

# Altera PC

報告人：國科會晶片系統設計中心 周育德 工程師

電話：(03)5773693 ext. 148

傳真：(03)5783372

Email：steven@cic.edu.tw



July 2001



# Course Outline - 1

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



# Course Outline - 2

## ◆ Design Implementation

## ◆ Project Verification

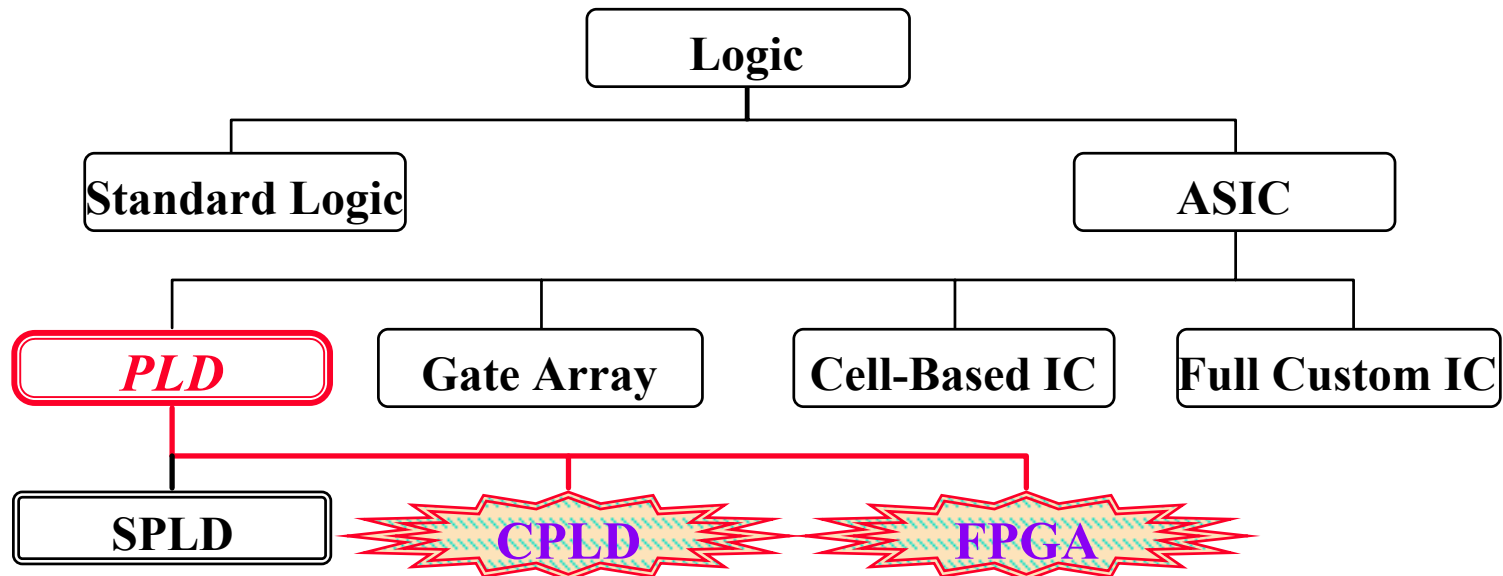
- Functional Simulation
- Timing Analysis
- Timing Simulation

