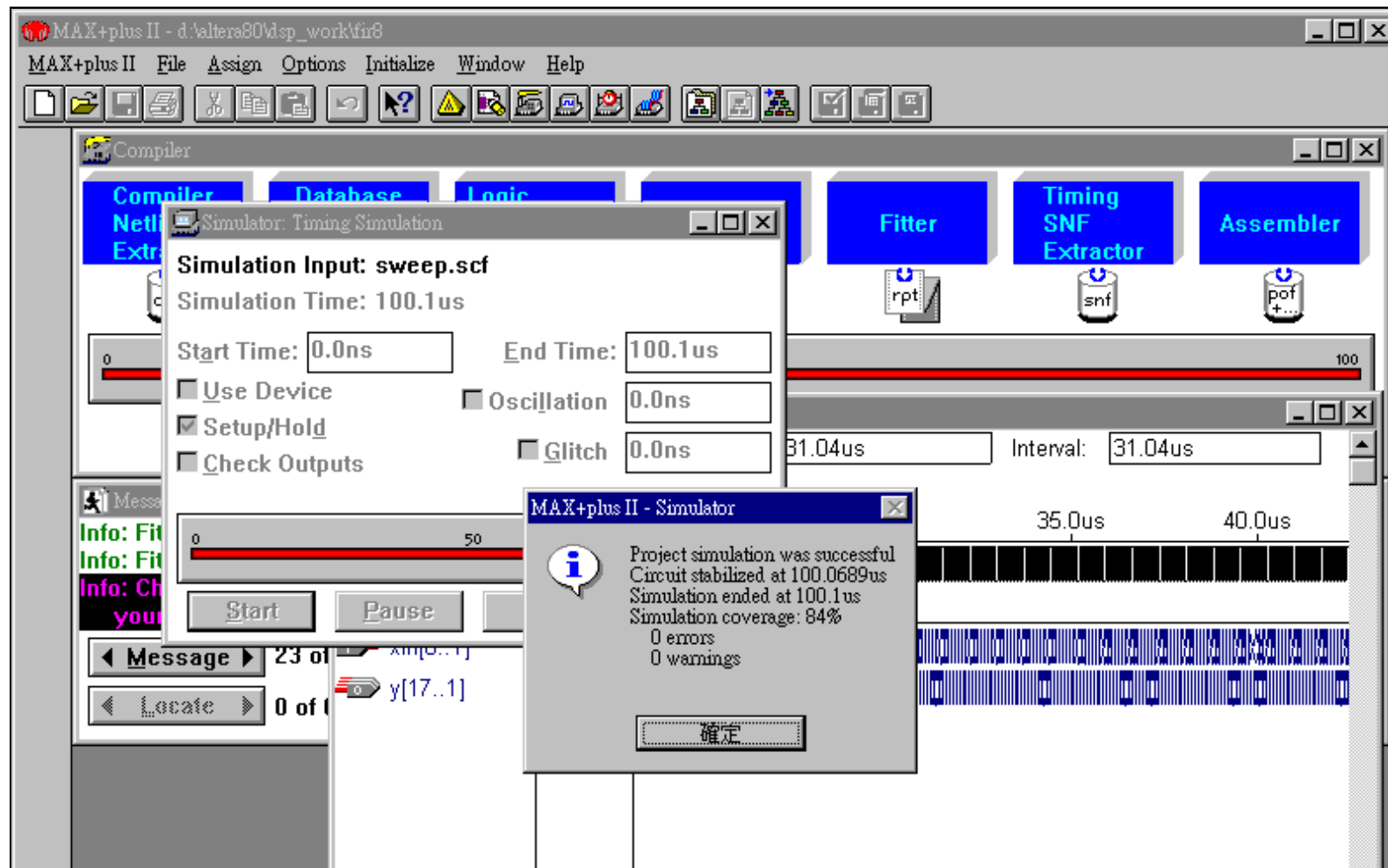


MAX+PLUS II Simulator

◆ Test the logical operation & internal timing of a project

- The Simulator allows you to simulate a project in interactive mode with menu commands and on-screen options and buttons, or in batch mode with a command file(*.cmd)
- Use the SNF file generated by MAX+PLUS II Compiler
 - Functional SNF file: for functional simulation
 - Timing SNF file: for timing analysis & simulation
 - Linked SNF file: for multi-project simulation
- The Simulator uses an SCF(waveform) or VEC(ASCII) file as the source of simulation input vectors
- The HEX or MIF(Memorization Initialization File) is used to specify the initial content of a memory block
- You can monitor you project for glitches, oscillations, and setup & hold time violations
- The Simulator allows you to check the outputs of simulation against any outputs in the SCF

Simulator Environment



Functional Simulation

◆ To verify functionality before project compilation

- Must generate functional SNF file of the project

◆ Perform functional simulation

- Create SCF with Waveform Editor first
 - The SCF should be considered for the future re-use in timing simulation

- Generate Functional SNF file

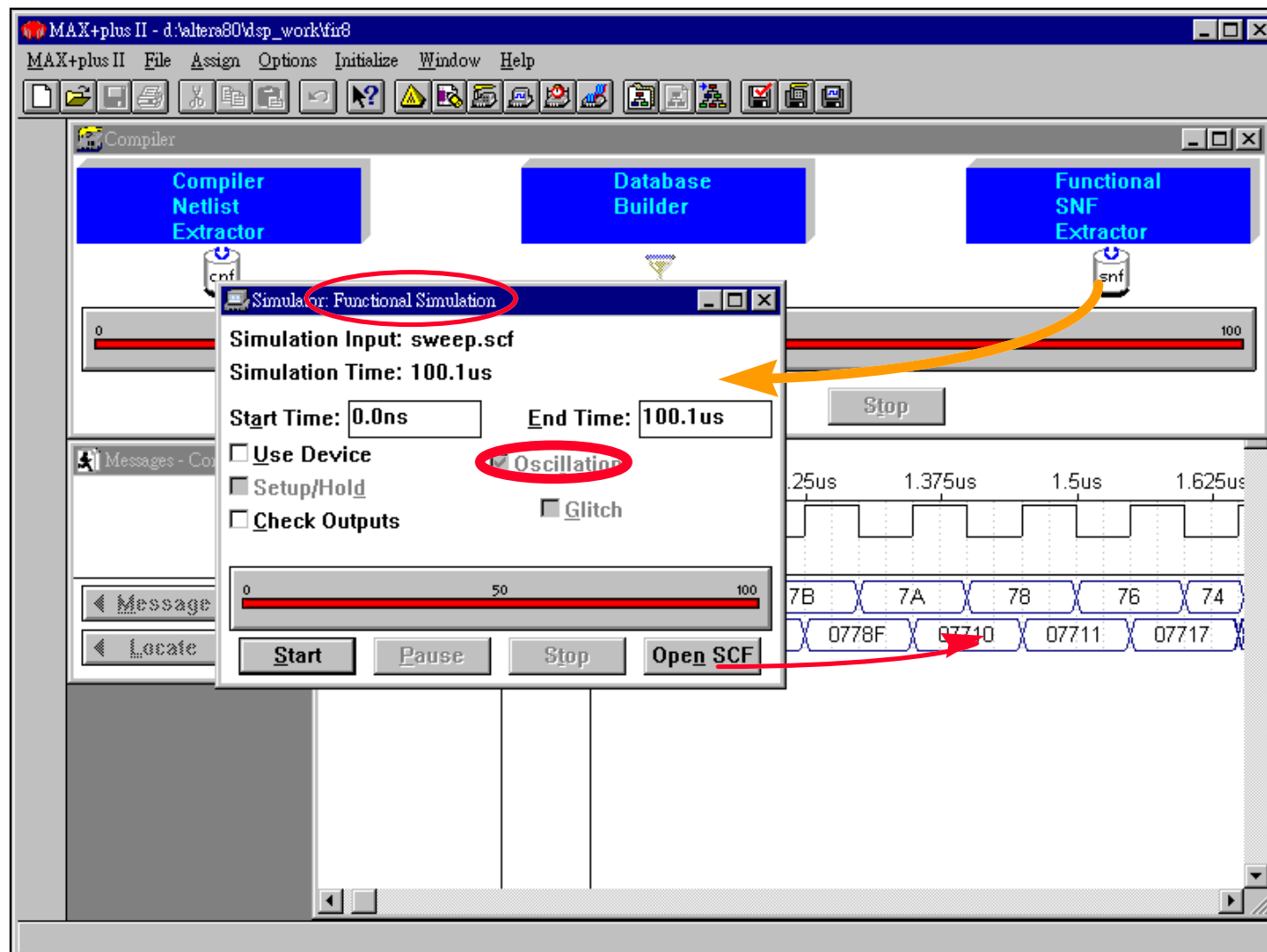
Menu: MAX+PLUS II -> Compiler

Menu: Processing -> Functional SNF Extractor (then click Start)

- Invoke MAX+PLUS II Simulator to do functional simulation

Menu: MAX+PLUS II -> Simulator (then click Start & Open SCF)

Running Functional Simulation



Timing Simulation

◆ To verify real-world functionality & timing of your project

- Must generate timing SNF during project compilation

◆ Perform timing simulation

- Generate Timing SNF file

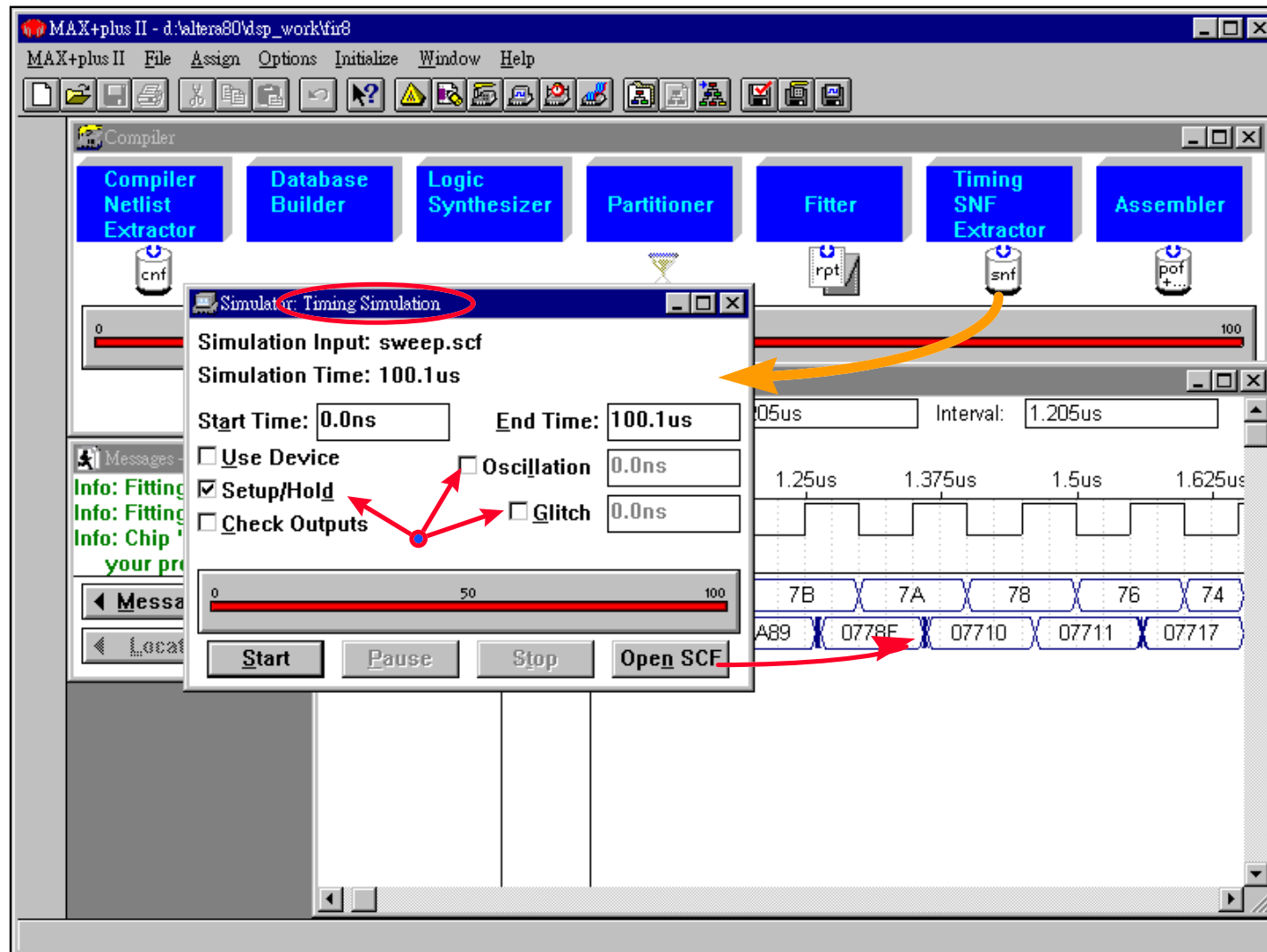
Menu: *MAX+PLUS II -> Compiler*

Menu: *Processing -> Timing SNF Extractor (then click Start)*

- Invoke MAX+PLUS II Simulator to do timing simulation

Menu: *MAX+PLUS II -> Simulator (then click Start & Open SCF)*

Running Timing Simulation



Simulation Input & Output Files

◆ Specify simulation input and output files

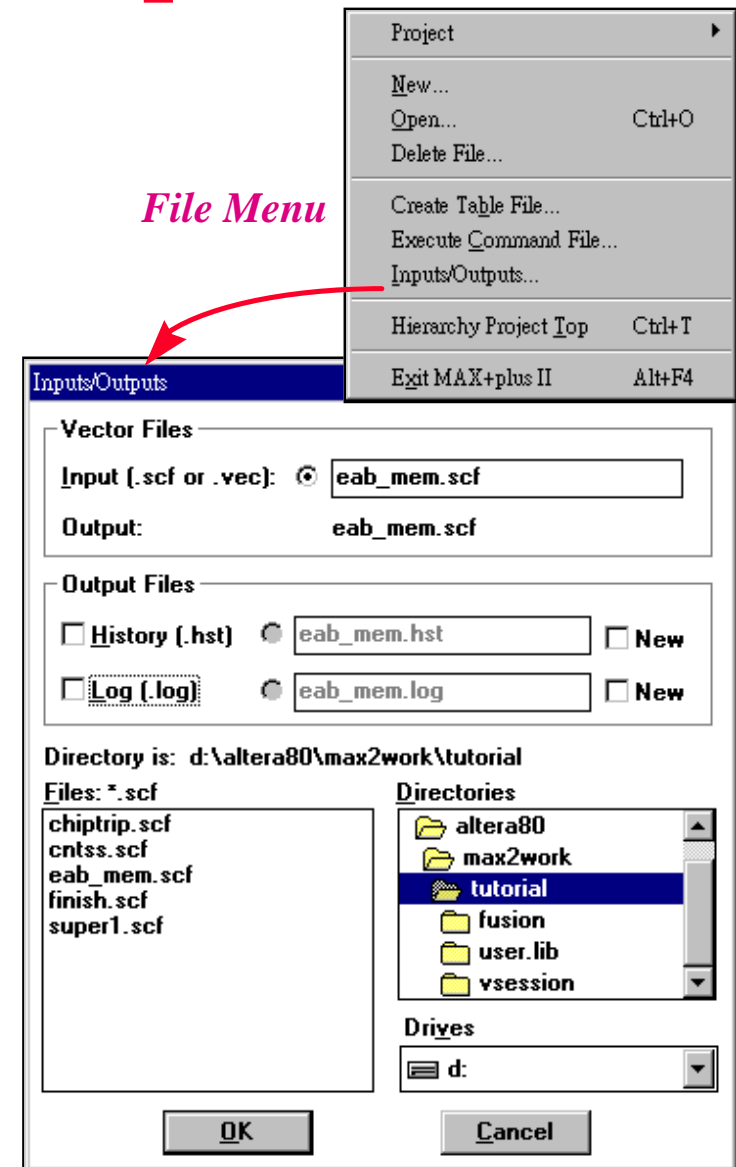
- You can specify SCF or VEC file as the source of simulation input vectors

Menu: *File -> Inputs/Outputs...*

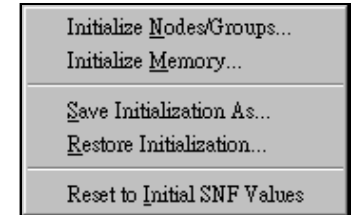
- VEC file will be converted into SCF file by Simulator
- You can specify a history(*.hst) or log(*.log) file to record simulation commands and outputs
- During and after simulation, the simulation results are written to the SCF file, you can create another ASCII-format table file

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



Initialize Menu

◆ 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

Memory: ILPM_RAM_DQ:1|altram:sram|content

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

Addr Radix: ☐ BIN ☐ OCT ☐ DEC ☒ HEX

Value Radix: ☐ BIN ☐ OCT ☐ DEC ☒ HEX

☐ List One Address Per Line

Memory Info:
Depth: 256
Width (Bits): 16
Type: RAM

Initialize to 0's
Initialize to 1's
Import File...
Export File...

OK Cancel

Import Memory Content File

File Name: *.mif

Directory is: d:\altera80\max2work\tutorial

Files: mem1.mif

Directories: altera80, max2work, tutorial, fusion, user.lib, vsession

Drives: d:

Automatic Extension: .mif, .hex, .mif

OK Cancel

Export Memory Content File

File Name: mem1.mif

Directory is: d:\altera80\max2work\tutorial

Files: *.mif

Directories: altera80, max2work, tutorial, fusion, user.lib, vsession

Drives: d:

Automatic Extension: .mif, .hex, .mif

OK Cancel

Memory Initialization File Formats

Initialize Memory

Memory: ILPM_RAM_DQ:1|altram:sram|content

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

Addr Radix: ☐ BIN ☐ OCT ☐ DEC ☒ HEX

Value Radix: ☐ BIN ☐ OCT ☐ DEC ☒ HEX

☐ List One Address Per Line

Memory Info:
 Depth: 256
 Width (Bits): 16
 Type: RAM

Buttons: Initialize to 0's, Initialize to 1's, Import File..., Export File..., OK, Cancel

```
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 : 0000;
8 : ffff;
9 : ffff;
a : ffff;
b : ffff;
c : ffff;
d : ffff;
e : ffff;
f : ffff;
```

```
...
ff : 0000;
END;
```

MIF file example

```
:020000000000fe
:020001000000fd
:020002000000fc
:020003000000fb
:020004000000fa
:020005000000f9
:020006000000f8
:020007000000f7
:02000800fffff8
:02000900fffff7
:02000a00fffff6
:02000b00fffff5
:02000c00fffff4
:02000d00fffff3
:02000e00fffff2
:02000f00fffff1
...
:0200ff000000ff
:00000001ff
```