## ◆ Device Programming

## ◆ Summary & Getting Help



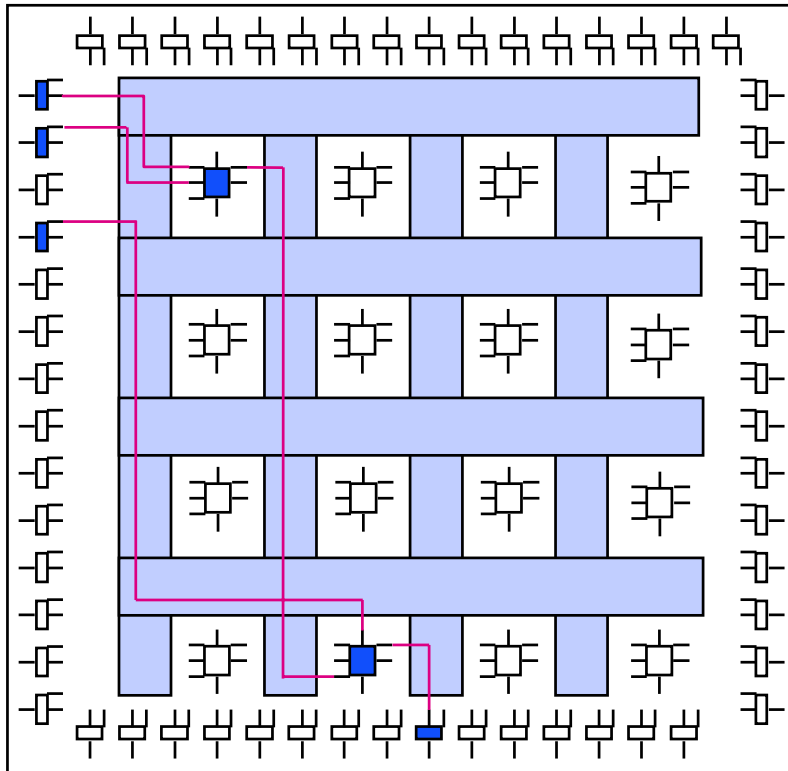
# Introduction to PLD



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



# Main Features



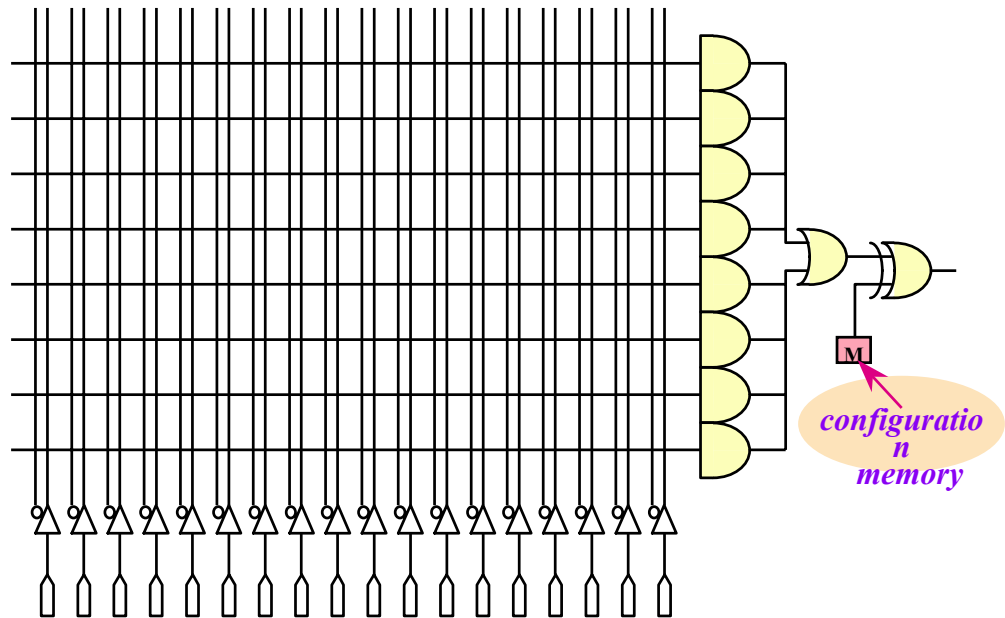
- ◆ Field-programmable
- ◆ Reprogrammable
- ◆ In-circuit design verification
- ◆ Rapid prototyping
- ◆ Fast time-to-market
- ◆ No IC-test & NRE cost
- ◆ H/W emulation instead of S/W simulation
- ◆ Good software
- ◆ ...