HEX file example

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 %
END;      % Addr[8]=F, Addr[9]=E, Addr[A]=5 %
```


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

Notes for Compiling & Simulating RAM / ROM - (2)

◆ 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

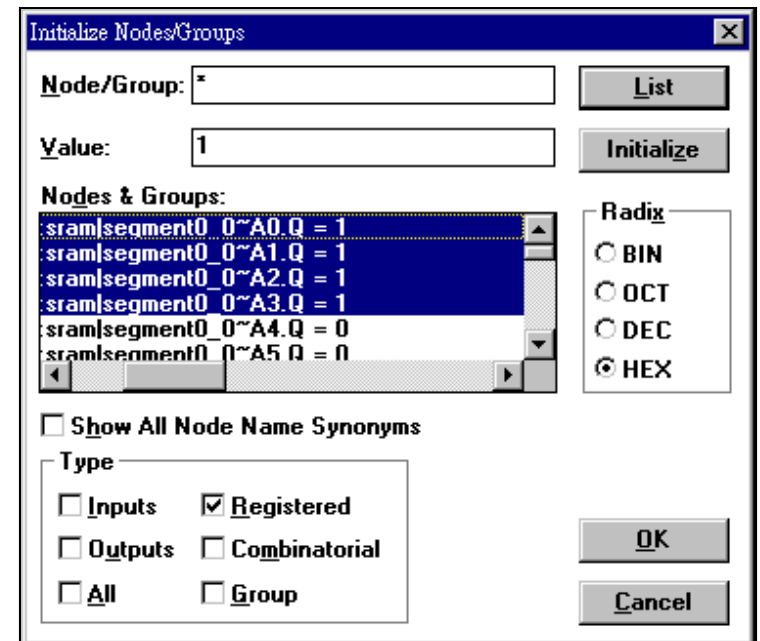
Node/Group Initialization

◆ 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...*

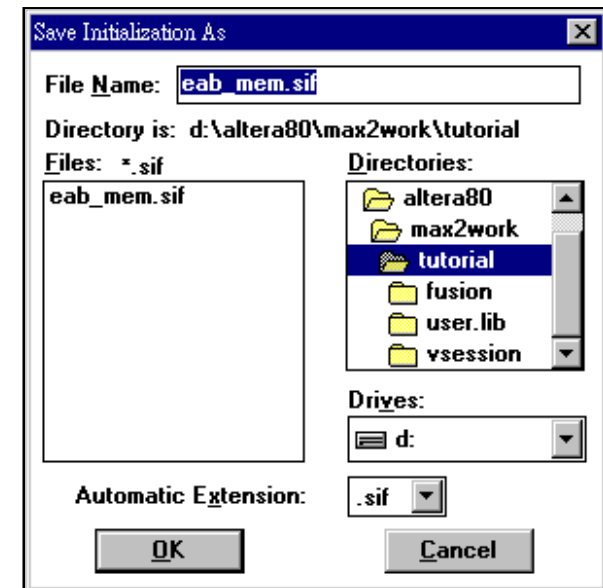
- To retrieve initialized node, group, and memory values stored in a SIF file

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



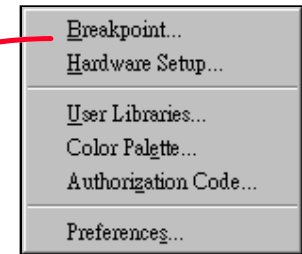
Creating Breakpoints

◆ Specify simulation breakpoints

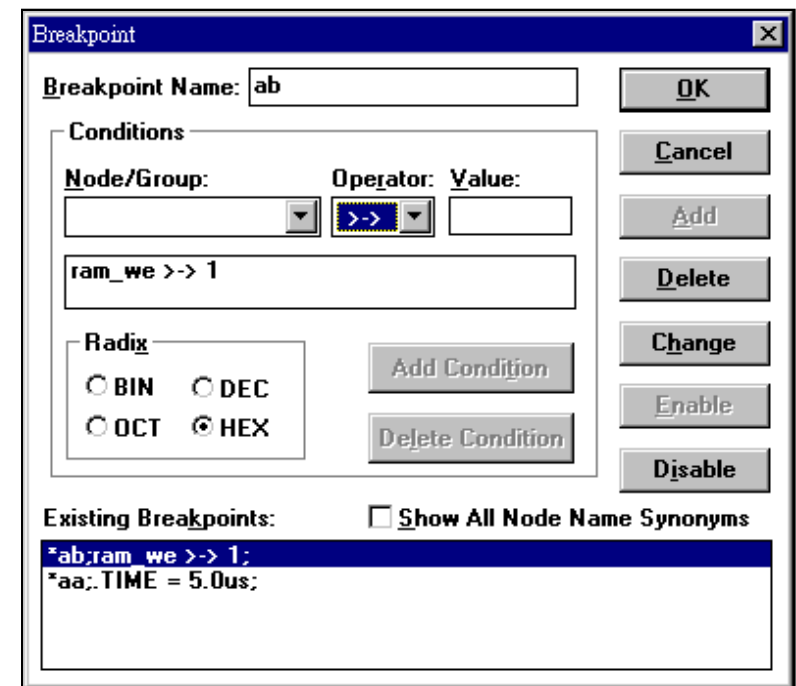
- You can create one or more breakpoints, each of which consists of one or more node value, group value, and time conditions

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



Options Menu



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

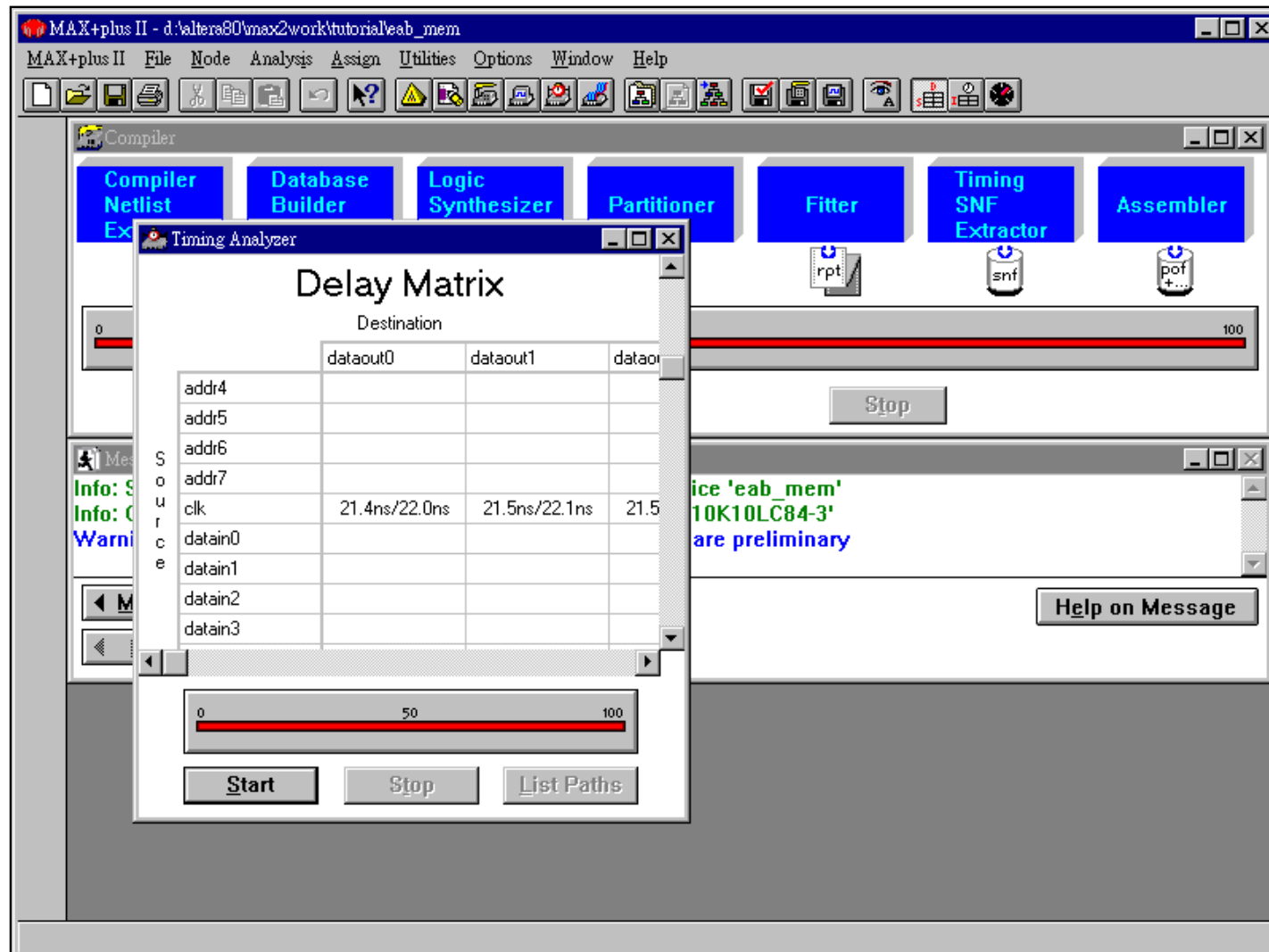
Timing Analysis

◆ Static timing analysis

- It perform path analysis, easily find the critical path and worst-case timing
- Must generate timing SNF during project compilation
- It's pattern-independent
- Invoke MAX+PLUS II Timing Analyzer to do timing analysis

Menu: *MAX+PLUS II -> Timing Analyzer*

MAX+PLUS II Timing Analyzer

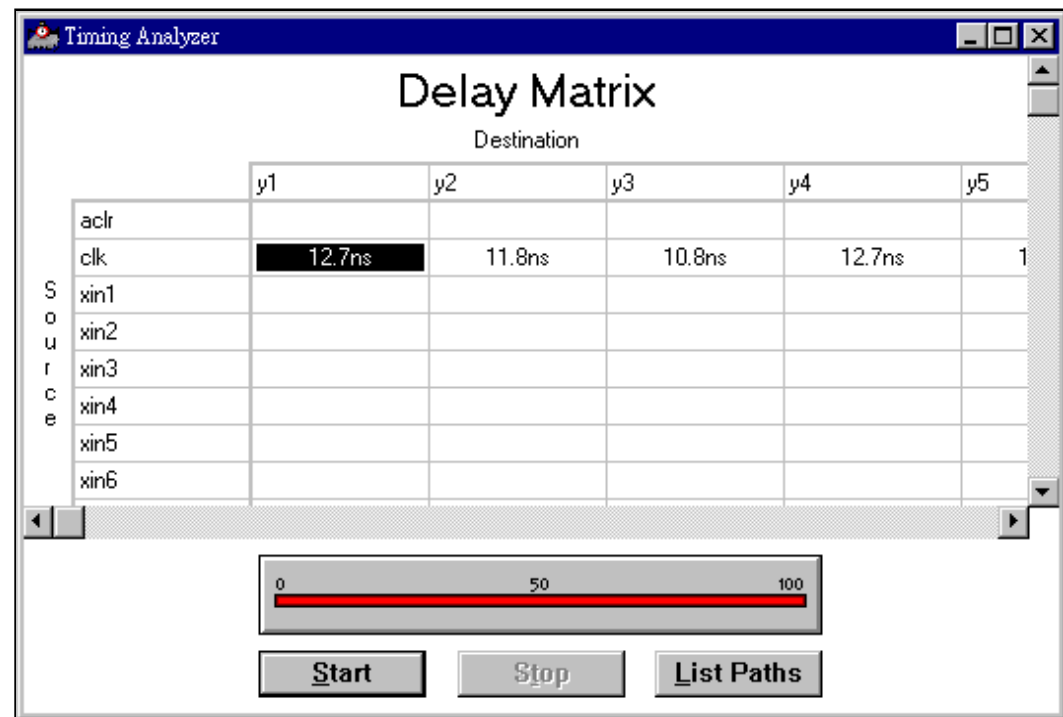


Timing Analysis Modes - (1)

◆ “Delay Matrix” analysis mode

- Delay Matrix creates a matrix for displaying propagation delays between source nodes and destination nodes in the current project

Menu: *Analysis -> Delay Matrix*

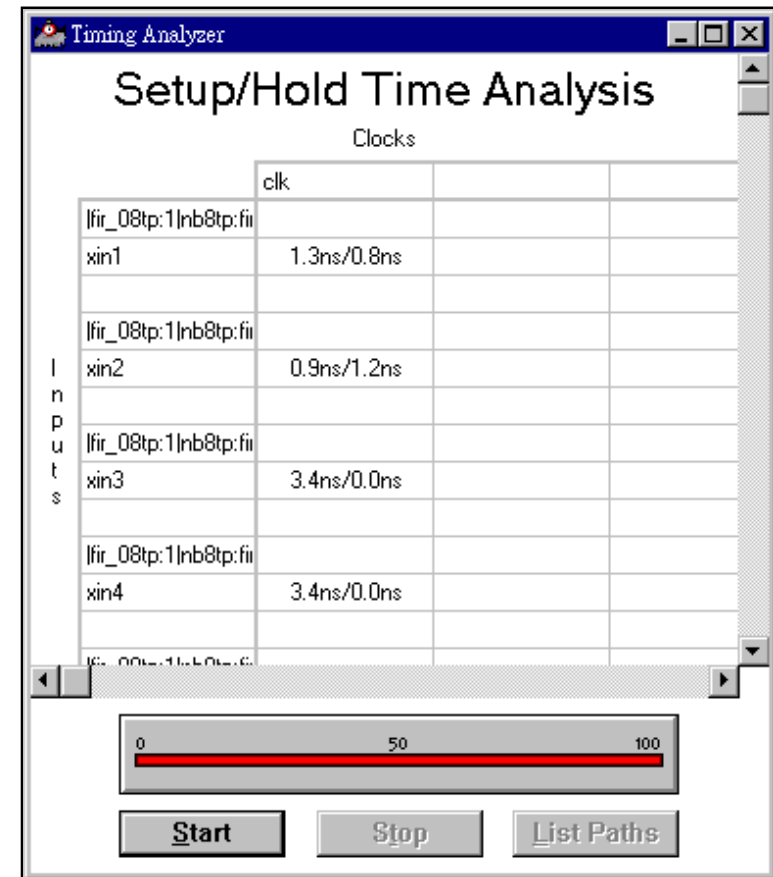


Timing Analysis Modes - (2)

◆ “Setup/Hold Matrix” analysis mode

- Setup/Hold Matrix creates a matrix for displaying setup and hold time requirements between source nodes and destination nodes in the current project

Menu: *Analysis -> Setup/Hold Matrix*

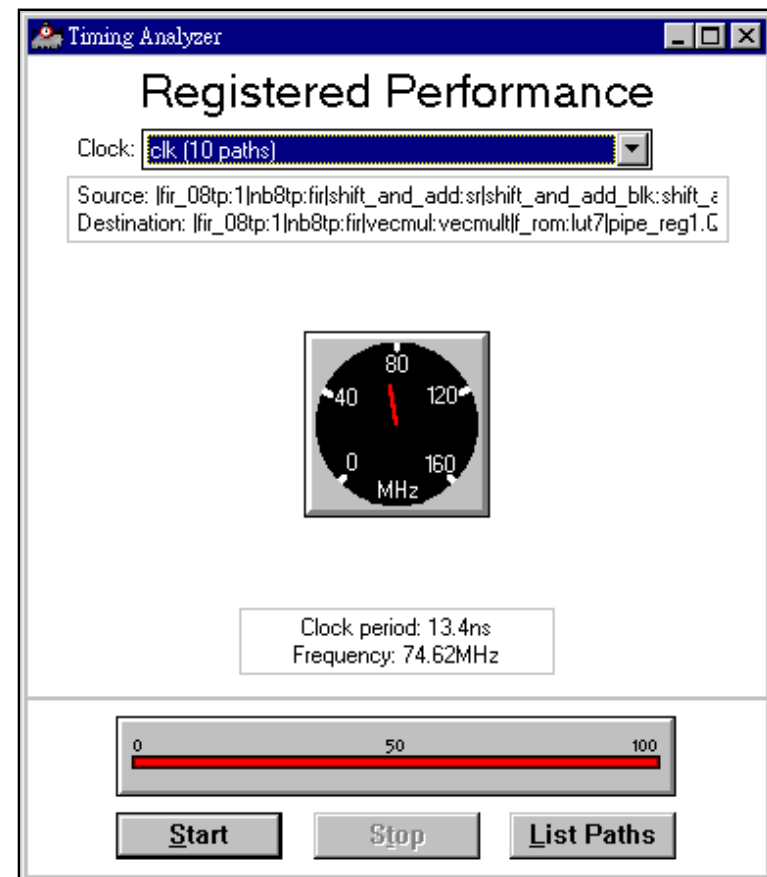


Timing Analysis Modes - (3)

◆ “Registered Performance” analysis mode

- Registered Performance displays the worst-case registered performance for each Clock signal in the current project

Menu: *Analysis -> Registered Performance*



Timing Analysis Source & Destination

◆ Specify source/destination nodes for timing analysis

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

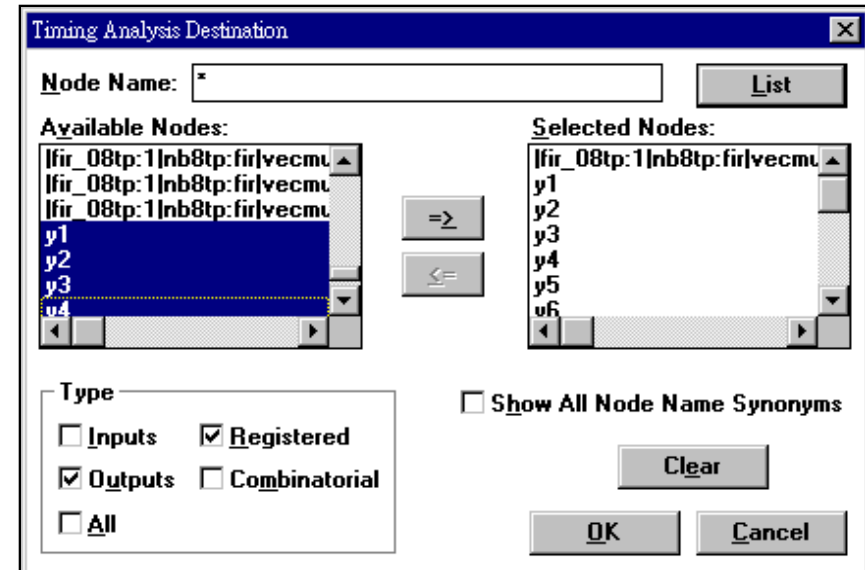
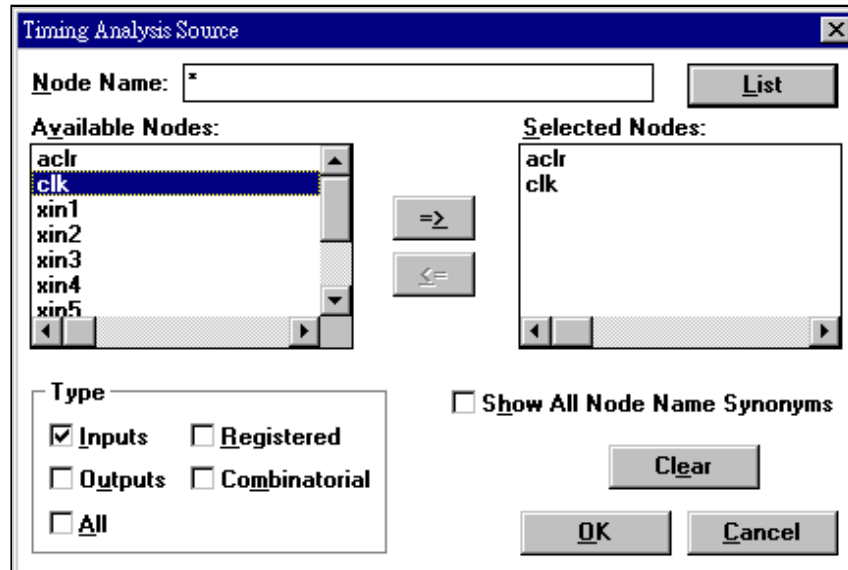
Menu: *Node -> Timing Analysis Source...*

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

Timing Analysis Source...	Ctrl+Alt+S
Timing Analysis Destination...	Ctrl+Alt+D
Timing Analysis Cutoff...	Ctrl+Alt+C

Node Menu

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



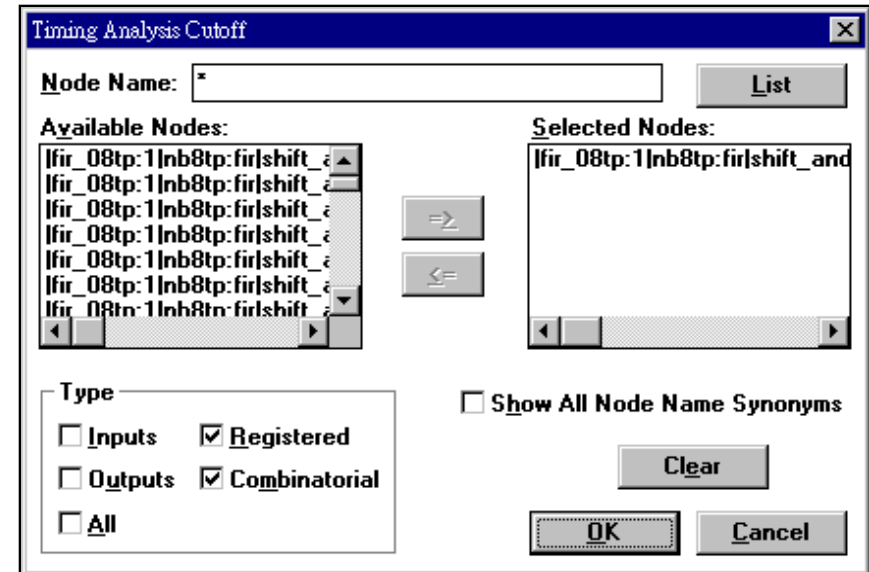
Timing Analysis Cutoff

◆ Specify cutoff point(s) for timing analysis

- You can exclude a signal path from timing analysis by tagging a node as a cutoff node

Menu: Node -> Timing Analysis Cutoff...

- When a cutoff node is tagged, only the signal path that leads to the node is included in the analysis



Timing Analyzer Options - (1)

◆ Time Restrictions option

- To define restrictions on the delay paths to be included and displayed in the next Delay Matrix or Registered Performance timing analysis

◆ Auto-Recalculate option

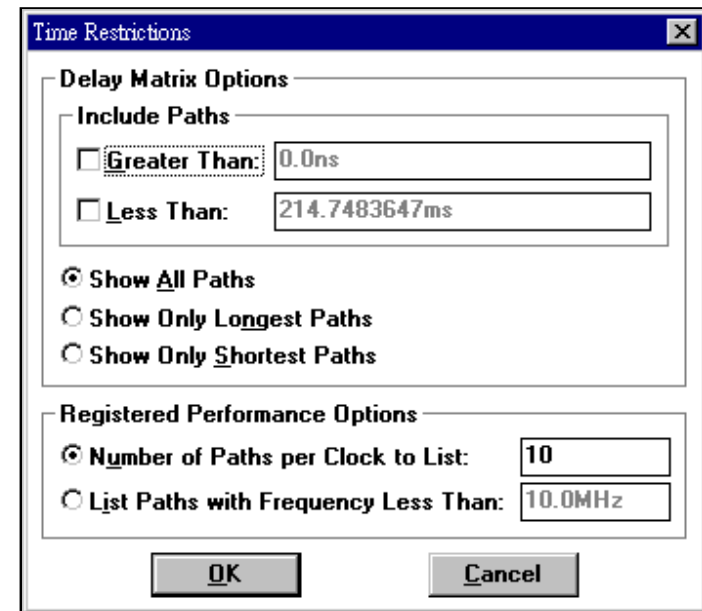
- When Auto-Recalculate is turned on, the Timing Analyzer automatically updates the current timing analysis when the Compiler creates a new timing SNF or linked SNF

◆ Cell Width option

- To specify the width of each cell in the current timing analysis matrix



Options Menu



Timing Analyzer Options - (2)

◆ Cut Off I/O Pin Feedback option

- To cut off the delay that is fed back from a bidirectional (BIDIR) pin during timing analysis
 - It is especially useful when a bidirectional pin is connected directly or indirectly to both the input and the output of a latch. This option allows you to easily eliminate the false paths caused by latch-to-pin loops

◆ Cut Off Clear & Preset Paths option

- To cut off the timing paths for all clear and preset signals in a project, so that the delays along these paths are not included in the timing analysis
 - Cutting off these signal paths may help eliminate invalid paths from the timing analysis

◆ List Only Longest Path

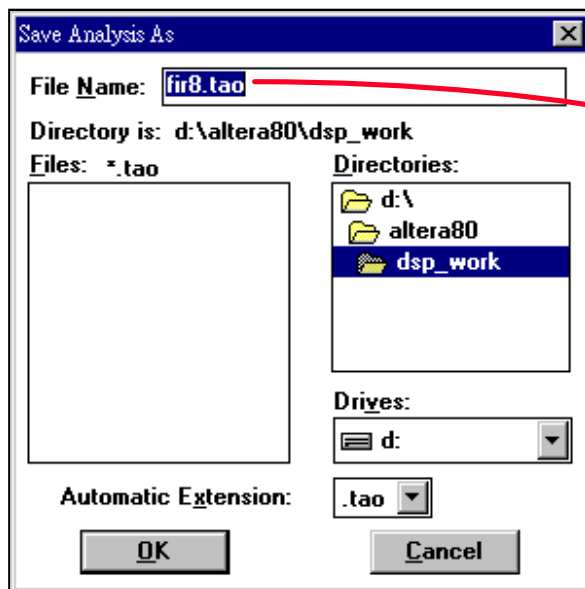
- To direct the Timing Analyzer to calculate only the longest delay path between a pair of nodes when you choose the List Paths button

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		
		y3	y4	y5
S	aclr	.	.	.
o	clk	10.8ns	12.7ns	11.7ns
u	xin1	.	.	.
r	xin2	.	.	.
c	xin3	.	.	.
e	xin4	.	.	.
	xin5	.	.	.
	xin6	.	.	.
	xin7	.	.	.
	xin8	.	.	.

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

The screenshot shows the MAX+plus II Timing Analyzer window. The main window displays a Delay Matrix table. Below the table is a progress bar and buttons for Start, Stop, and List Paths. A red arrow points from the List Paths button to the Messages - Timing Analyzer window. The Messages window shows a message: "Info: Delay path from 'clock' to 'at_altera': 10.5ns". A red arrow points from the Locate button in the Messages window to the code editor. The code editor shows a VHDL snippet for a machine with states and outputs.

Delay Matrix

	Destination				
	at_altera	ticket0	ticket1	ticket2	ticket3
accel					
clock	10.5ns	10.6ns	10.5ns	10.2ns	10.4ns
dir0					
dir1					
enable					
reset	10.7ns				

Messages - Timing Analyzer

Info: Delay path from 'clock' to 'at_altera': 10.5ns

1 of 1

Locate

Locate All

```
VARIABLE
street_map : MACHINE
    OF BITS (q2,q1,q0)
    WITH STATES (
        yc,
        mp1d,
        ep1d,
        gdf,
        cnf,
        a );
clk; % inp
```

Locating Delay Paths in Floorplan

◆ To trace delay paths in Floorplan Editor

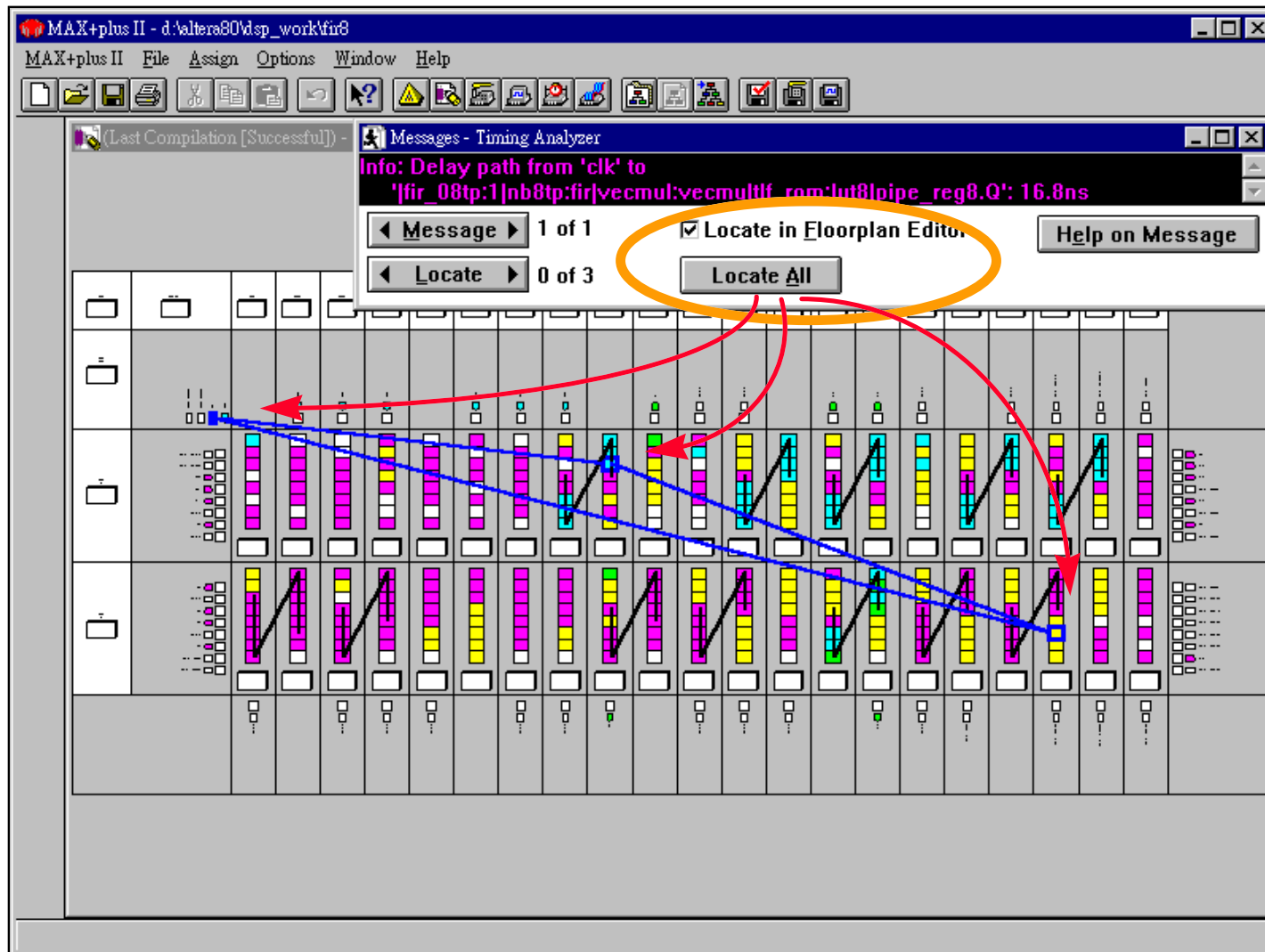
- You can locate the selected signal paths in the Floorplan Editor
- Select the matrix cell or clock, click List Paths
- Turn on Locate in Floorplan Editor
- Click Locate All to invoke Floorplan Editor
- In Floorplan Editor, you can show fan-in, fan-out or path of the nodes

Menu: *Options -> Show Node Fan-In*

Menu: *Options -> Show Node Fan-Out*

Menu: *Options -> Show Path*

Locating Paths in Floorplan Editor



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

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

Programming Methods

◆ For EEPROM-based devices (MAX 7000/E/S, MAX 9000)

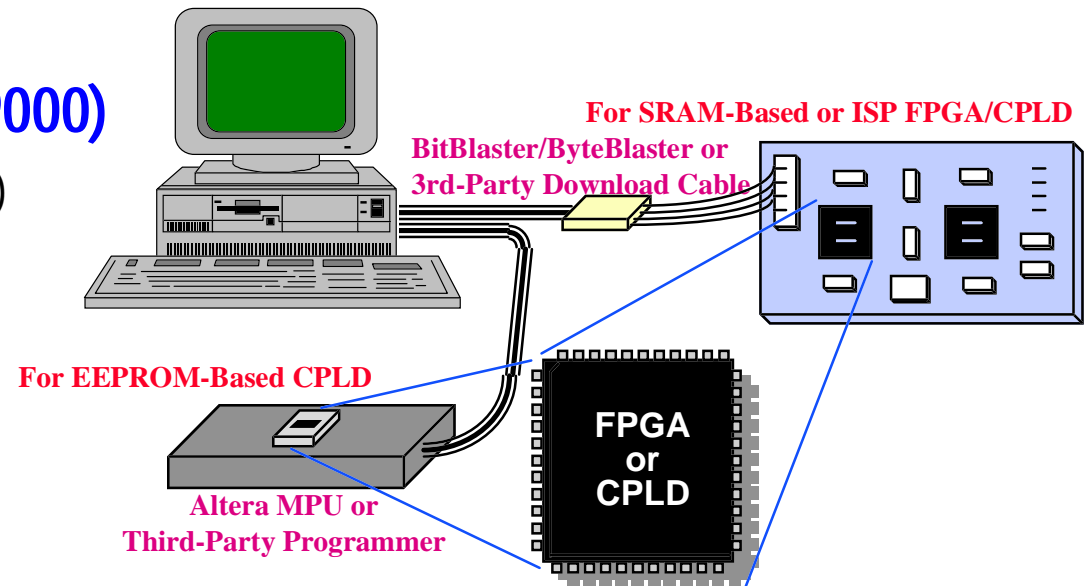
- Use hardware programmer (must generate .pof file)

◆ For SRAM-based devices (all FLEX devices)

- Use download cable (using .sof file)
- Program on a serial PROM (using .pof file) and attach it on board
- Program on a parallel EPROM (using .hex file) and attach it on board
- Other methods

◆ For ISP (MAX 7000S, MAX 9000)

- Use download cable (using .pof file)



Programming MAX Devices

◆ Program the device with external hardware

- Use Altera hardware programmer
 - Altera MAX 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 devices

◆ MAX 7000S & MAX 9000 ISP

- Altera MAX 7000S and 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
- ISP device can internally generate 12.0-V programming voltage

Configuring FLEX 8000A Devices

◆ FLEX 8000A configuration schemes & data source

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

	Configuration Scheme	Data Source
AS	(Active Serial)	Serial configuration EPROM
APU	(Active Parallel Up)	Parallel EPROM
APD	(Active Parallel Down)	Parallel EPROM
PS	(Passive Serial)	Serial data path (e.g. serial download cable)
PPS	(Passive Parallel Synchronous)	Intelligent host
PPA	(Passive Parallel Asynchronous)	Intelligent host

Configuring FLEX 10K Devices

◆ 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*

Configuration Scheme	Data Source
PS (Passive Serial)	Altera's EPC1 configuration EPROM, BitBlaster or ByteBlaster download cable, serial data source
PPS (Passive Parallel Synchronous)	Intelligent host, parallel data source
PPA (Passive Parallel Asynchronous)	Intelligent host, parallel data source
JTAG	JTAG controller

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

Device	Data Size(bits)	Device	Data Size(bits)
EPF8282A/V	40,000	EPF10K10	115,000
EPF8452A	64,000	EPF10K20	225,000
EPF8636A	96,000	EPF10K30	368,000
EPF8820A	128,000	EPF10K40	488,000
EPF81188A	192,000	EPF10K50	609,000
EPF81500A	250,000	EPF10K70	881,000
		EPF10K100	1,172,000

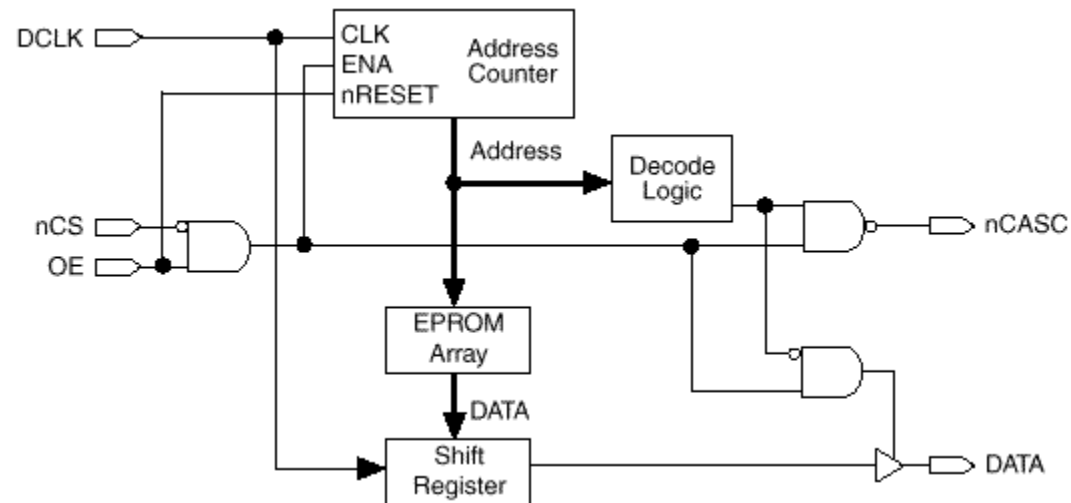
Altera Configuration EPROM Family

◆ 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

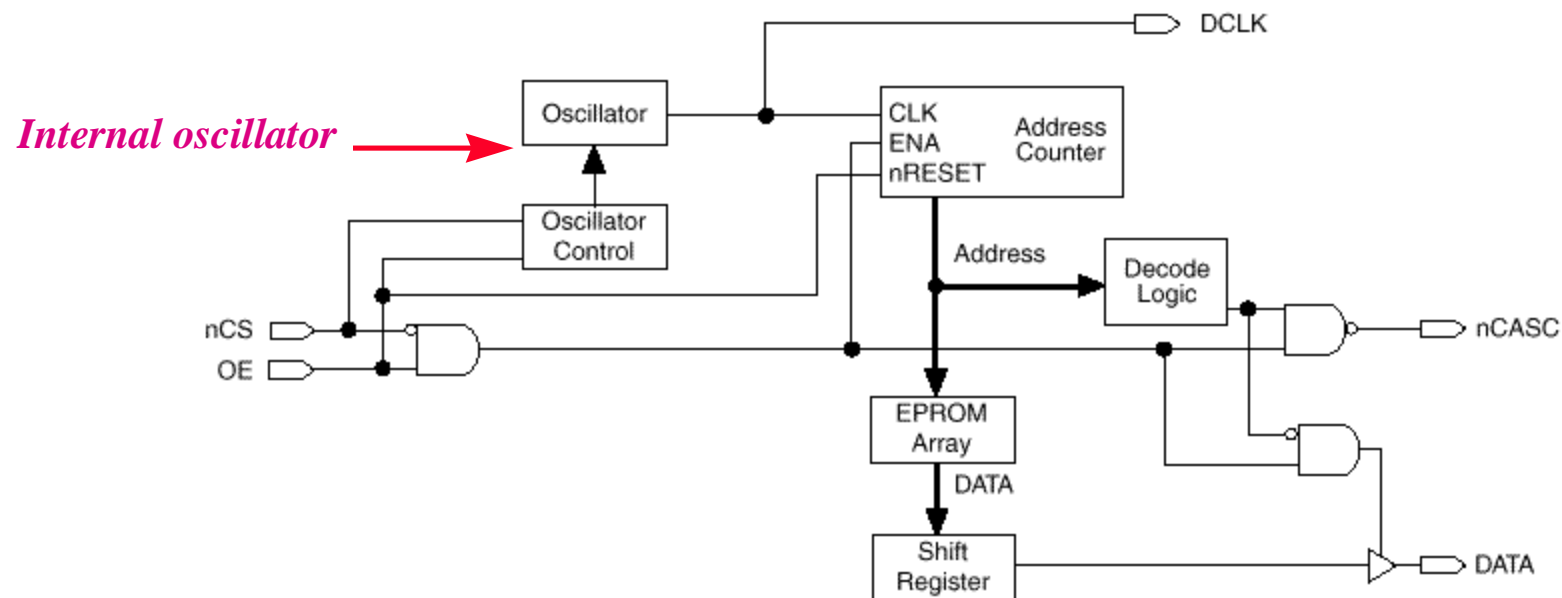
Configuration EPROM Block Diagram - (1)

◆ 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

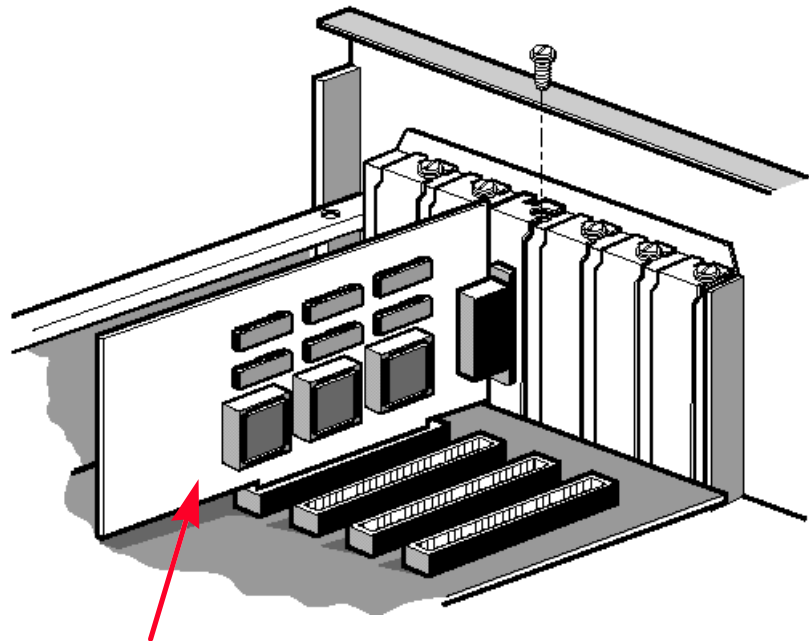
- For MAX 7000/E/S, MAX 9000 and Altera configuration EPROM(EPC- series) devices
 - Altera stand-alone programmer: PL-ASAP2 (PC platform)