# Programmability

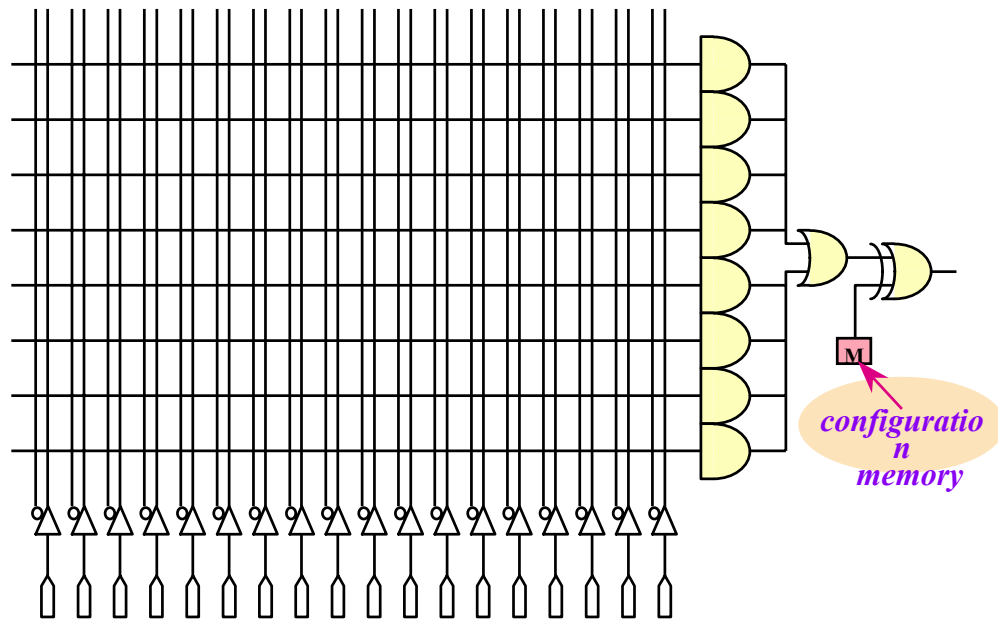
## ◆ Why programmable? Why reprogrammable?