 - 3rd-party universal programmer (PC platform)
- For MAX 7000S and MAX 9000 ISP, FLEX devices downloading
 - Altera BitBlaster download cable (RS-232 port)
 - Altera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

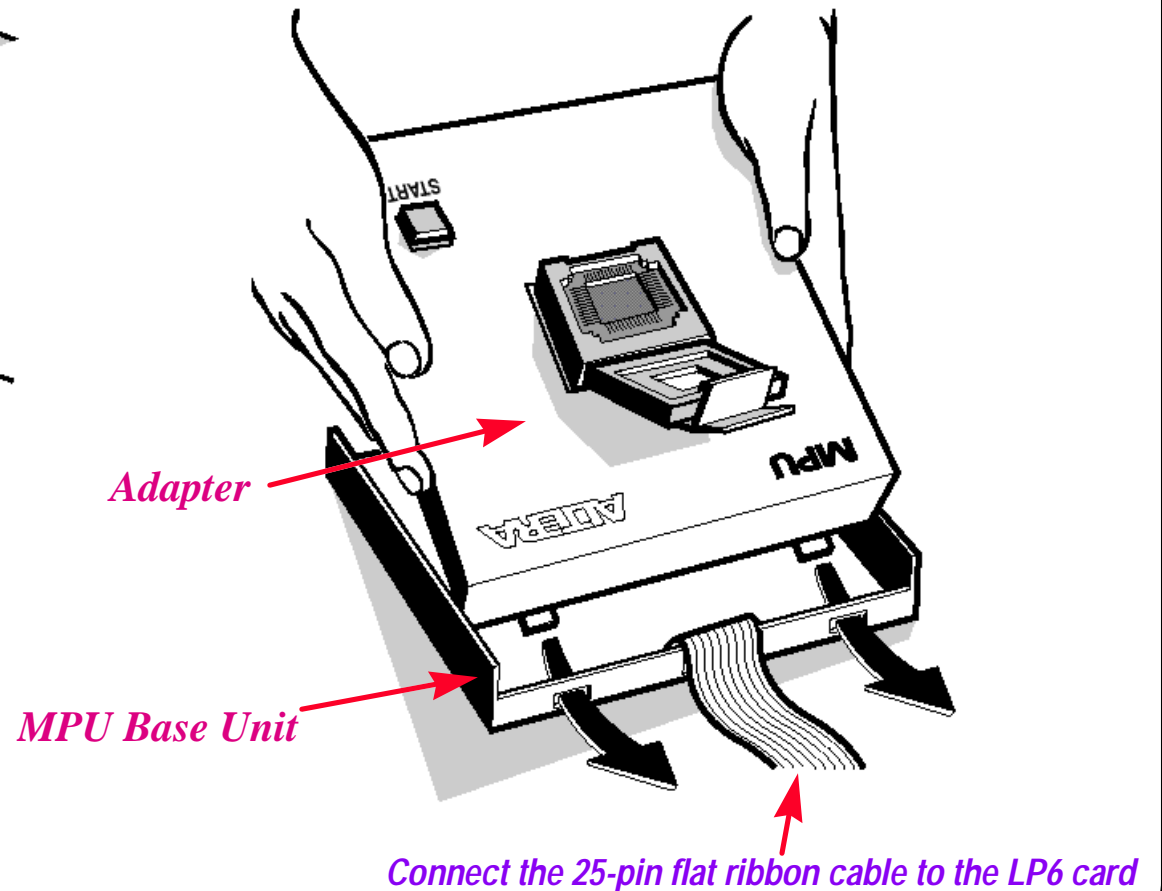
◆ PL-ASAP2: Altera stand-alone hardware programmer

- 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



LP6 Programmer Card

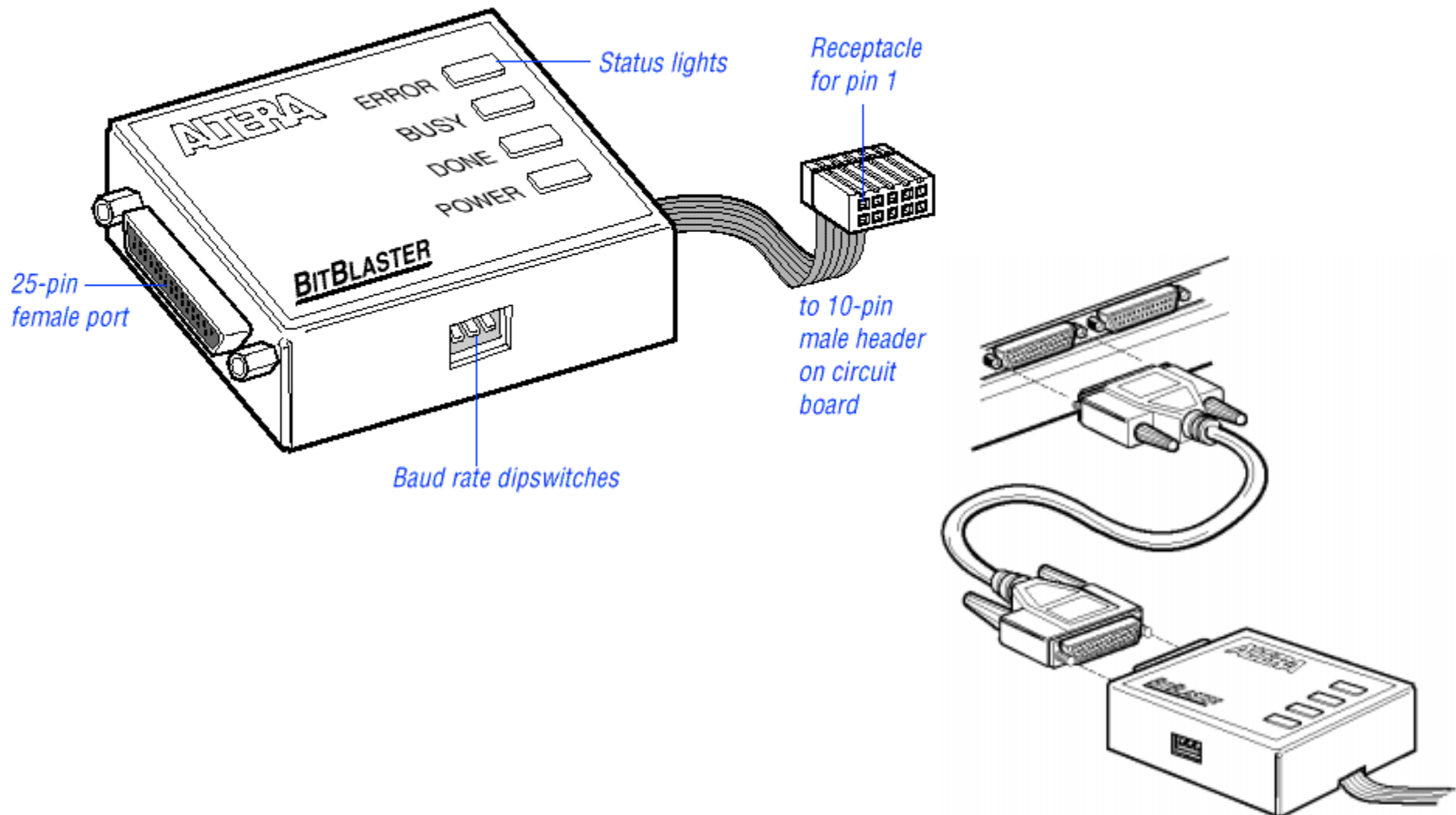


BitBlaster Download Cable

◆ 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

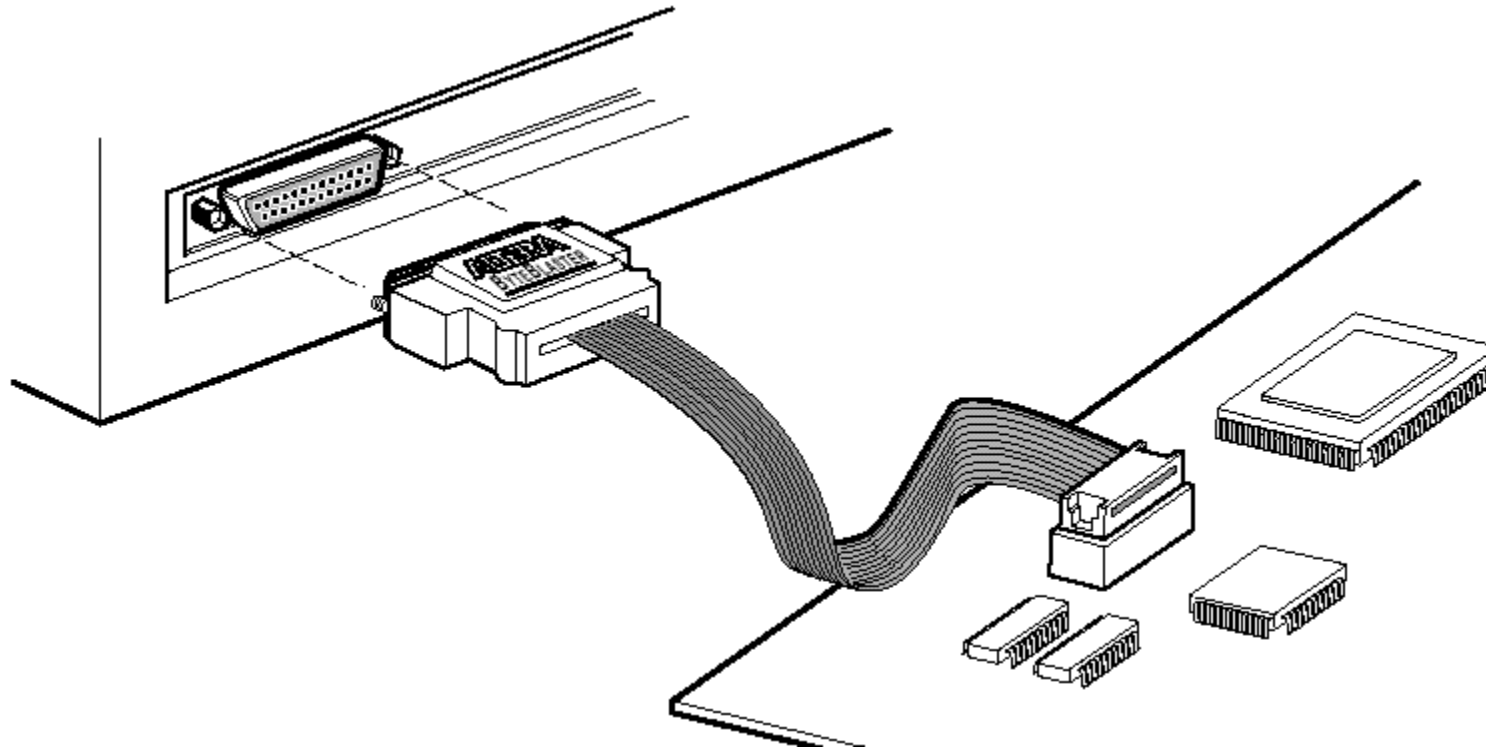


ByteBlaster Download Cable

◆ Altera ByteBlaster parallel port download cable

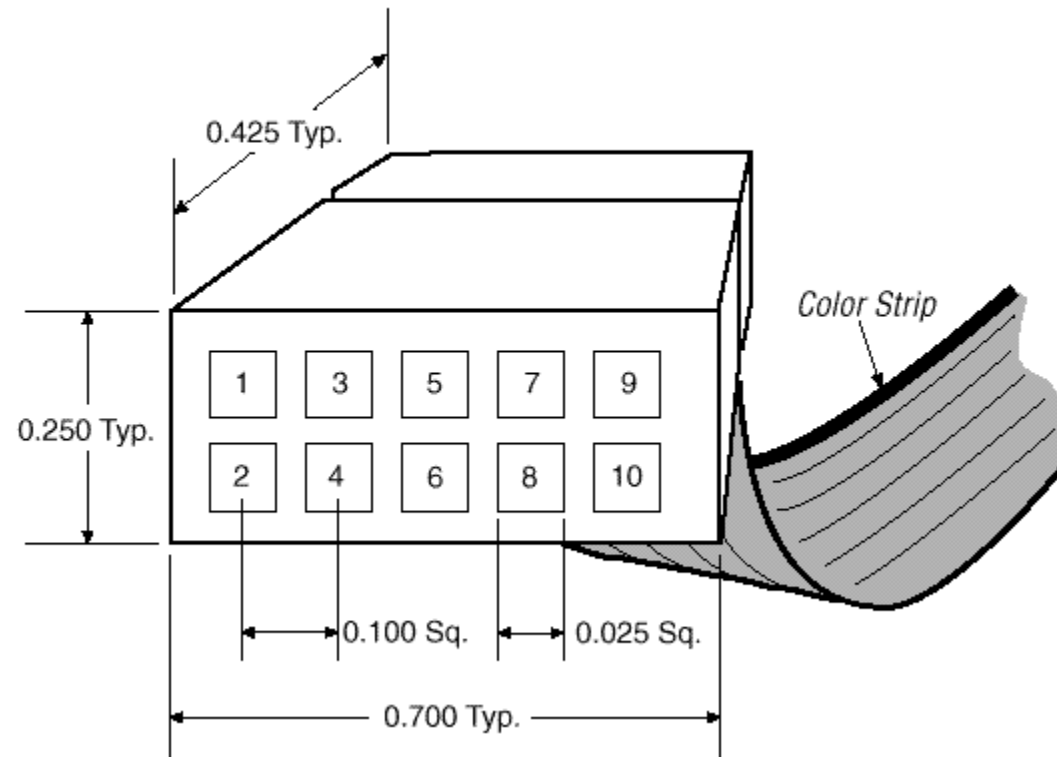
- ByteBlaster serial download cable allows PC users to
 - Program MAX 9000, MAX 7000S in-system via a standard parallel port
 - Configure FLEX devices in circuit via a standard parallel port
- ByteBlaster provides two download modes
 - Passive Serial(PS) mode: used for configuring all FLEX devices
 - 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.



<u>Pin</u>	<u>PS Mode</u>	<u>JTAG Mode</u>
1	DCLK	TCK
2	GND	GND
3	CONF_DONE	TDO
4	VCC	VCC
5	nCONFIG	TMS
6	N.C.	N.C.
7	nSTATUS	N.C.
8	N.C.	N.C.
9	DATA0	TDI
10	GND	GND

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

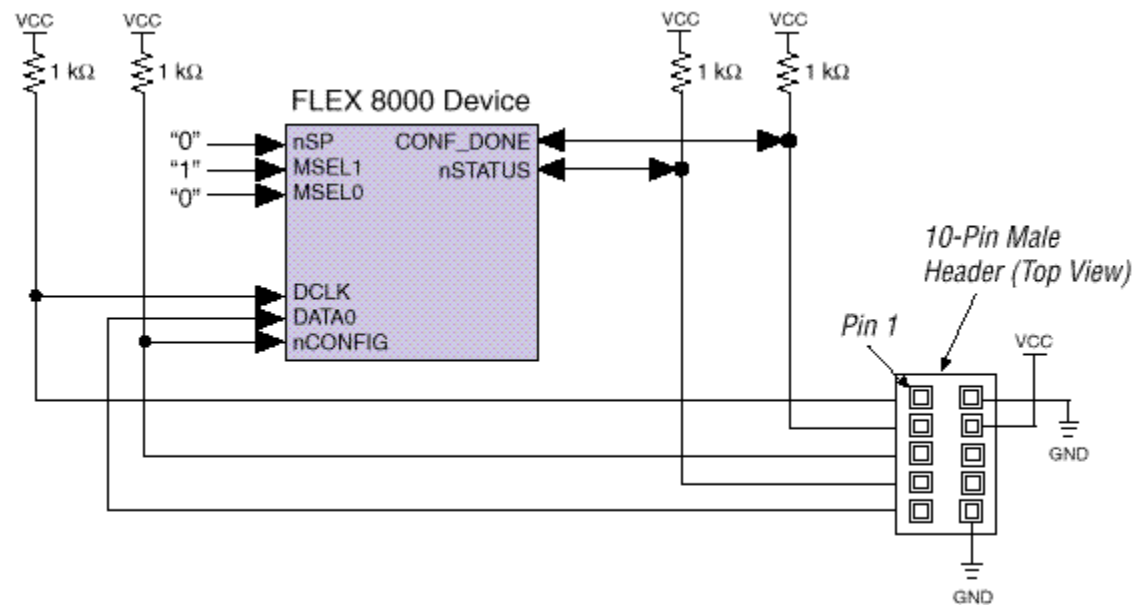
◆ Serial configuration EPROM configuration

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

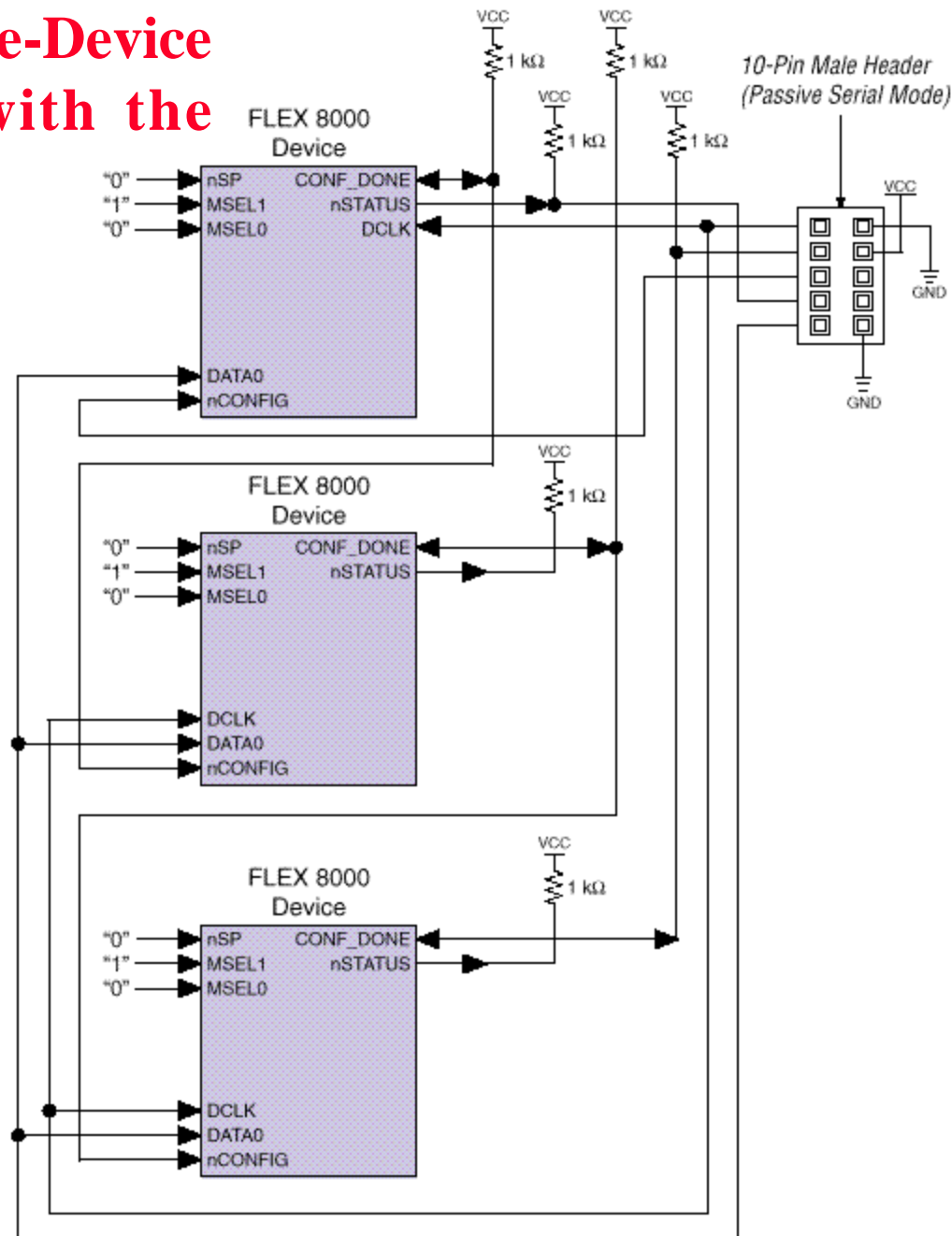
◆ Parallel EPROM configuration

- FLEX 8000A Active Parallel Up(APU) or Active Parallel Down(APD) configuration
- Not available for FLEX 10K devices

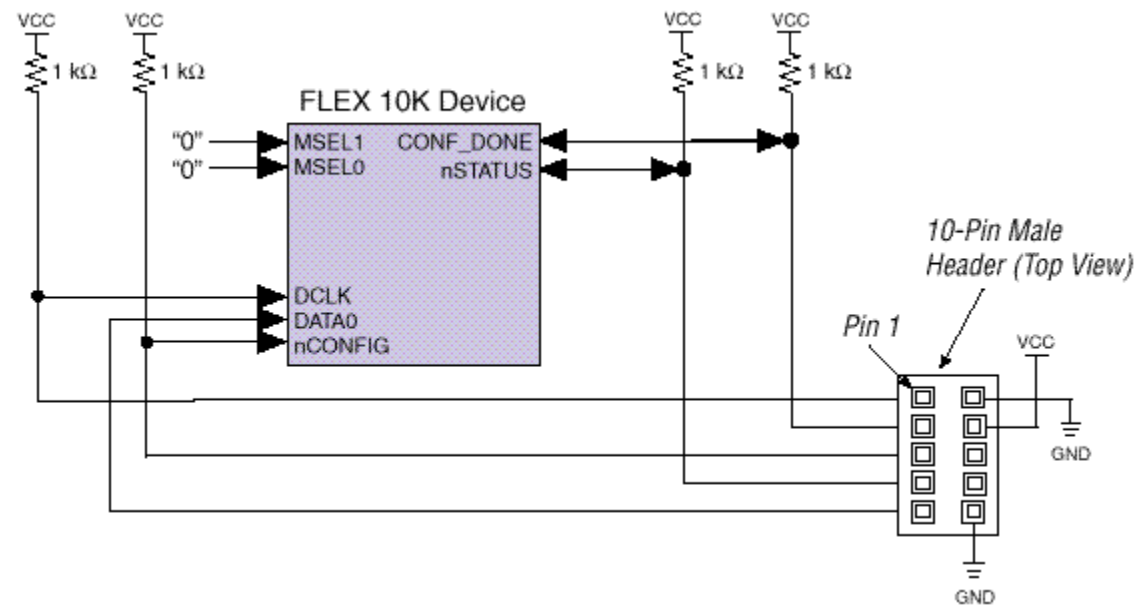
FLEX 8000A Single-Device PS Configuration with the Download Cable



FLEX 8000A Multiple-Device PS Configuration with the Download Cable

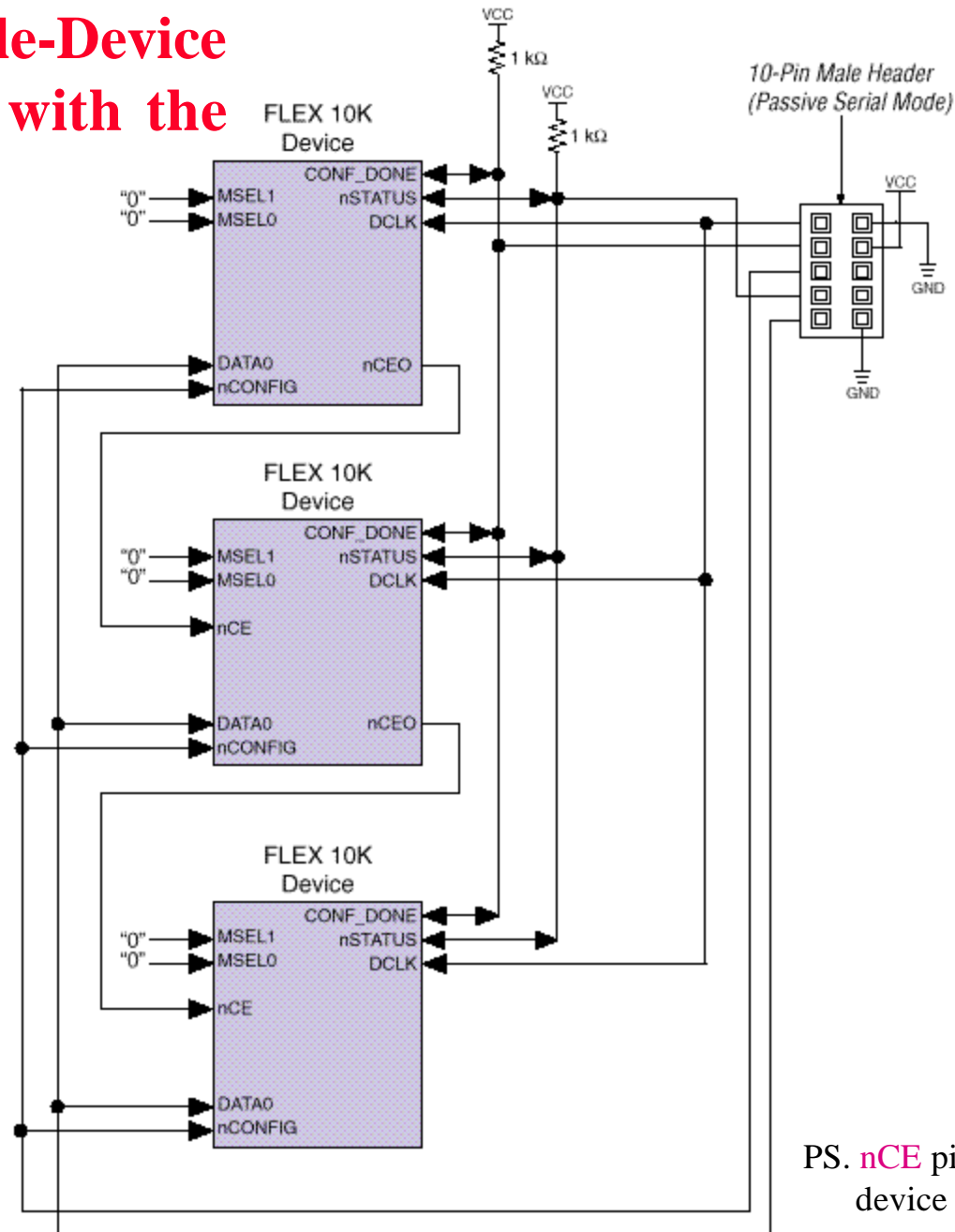


FLEX 10K Single-Device PS Configuration with the Download Cable



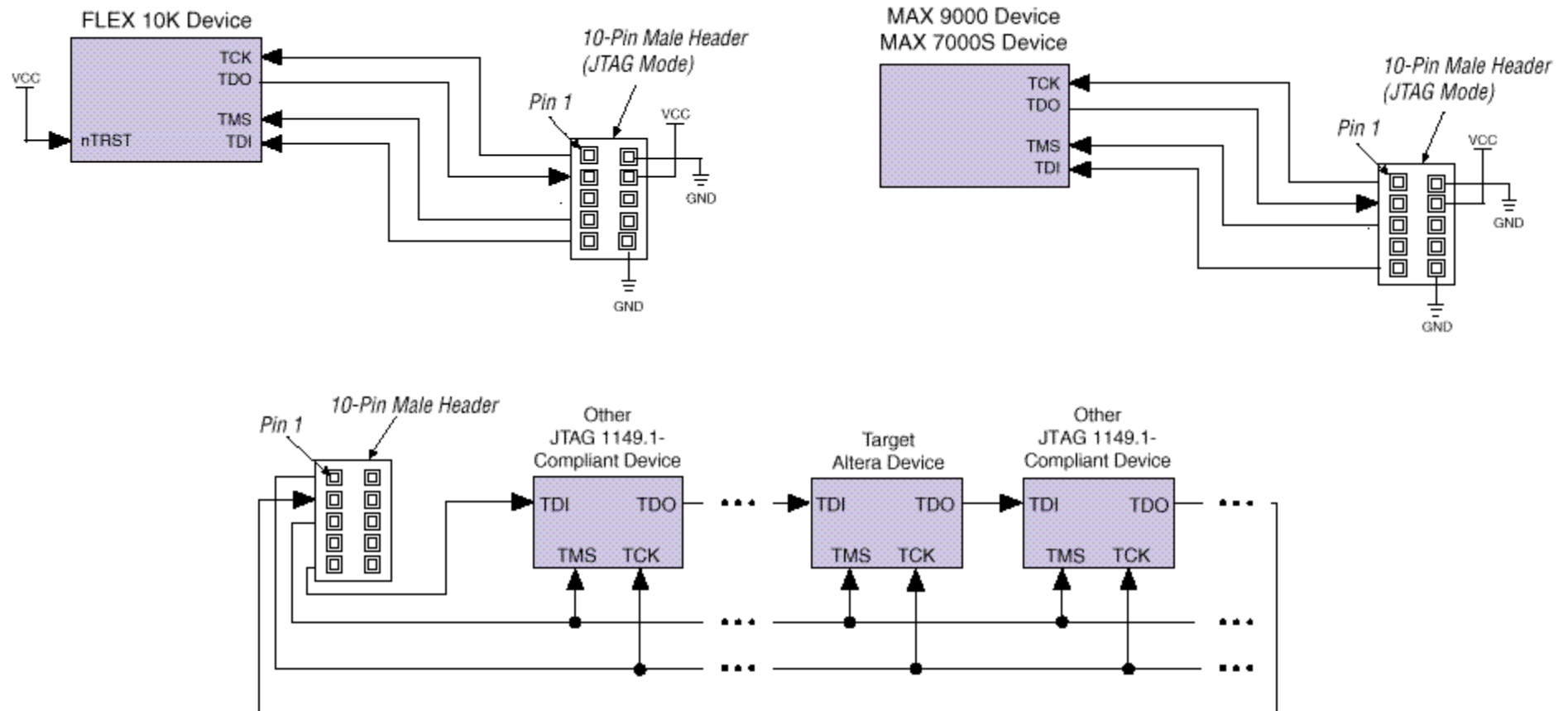
PS. **nCE** pin of FLEX 10K device must connect to GND.

FLEX 10K Multiple-Device PS Configuration with the Download Cable

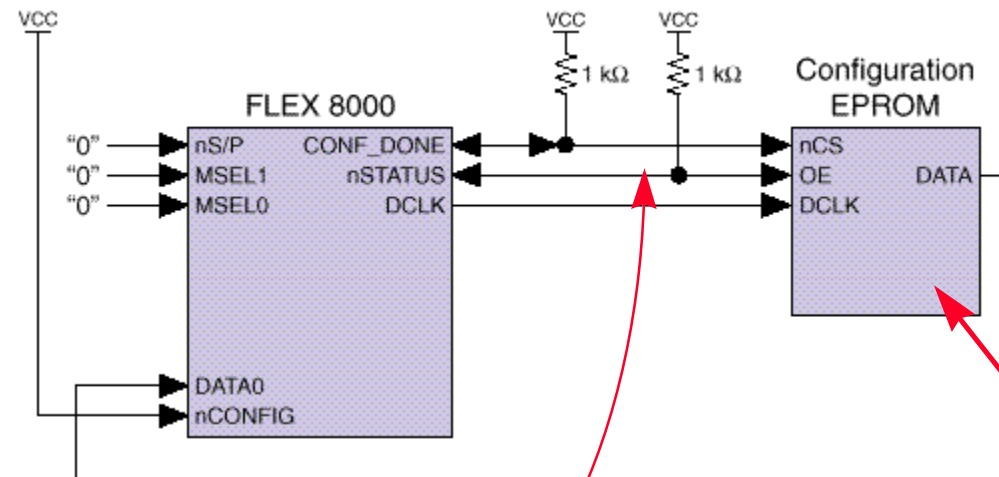


PS. **nCE** pin of the lead FLEX 10K device must connect to GND.

JTAG Configuration with the Download Cable



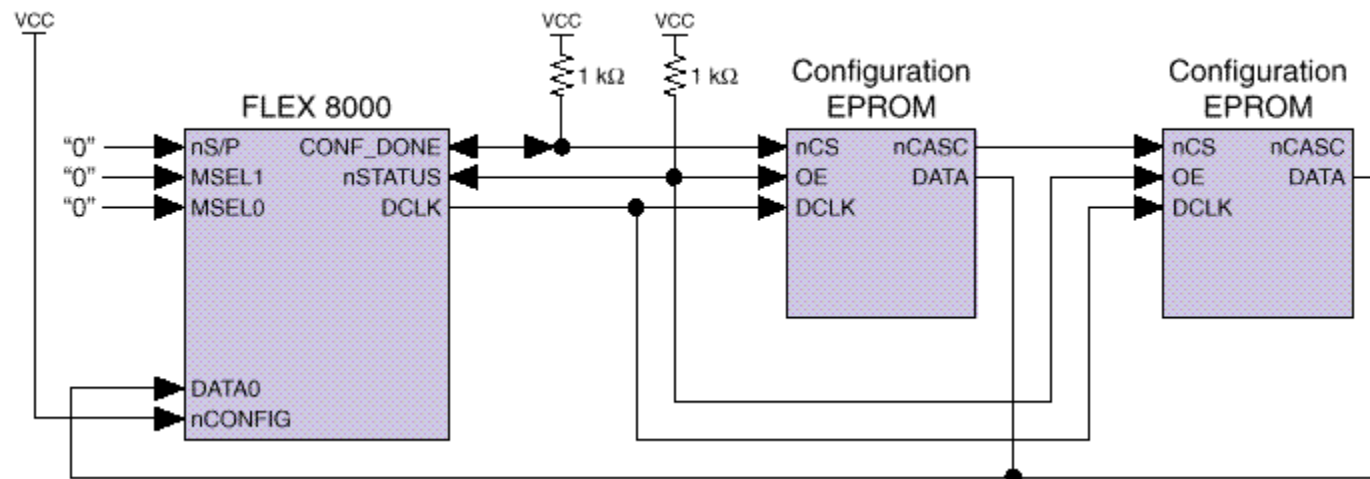
FLEX 8000A Configuration EPROM Configuration (AS Mode)



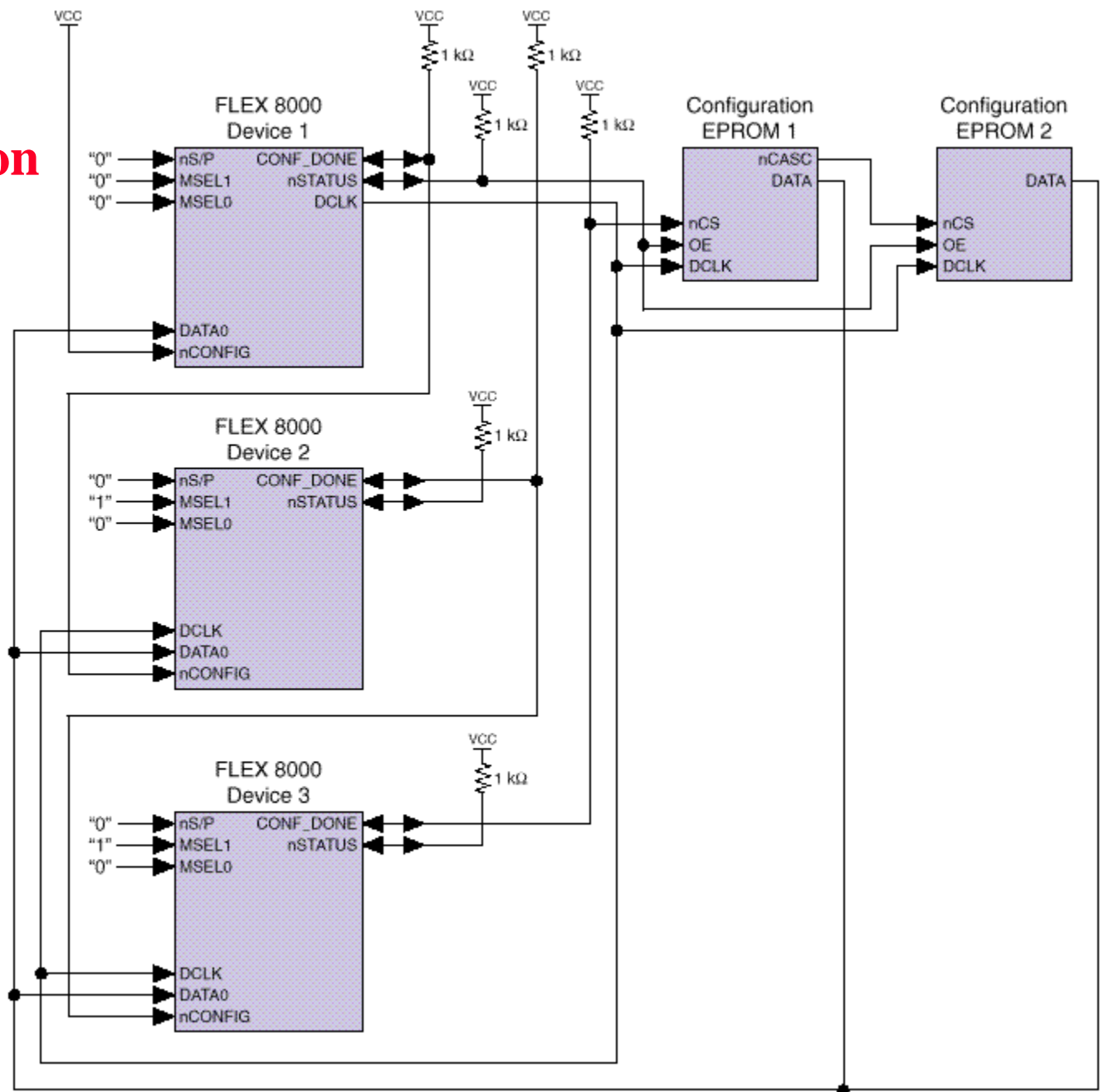
AS configuration with automatic reconfiguration on error (nStatus pin is connected to OE pin of the configuration EPROM and when "Auto-Restart Configuration on Frame Error" option bit is turned on)

Serial configuration EPROM (e.g. Altera's EPC1213)

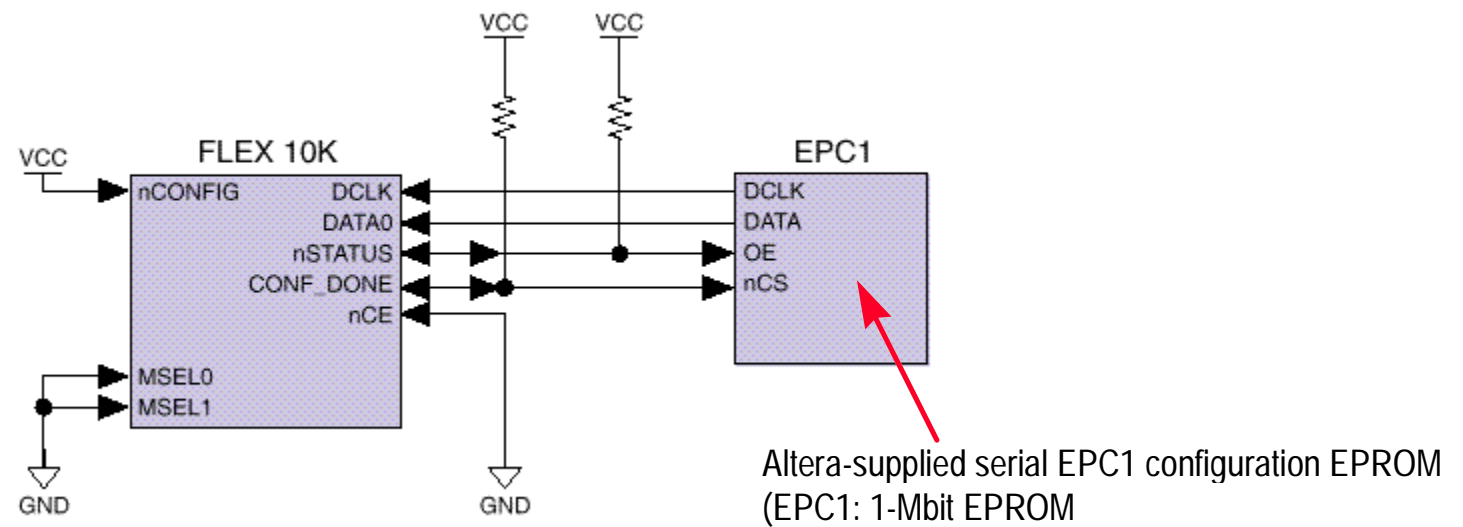
FLEX 8000A Multiple Configuration EPROMs Configuration



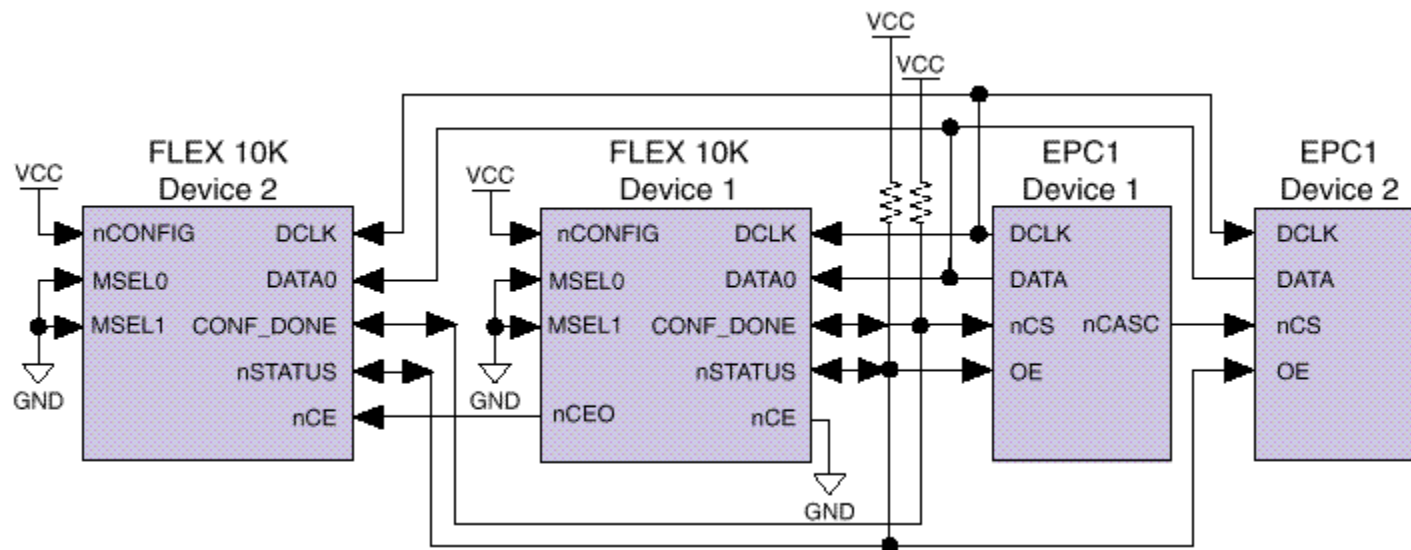
FLEX 8000A Multi-Device Configuration EPROM Configuration



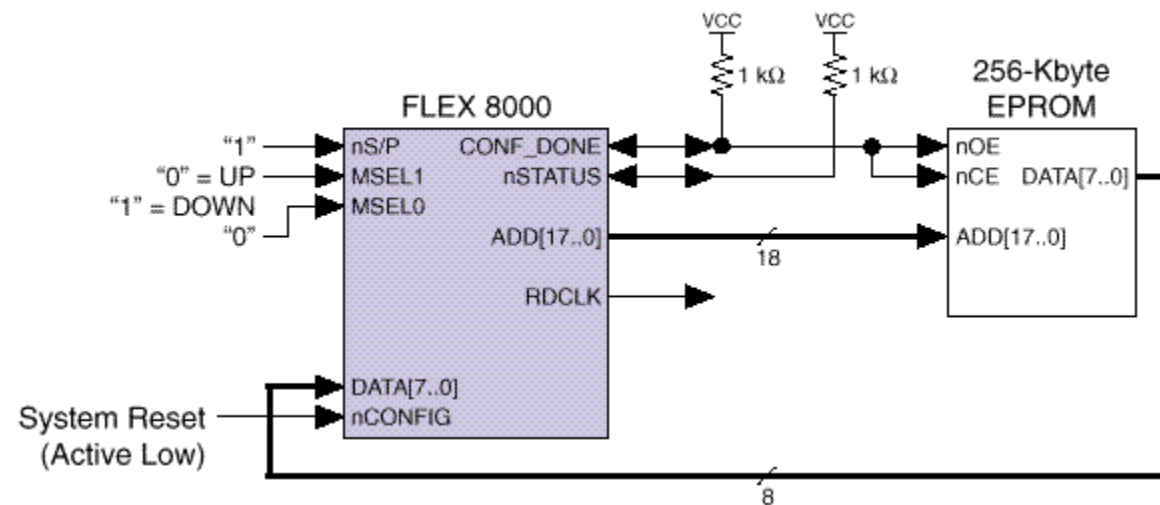
FLEX 10K Configuration EPROM Configuration (PS Mode)



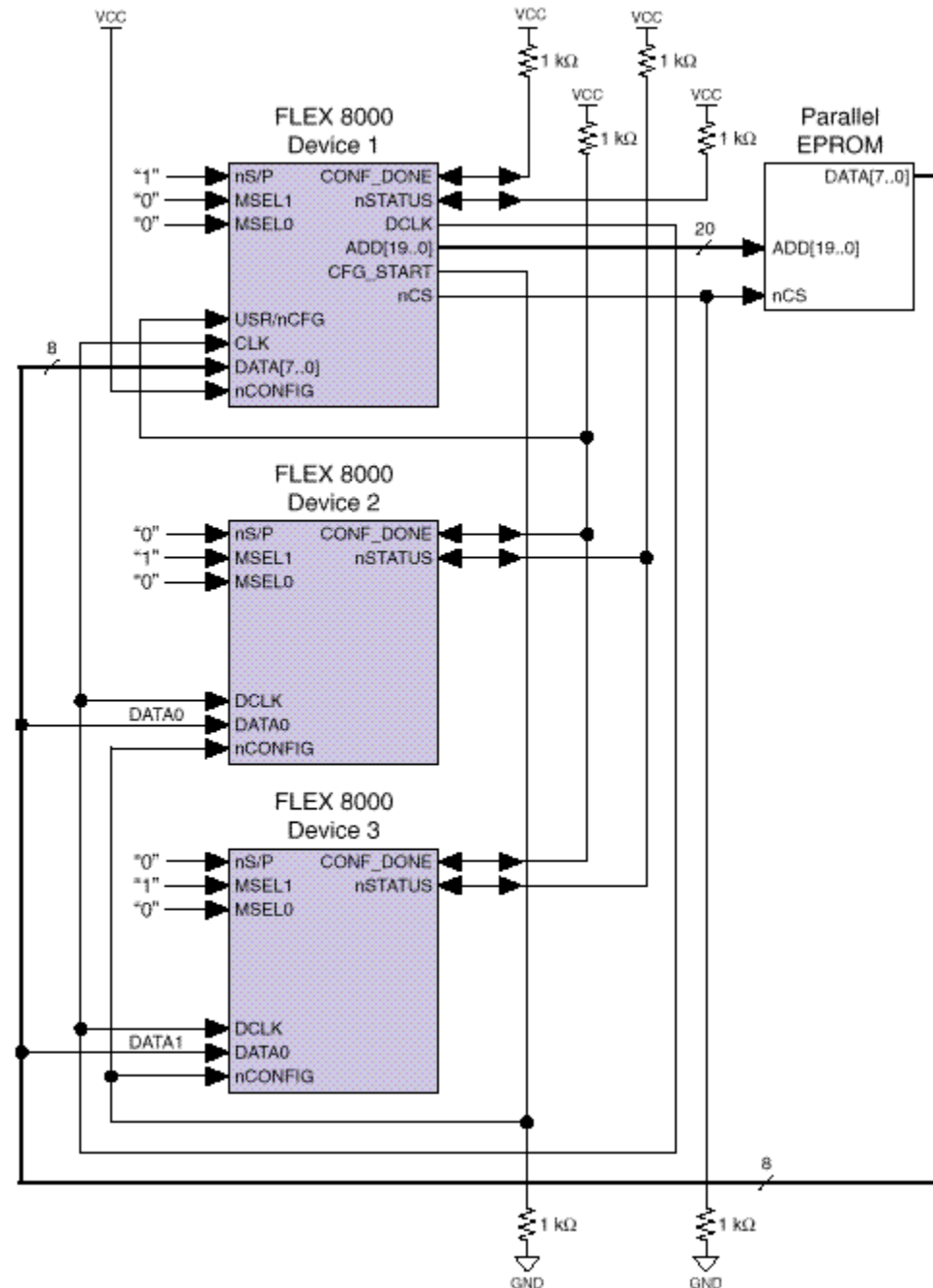
FLEX 10K Multi-Device Configuration EPROM Configuration



FLEX 8000A APU & APD Configuration



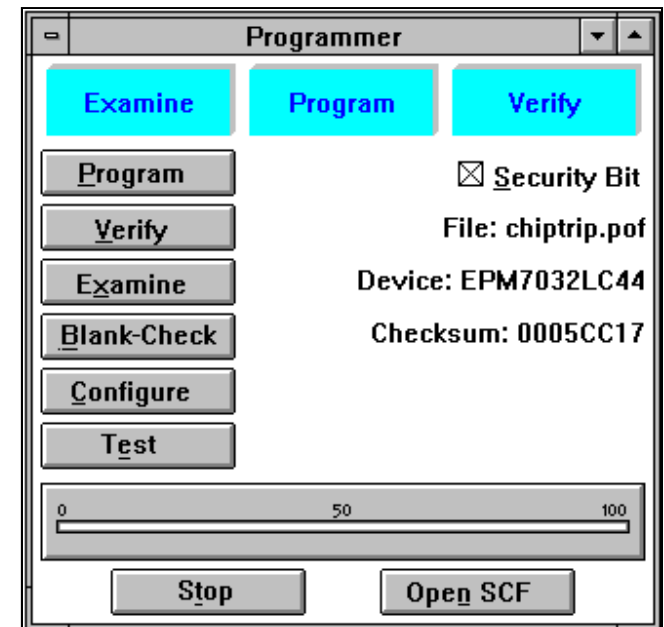
FLEX 8000A Multi-Device APU & APD Configuration



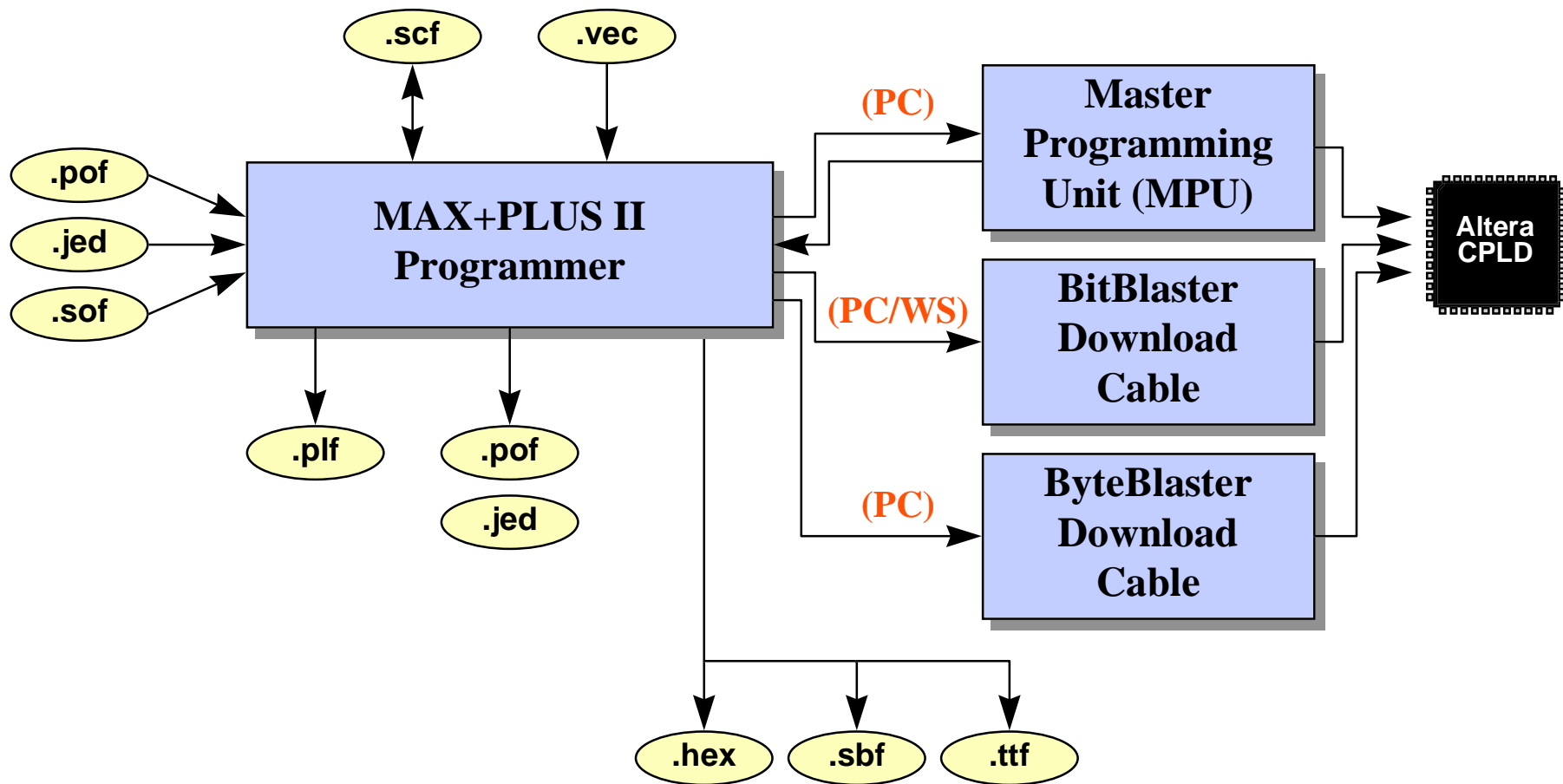
MAX+PLUS II Programmer

◆ 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

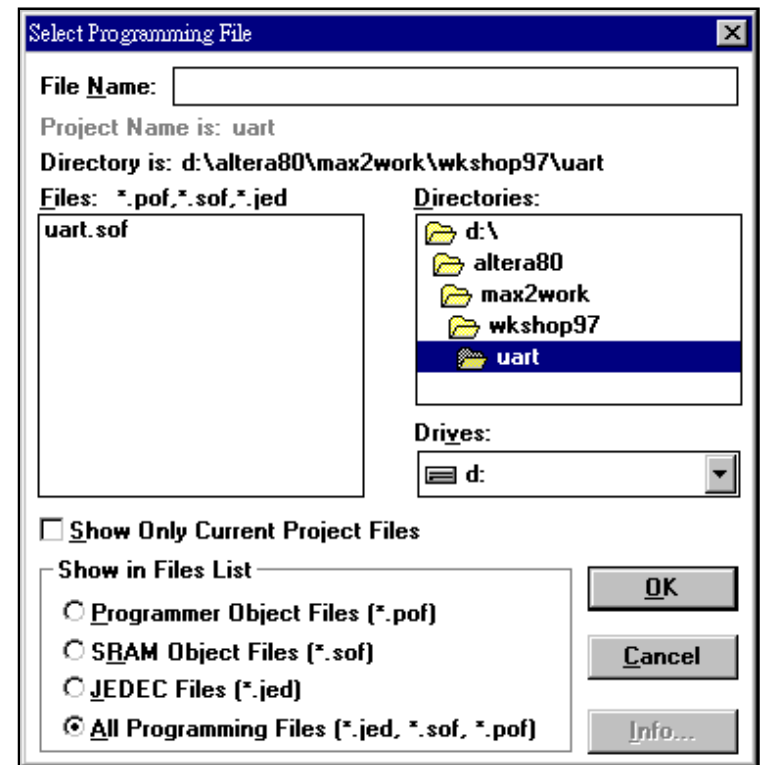
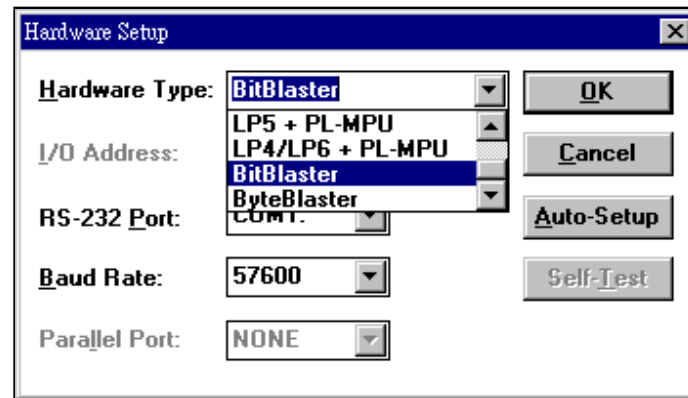
- Setup the hardware

Menu: Options -> Hardware Setup... -> Auto-Setup

- Specify the programming file

Menu: File -> Select Programming File...

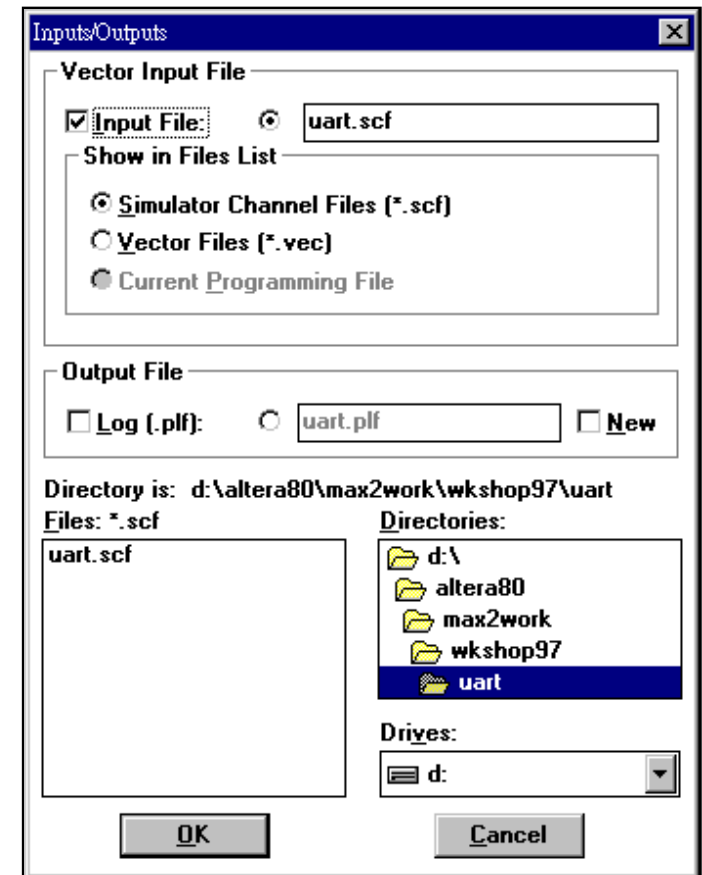
- Program or configure the device: just click on the Program or Configure button



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

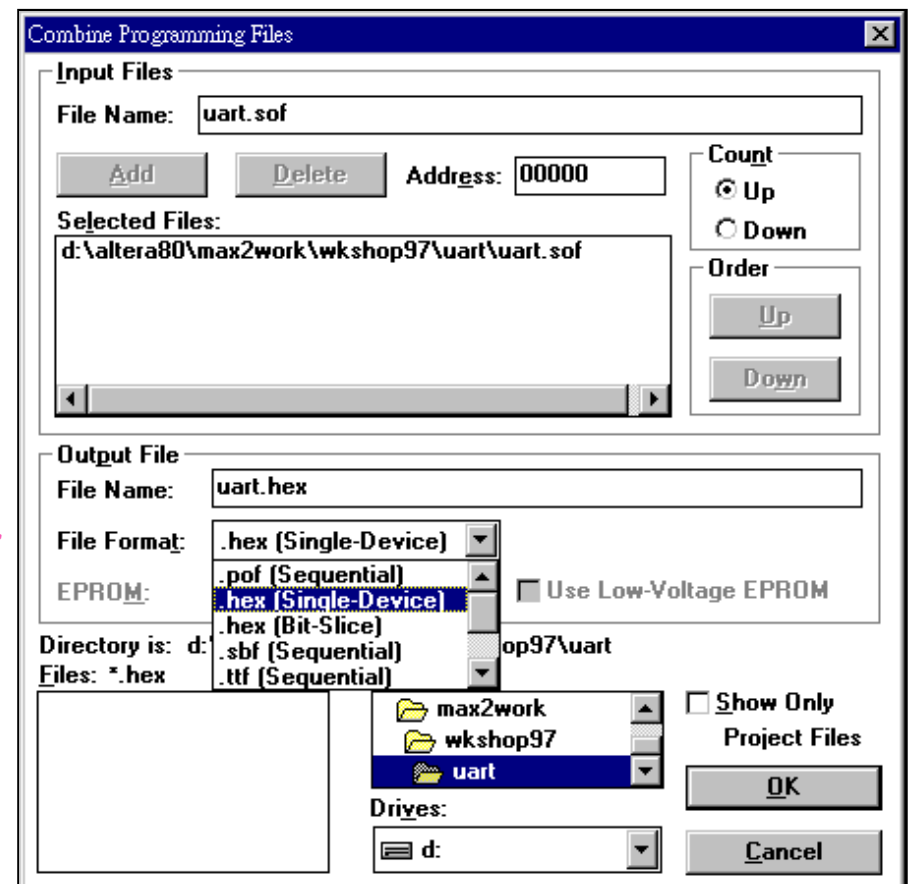


Converting or Combining Programming Files

◆ 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(*.tff)
 - Serial Bitstream File(*.sbf)
 - Hexadecimal (Intel-format) File(*.hex)

Menu: *File -> Combine Programming Files...*



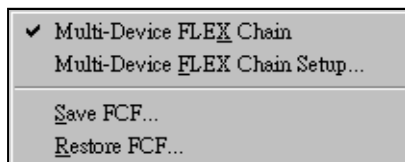
Configuring Multiple FLEX Devices

◆ 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

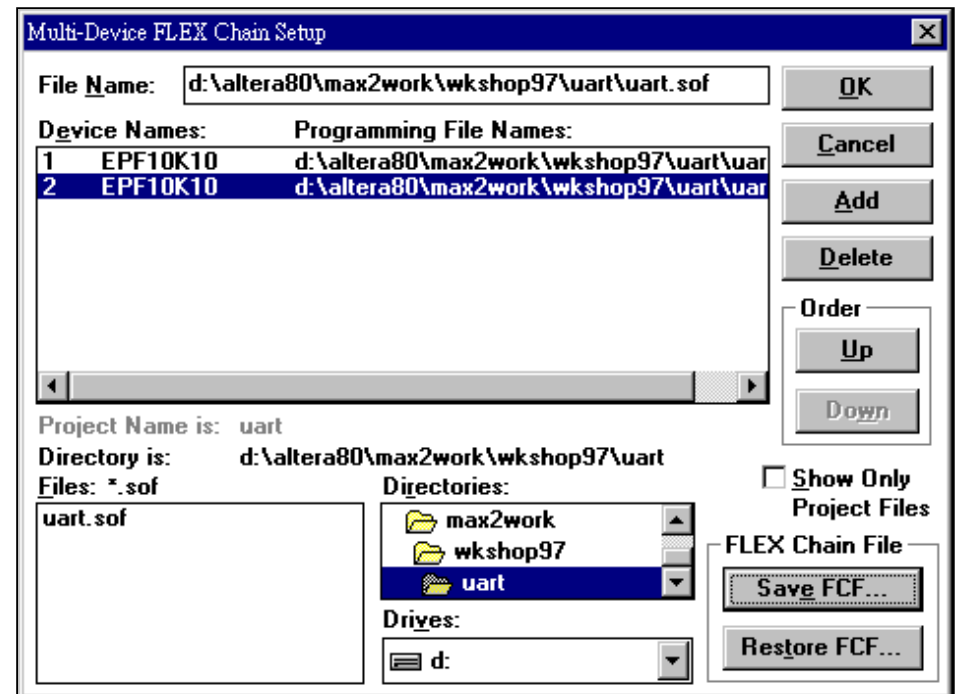
Menu: *FLEX -> Save FCF...*

Menu: *FLEX -> Restore FCF...*

- To turn on or off multi-device FLEX chain configuration mode

Menu: *FLEX -> Multi-Device FLEX Chain*

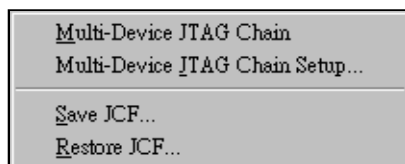
- Click Configure button on Programmer to start configuration



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
 - Multi-device JTAG chain: a series of devices through which programming and/or configuration data are passed from device to device via the JTAG boundary-scan test circuitry



JTAG Menu

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 existing JCF file

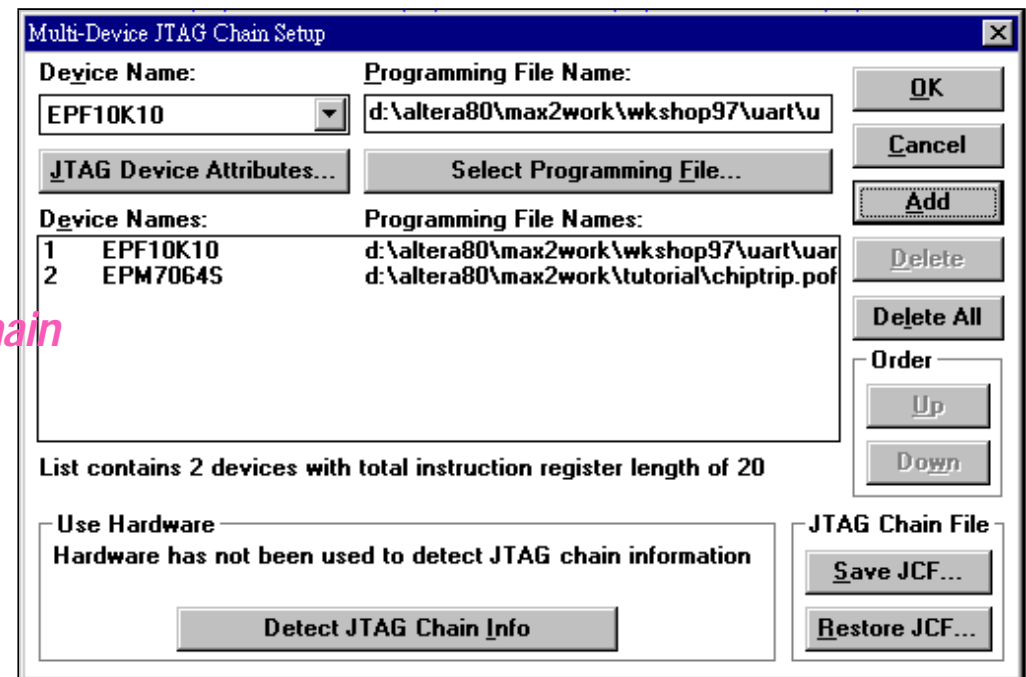
Menu: *JTAG -> Save JCF...*

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



Details about Device Programming

◆ 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*



Summary

◆ Why programmable logic?

- In-circuit verification
- Shorter time-to-market
- Reduced risk
- ...

◆ How to design?

- Various design entries: creates graphic, AHDL or waveform designs
- Design implementation: fits the design to the Altera device(s) and generates the programming file(s)
- Design verification: performs functional simulation, timing analysis, and timing simulation
- On-board device programming or configuration

Getting Help

◆ Get answers from on-line help or on-line document first

- Altera supports full on-line help in MAX+PLUS II

◆ CIC technical support: 林岡正

- Phone : (03)5773693 ext. 146
- Email : max@mbox.cic.edu.tw
- News : nsc.cic
- ftp-site : ftp://ftp.cic.edu.tw/pub (140.126.24.62) under [/pub/doc/manual/Altera](#)
- WWW : <http://www.cic.edu.tw/html/software/Altera>

◆ To buy Altera chips, hardware or demo boards:

- Contact Galaxy Far East Corp. 茂綸公司楊樂麗小姐 (02)7057266 ext. 213

◆ Altera technical support on Internet

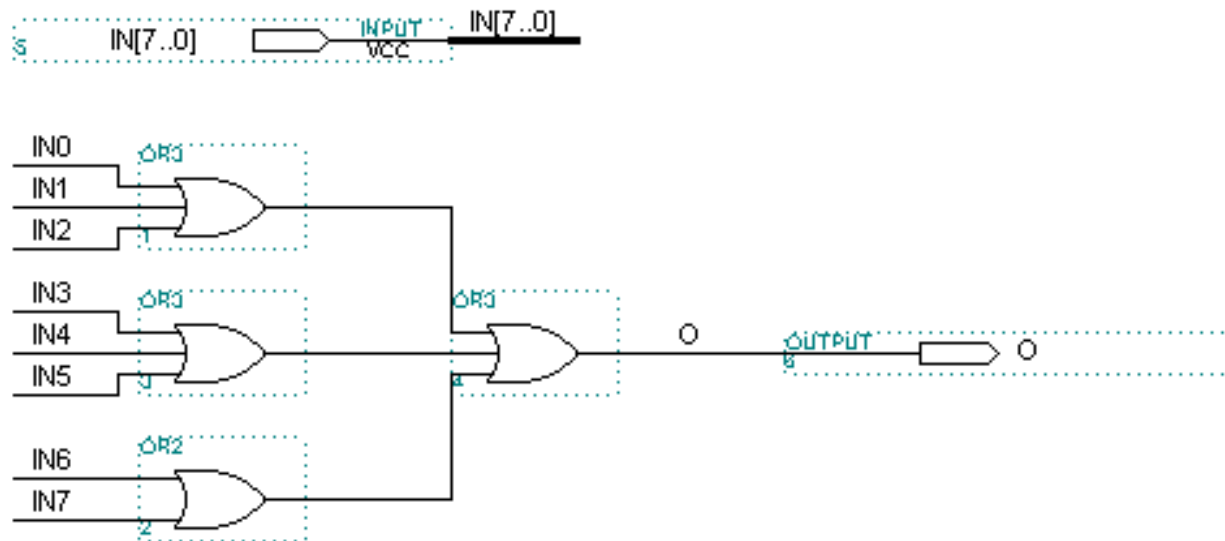
- WWW : <http://www.altera.com>
- FTP : <ftp://ftp.altera.com> (however, the international access may be slow)

MAX+plus II Lab

Fibonacci Generator

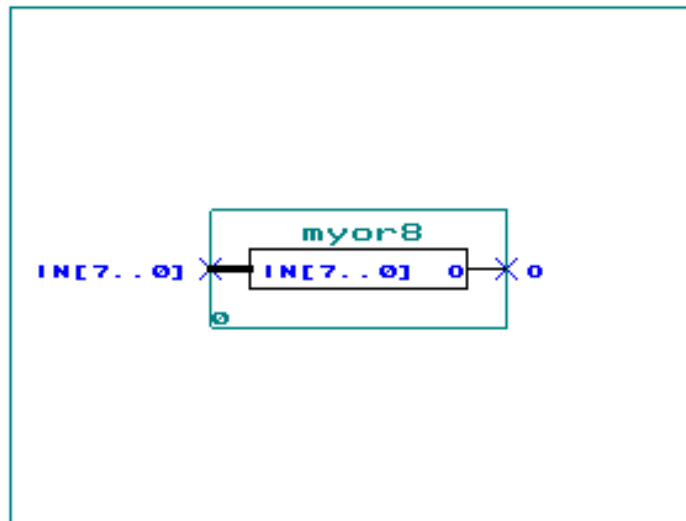
Lab 1 - myor8 I

- ◆ 用基本的邏輯閘兜電路
- ◆ Click Schematic Editor Icon
- ◆ File > New (Graphic Editor File - .gdf)
- ◆ 繪出下圖



Lab 1 - myor8 II

- ◆ **File > Project > Save and Check (myor8.gdf)**
- ◆ **Generate Symbol**
 - File > Create Default Symbol
- ◆ **View Symbol**
 - File > Open (myor8.sym)



Lab 2 - scan

- ◆ AHDL

- ◆ File > New

(Text Editor File - .tdf)

- ◆ File > Project > Save & Check

(scan.tdf)

- ◆ File > Create Default Symbol