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



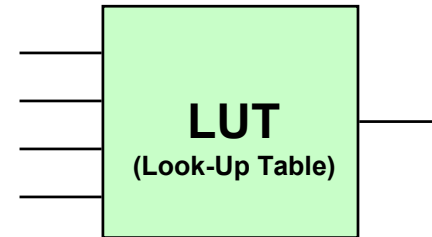


# Programmable Combinational Logic



Product Term-based Building Block

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

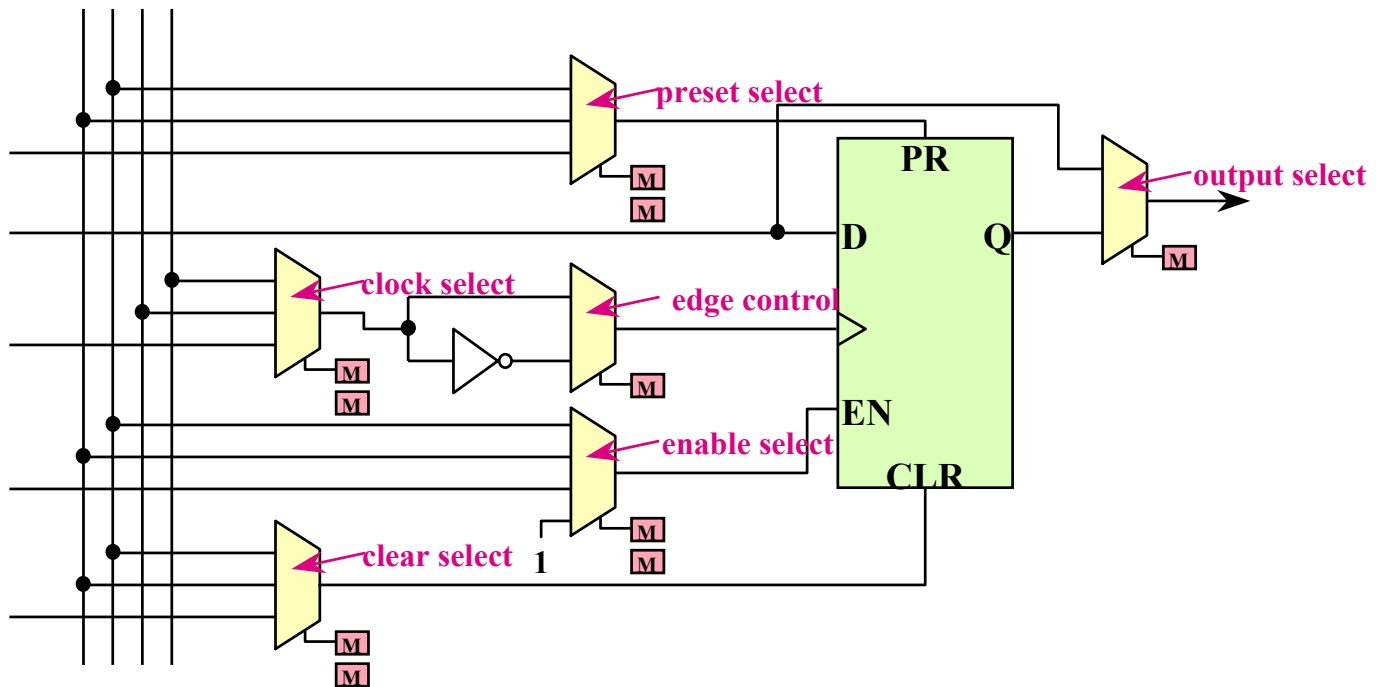


Look-up Table-based Building Block

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



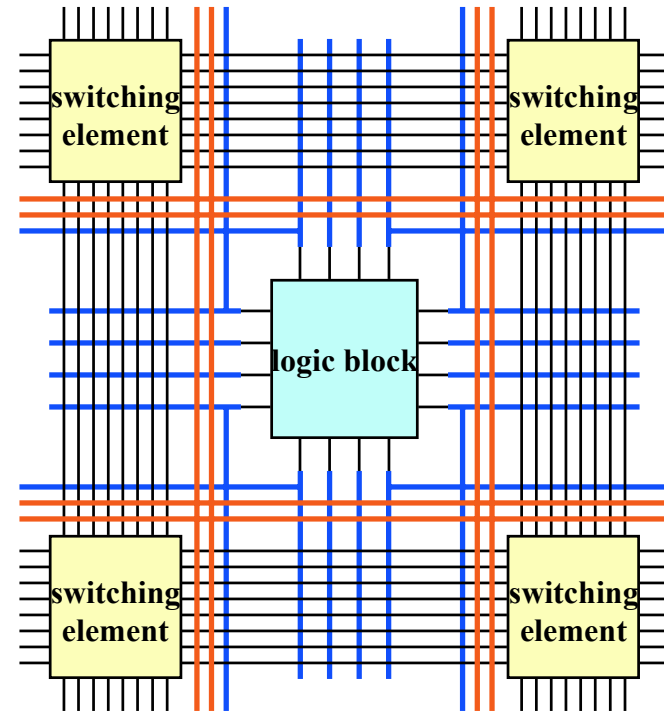
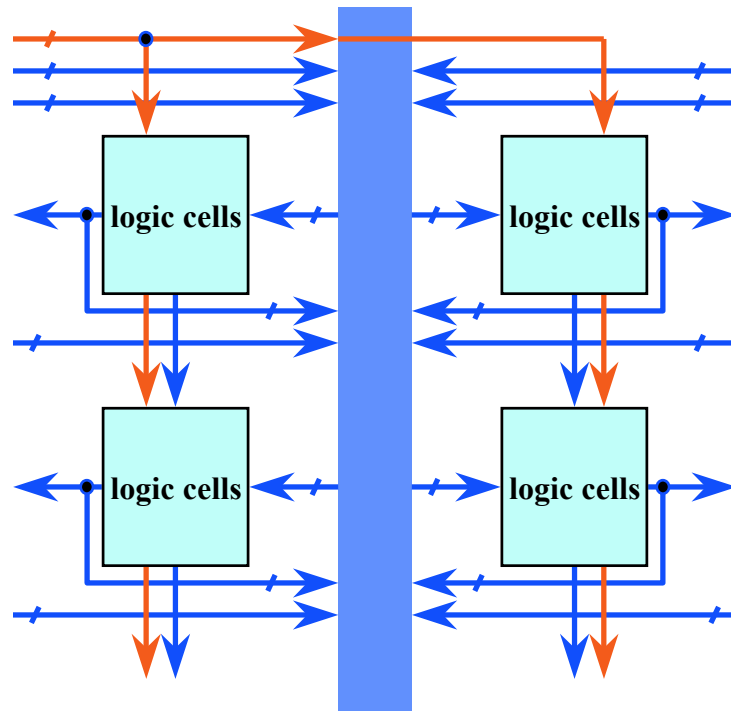
# Programmable Register