```
TITLE "SEVEN SEGMENT LED ARRAY SCAN COUNT";
SUBDESIGN SCAN
(
  SCANCLK                      :INPUT;
  SCANCODE[5..0]               :OUTPUT;
  SCANSEL[2..0]                :OUTPUT;
)
VARIABLE
  SCANSEL[2..0]                :DFF;

BEGIN
  SCANSEL[ ].CLK = SCANCLK;
  IF SCANSEL[ ] == 5 THEN
    SCANSEL[ ].D = 0;
  ELSE
    SCANSEL[ ].D = SCANSEL[ ].Q + 1;
  END IF;
  TABLE
    SCANSEL[ ]                =>  SCANCODE[ ];
                                0      =>  B"000001";
                                1      =>  B"000010";
                                2      =>  B"000100";
                                3      =>  B"001000";
                                4      =>  B"010000";
                                5      =>  B"100000";
                                6      =>  B"000000";
                                7      =>  B"000000";
  END TABLE;
END;
```

Lab 3 - switch

- ◆ AHDL

- ◆ File > New

(Text Editor File - .tdf)

- ◆ File > Project > Save & Check

(switch.tdf)

- ◆ File > Create Default Symbol

```
SUBDESIGN SWITCH
( SEL[2..0] : INPUT;
  A[7..0], B[7..0], C[7..0] : INPUT;
  Y[3..0] : OUTPUT;)

BEGIN

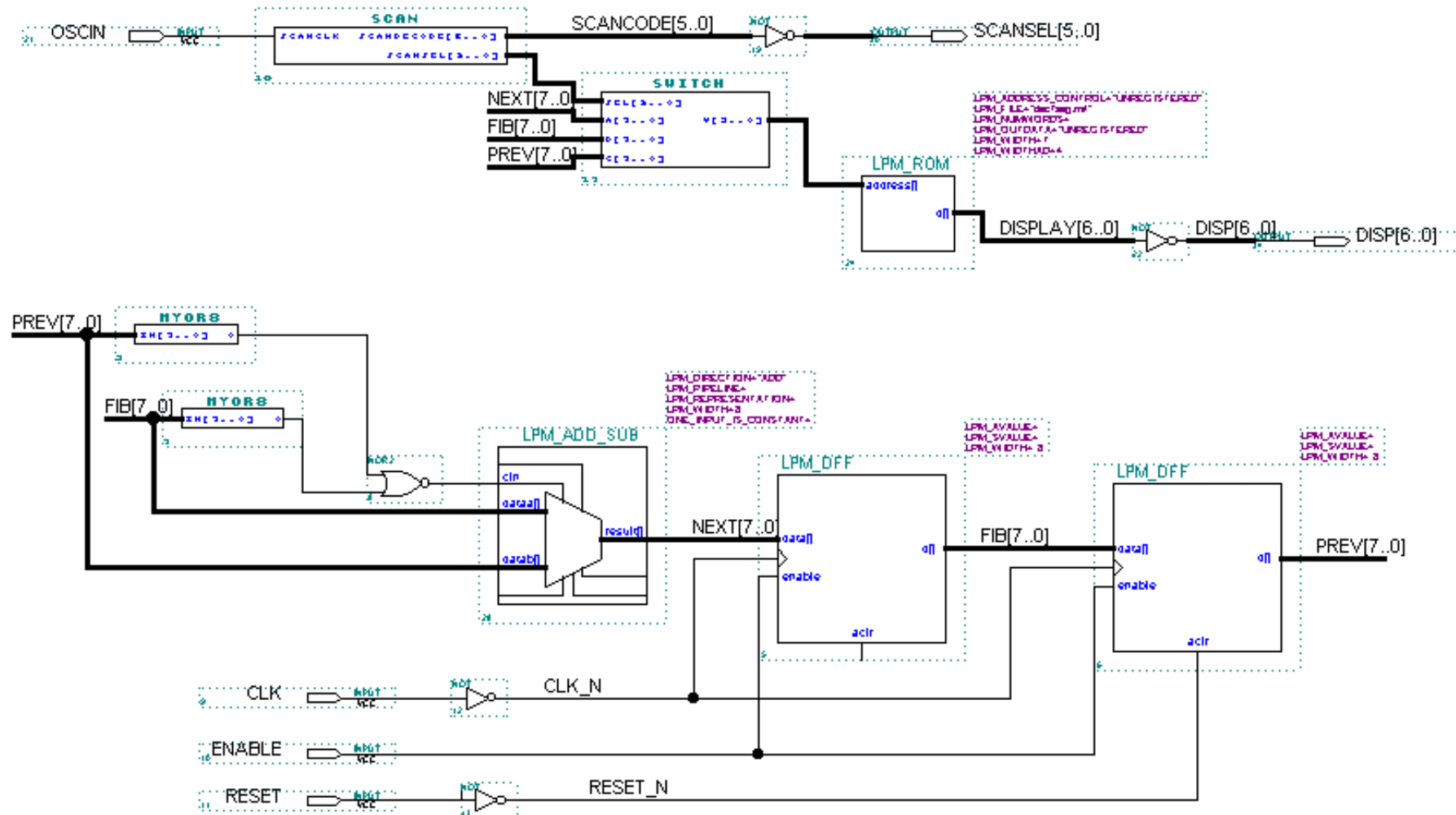
CASE SEL[] IS
  WHEN 0 =>
    Y[] = A[7..4];
  WHEN 1 =>
    Y[] = A[3..0];
  WHEN 2 =>
    Y[] = B[7..4];
  WHEN 3 =>
    Y[] = B[3..0];
  WHEN 4 =>
    Y[] = C[7..4];
  WHEN 5 =>
    Y[] = C[3..0];
END CASE;

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)

Lab 4 - fib_top II



Lab 4 - fib_top III

◆ 利用 LPM_ROM 去產生七段顯示器的解碼電路

- LPM_FILE - dec7seg.mif

```
WIDTH = 7;  
DEPTH = 16;
```

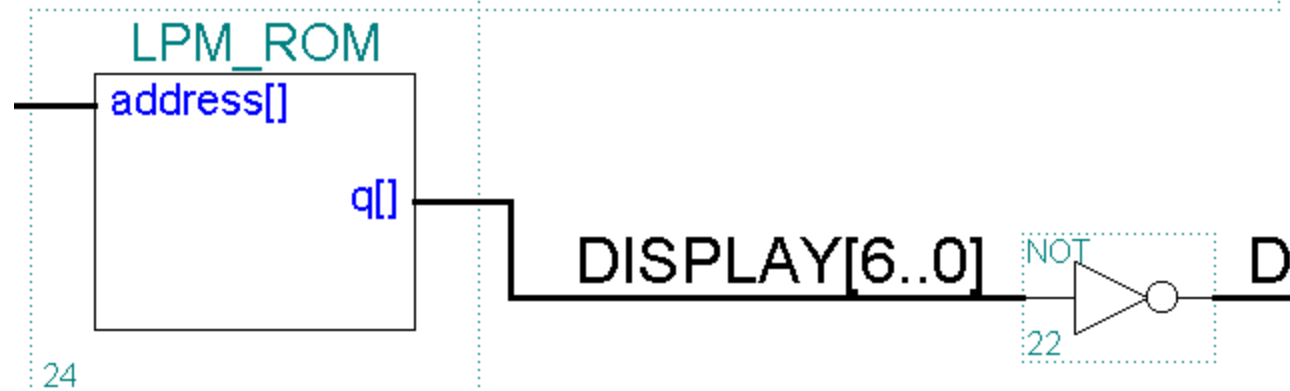
```
ADDRESS_RADIX = HEX;  
DATA_RADIX = BIN;
```

```
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;
```

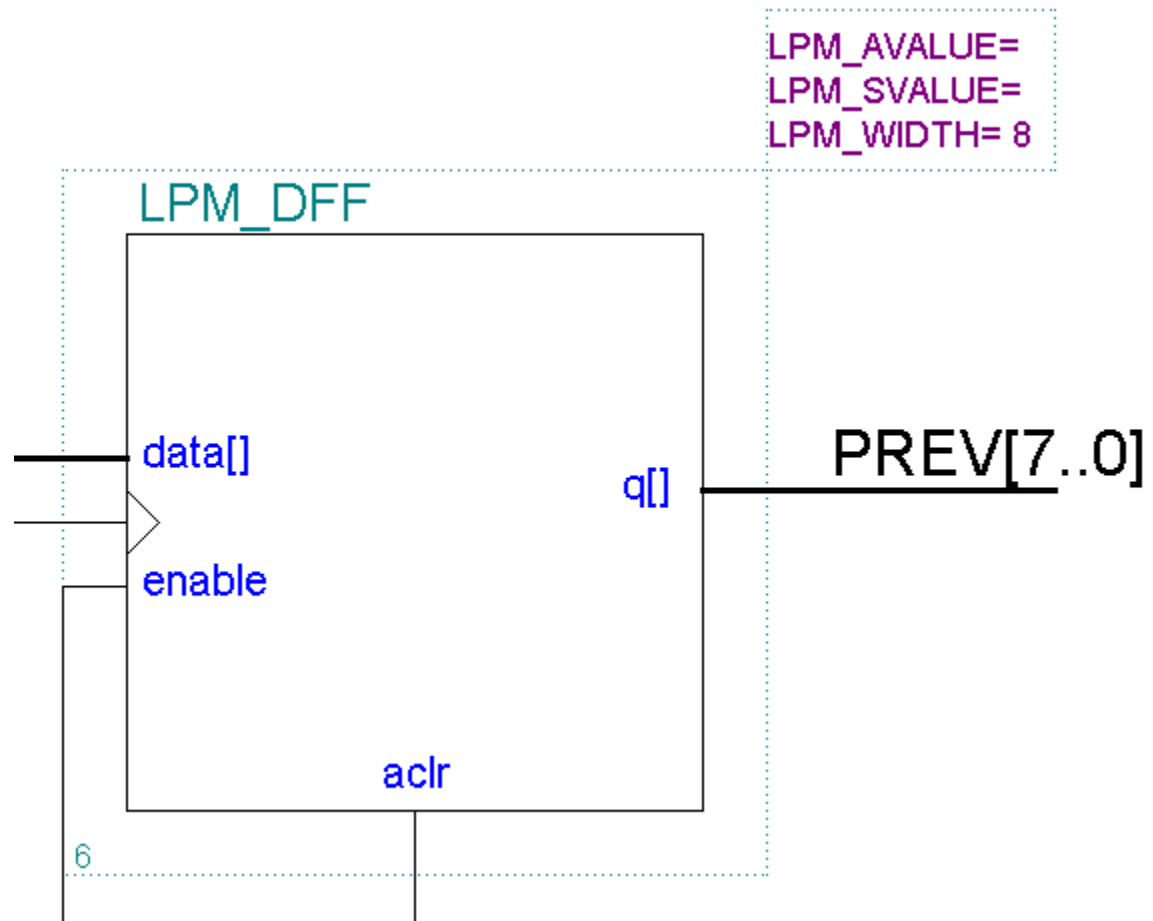
```
END;
```

```
LPM_ADDRESS_CONTROL="UNREGISTERED"  
LPM_FILE="dec7seg.mif"  
LPM_NUMWORDS=  
LPM_OUTDATA="UNREGISTERED"  
LPM_WIDTH=7  
LPM_WIDTHAD=4
```



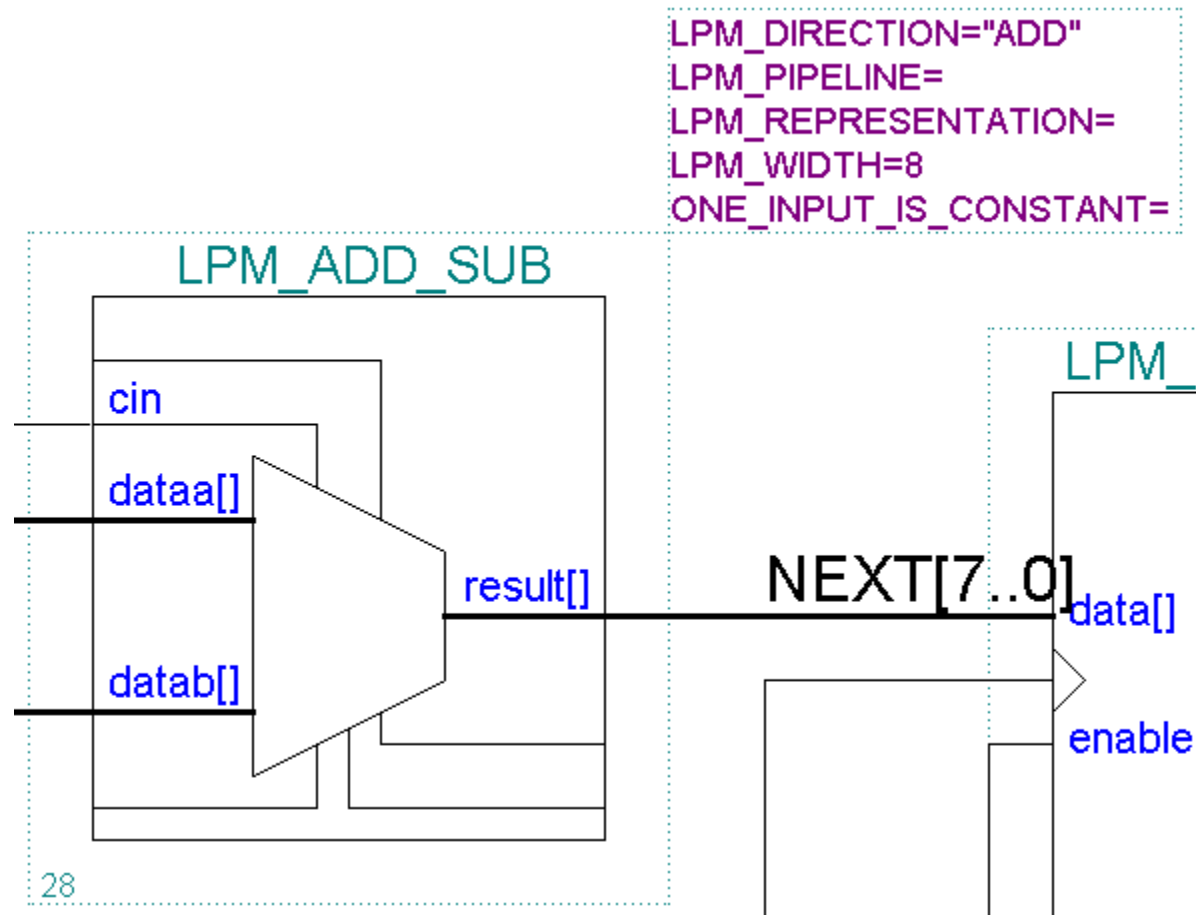
Lab 4 - fib_top IV

◆ LPM_REG



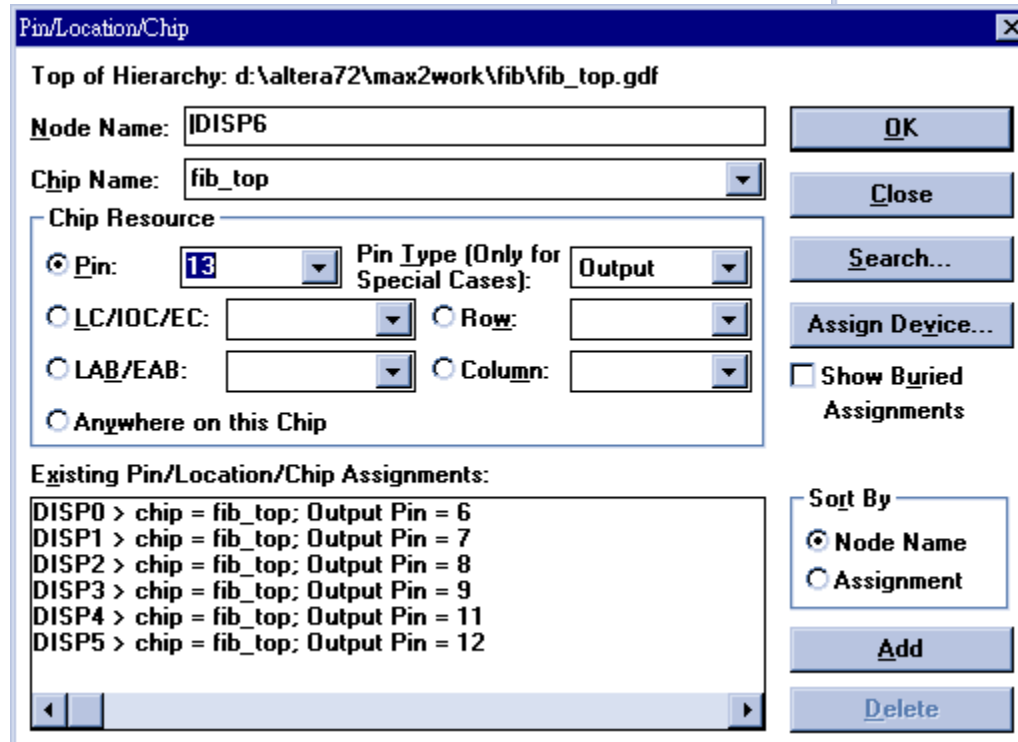
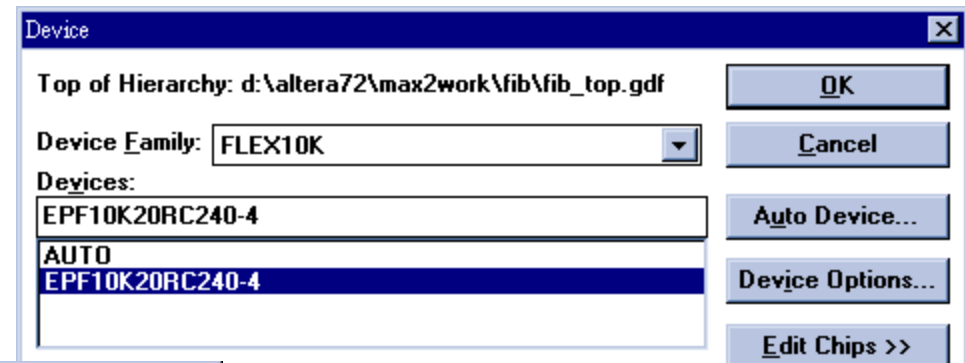
Lab 4 - fib_top V

◆ LPM_ADD_SUB



Lab 5 - Compile I

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

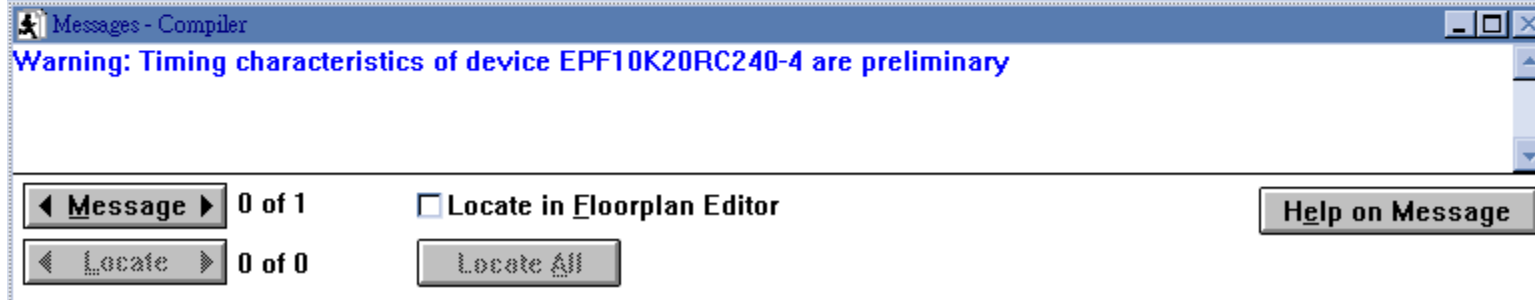
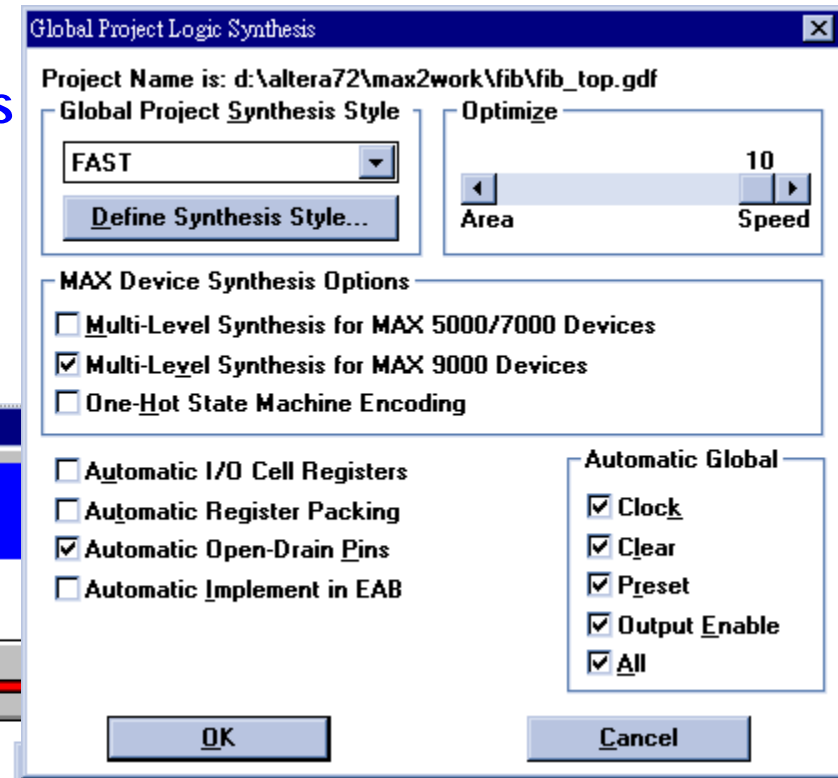
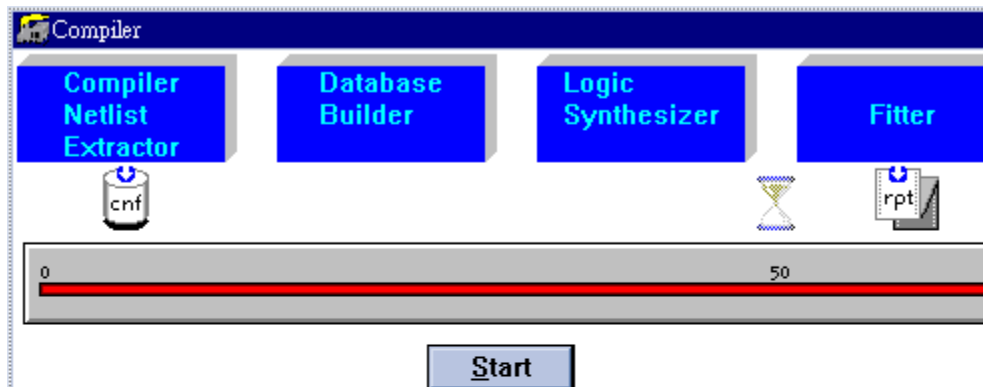