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



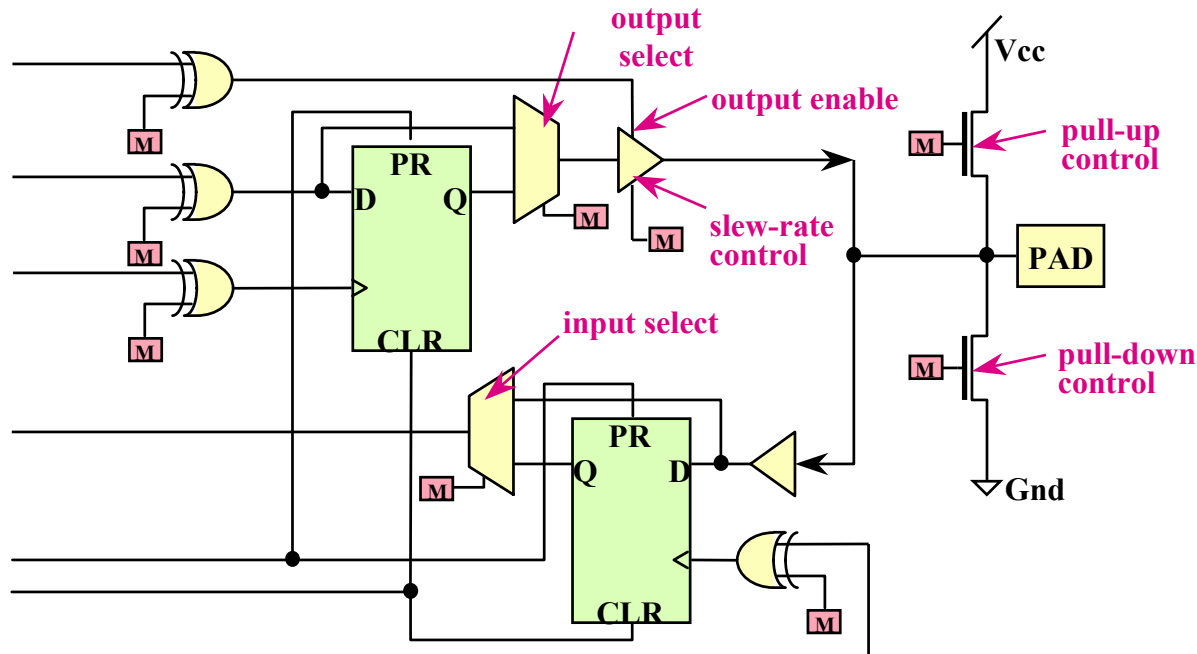
# Programmable Interconnect



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



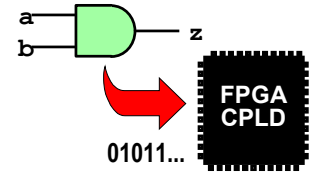