Lab 5 - Compile II

◆ Assign > Global Project Logic Synthesis

- Global Project Synthesis Style - FAST
- Optimize - 10 (Speed)

◆ Press "Start"



Lab 6 - Check Report File

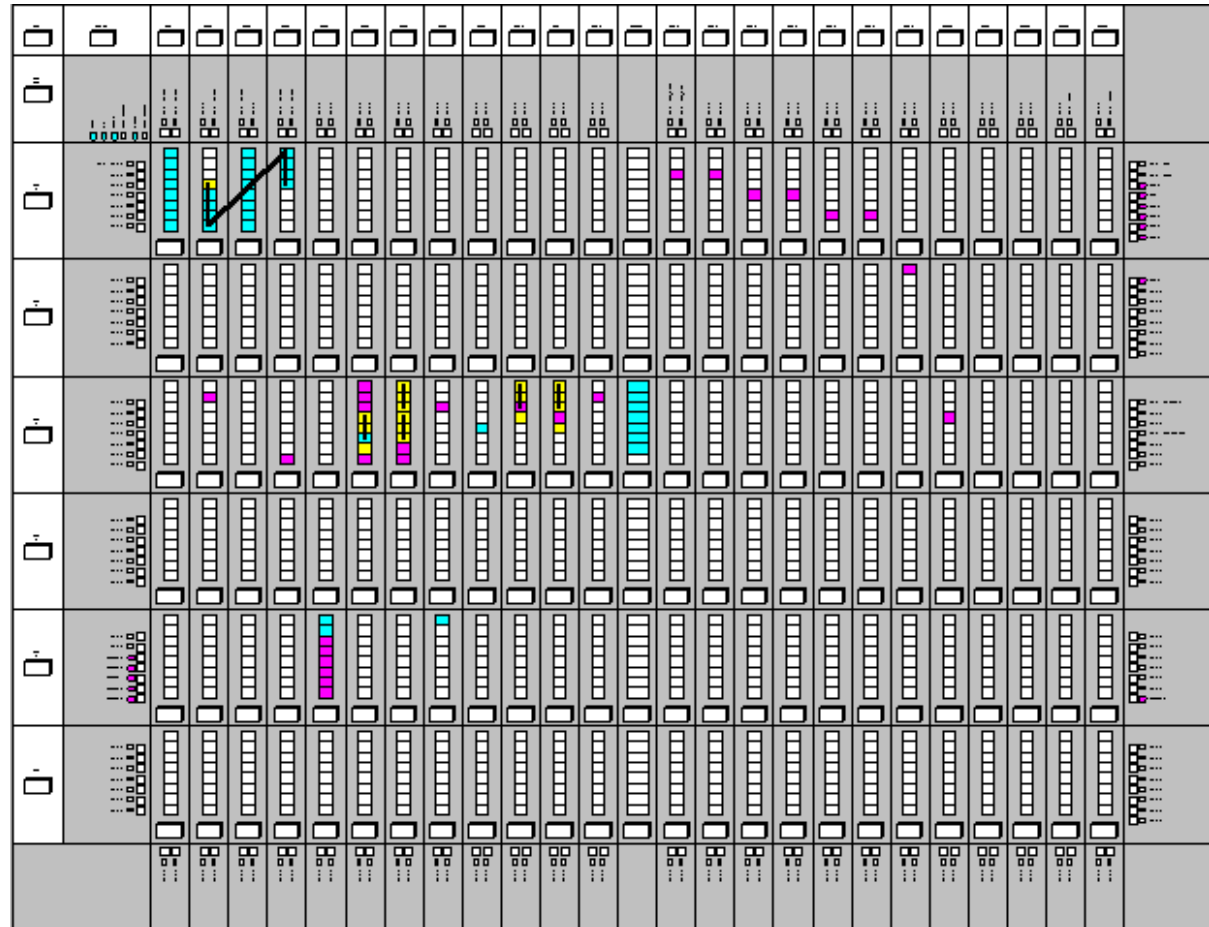
◆ Double click the Report File icon

◆ 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



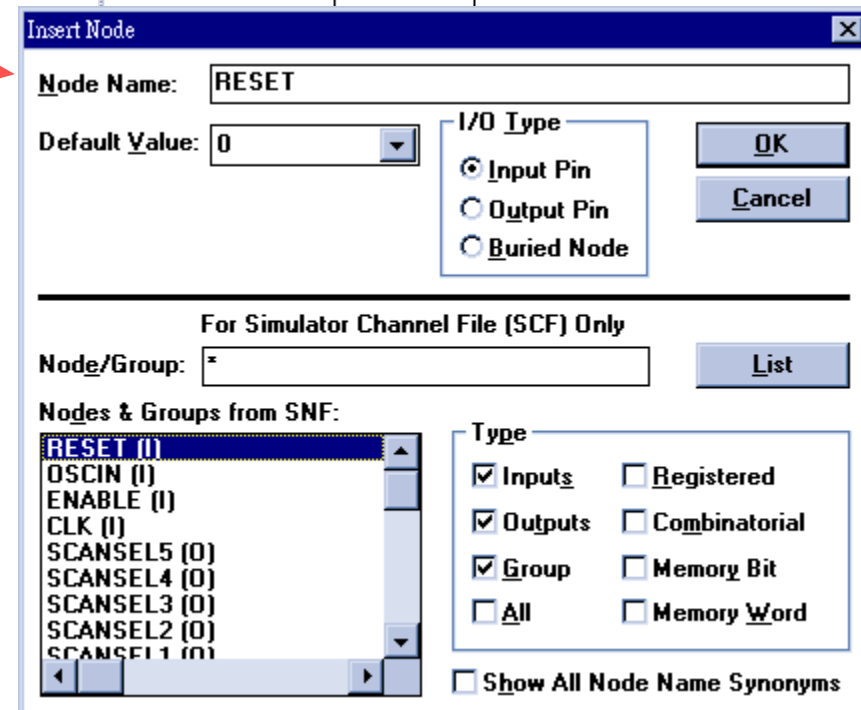
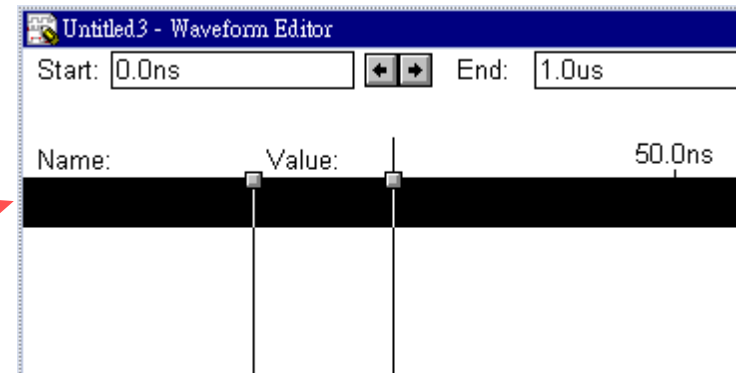
Lab 8 - Timing Simulation I

◆ Create a SCF file

- File > New (Waveform Editor File - .scf)

◆ Insert nodes in SCF

- Double click on "name" →
- Fill Node Name →
 - CLK, RESET, ENABLE, PREV[7..0], FIB[7..0], NEXT[7..0], OSCIN, CANSEL[5..0], DISP[6..0]



Lab 8 - Timing Simulation II

◆ Change Grid Size

- Options > Grid Size (10.0ns)

◆ Set End Time

- File > End Time (10us)

◆ Draw Waveforms

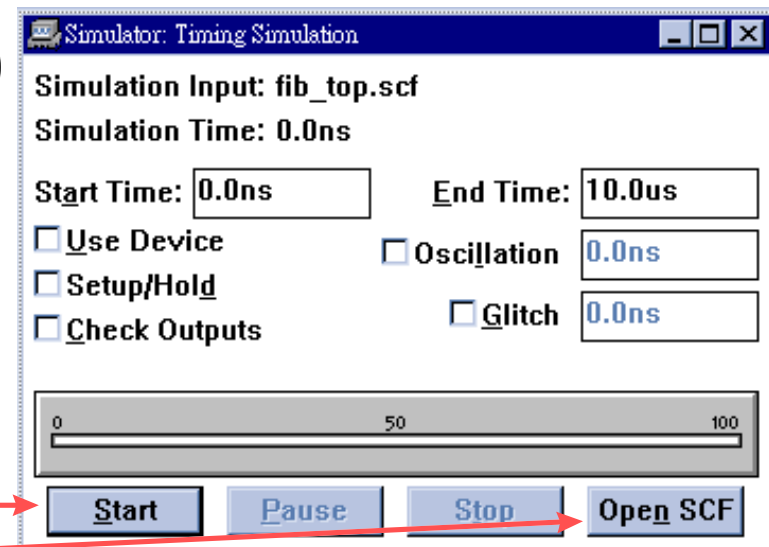
- OSCIN (clock, period 20ns * 1)
- CLK (clock, period 20ns * 12)
- RESET (0ns 0, 240ns 1, 3.76us 0, 4.49ns 1)
- ENABLE (0ns 1, 2.11us 0, 3.06us 1)

◆ Save SCF File

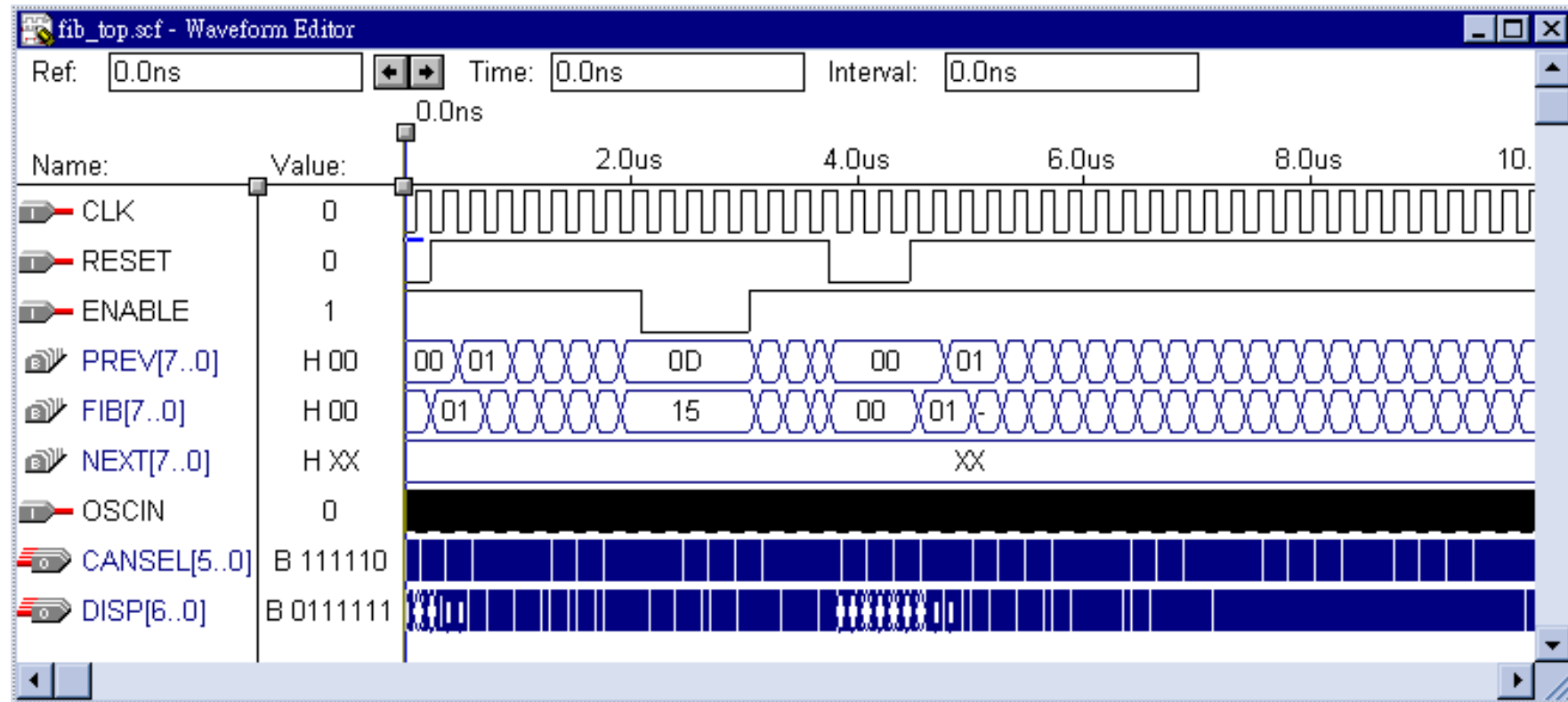
- File > Save as (fib_top.scf)

◆ Run Simulator

- MAX+plus II > Simulator
- Press "Start"
- Press "Open SCF"

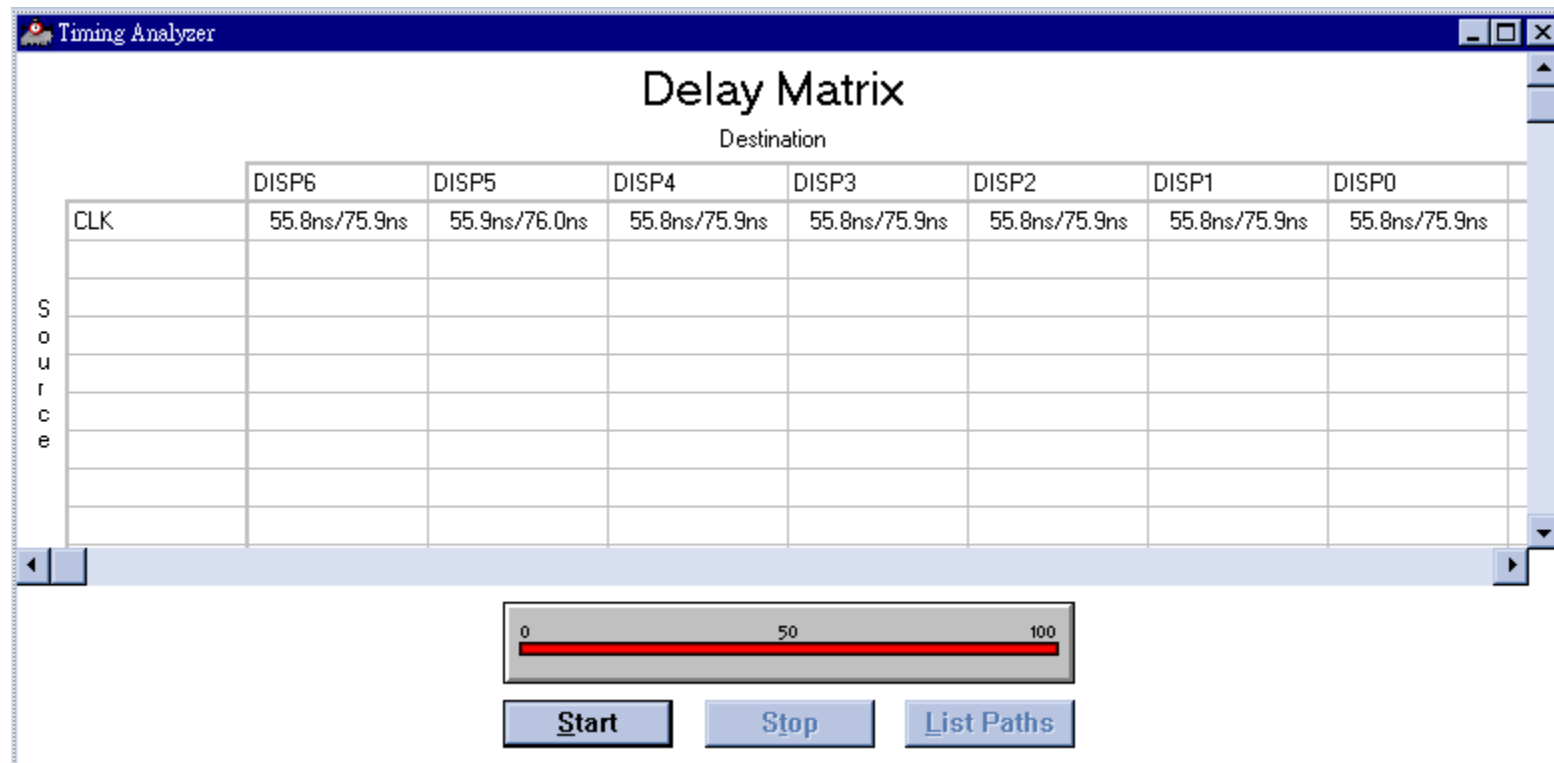


Lab 8 - Timing Simulation III



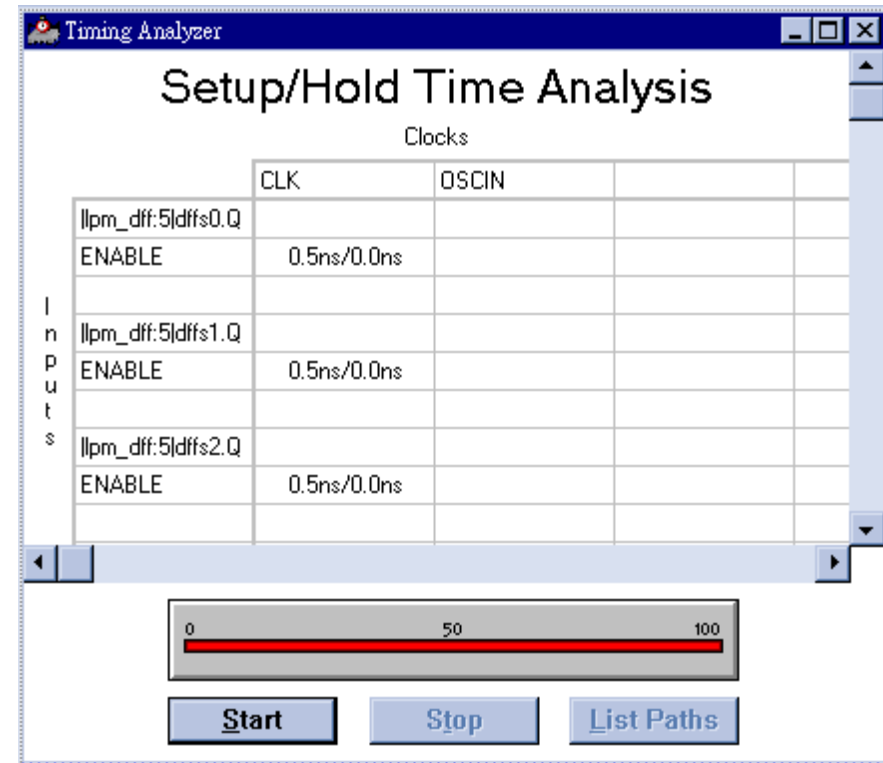
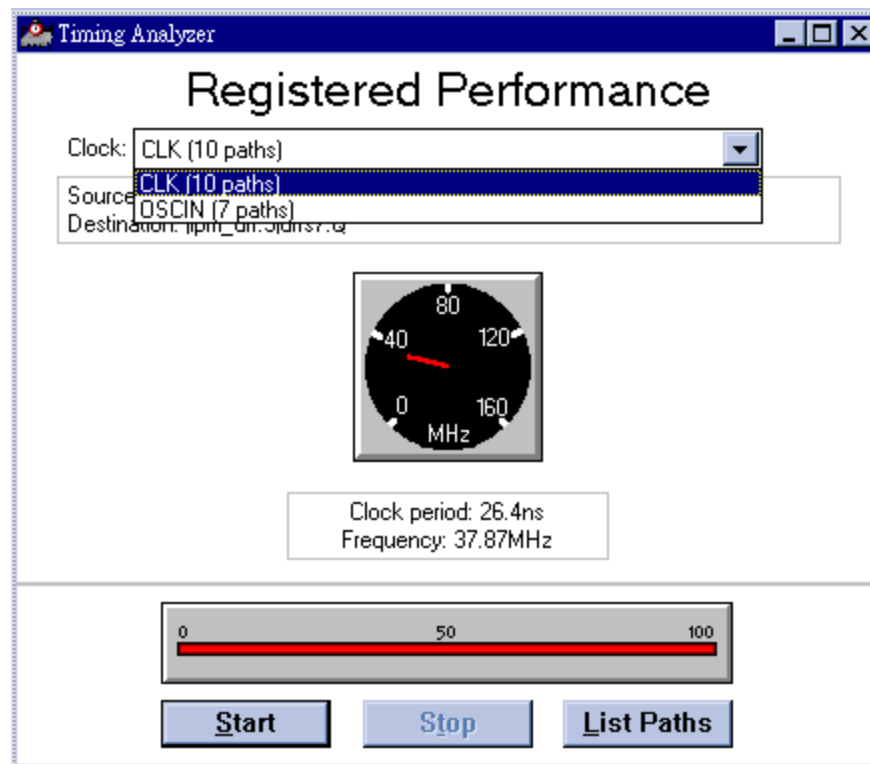
Lab 9 - Timing Analysis I

- ◆ 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"





Thank You!