# Programmable I/O



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

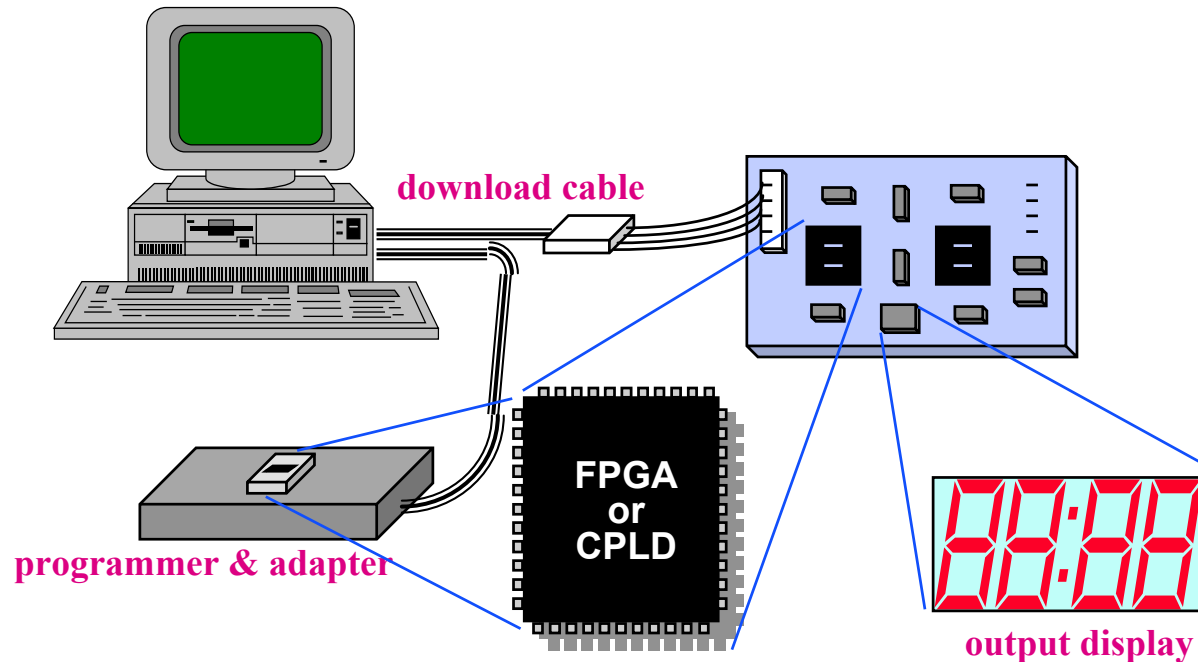


# Field-Programmability



## ◆ 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



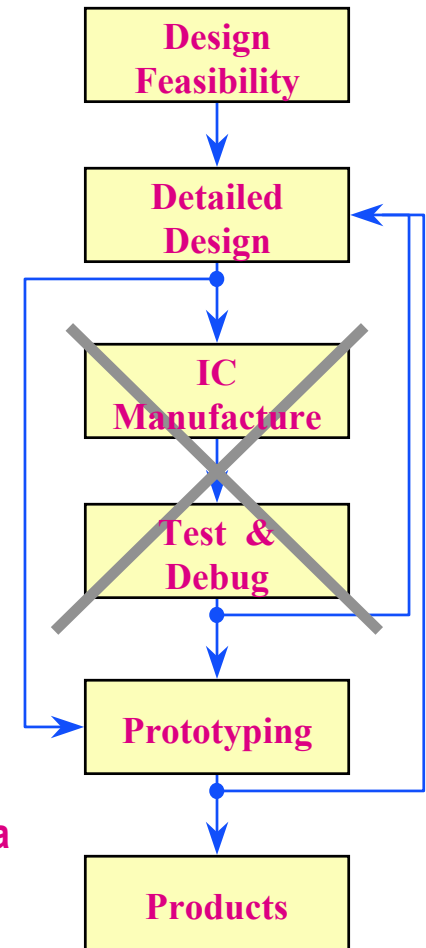
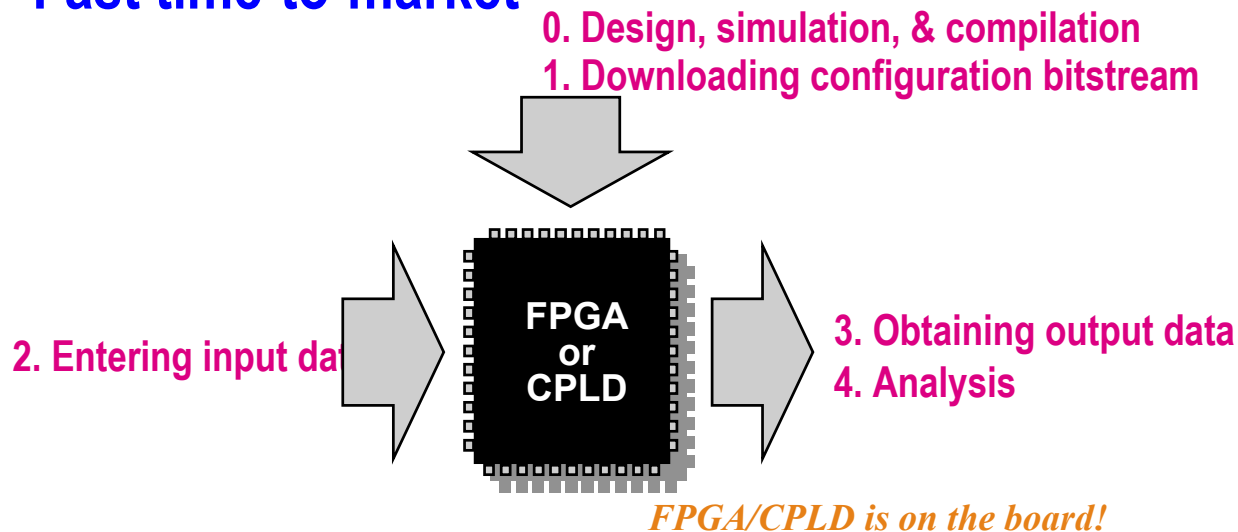


# Rapid Prototyping

## ◆ 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

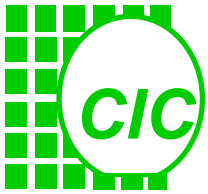


# FPGA/CPLD Benefits

	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

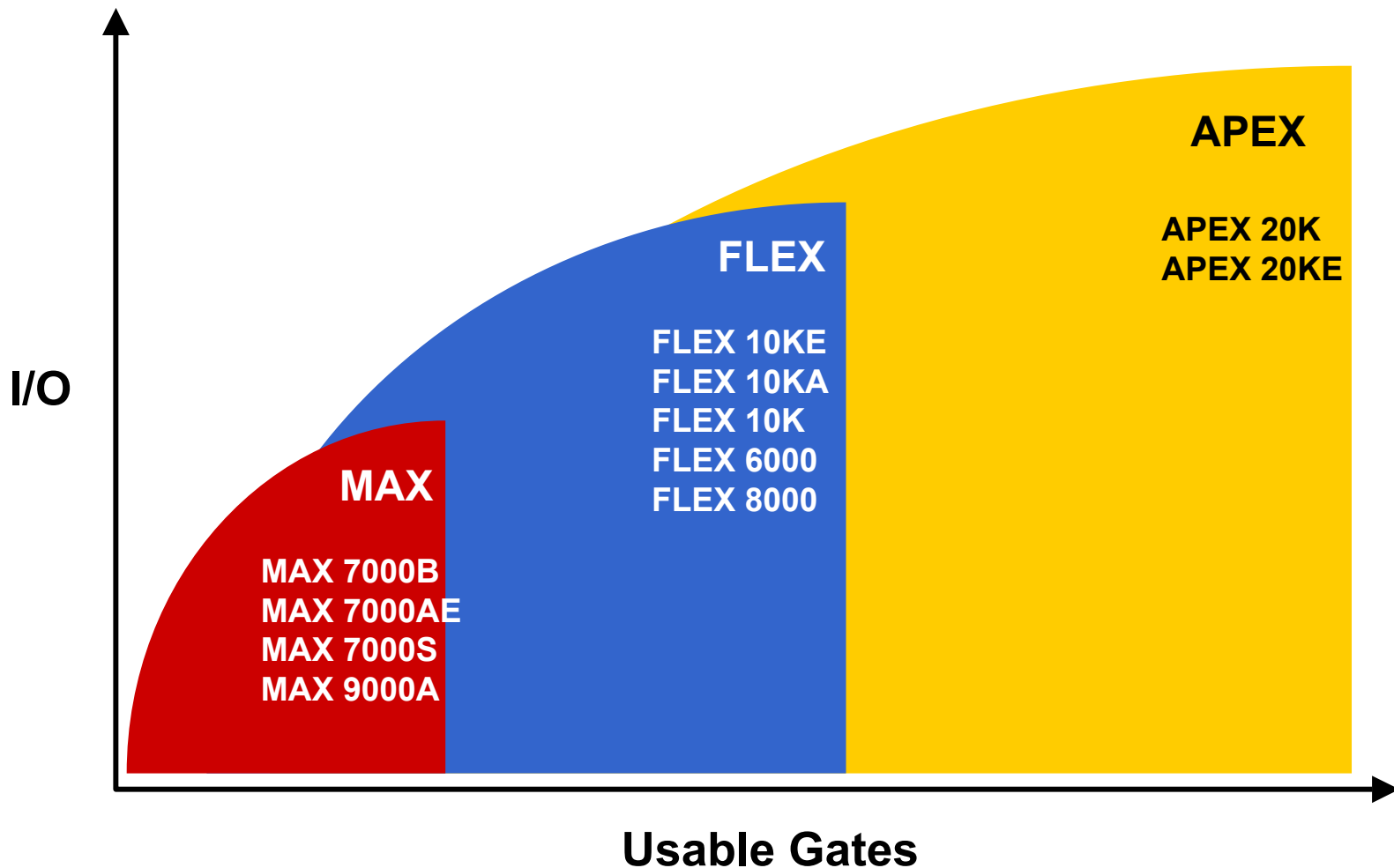
- One of the world leaders in high-performance & high-density PLDs & associated CAE tools
- Supports university program in Taiwan via CIC

## ◆ 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 -
- University Program Design Laboratory Package (since 9709):
  - UP1 Education Board
  - ByteBlaster download cable
  - Student Edition Software
- Of course, CIC is responsible for technical supports
- WWW: [http://www.cic.edu.tw/chip\\_design/design\\_intr/altera/](http://www.cic.edu.tw/chip_design/design_intr/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 <sup>(1)</sup>	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 <sup>(1)</sup>	EEPROM	SOP	6,000 ~ 12,000	EPM9320/A, 400/A, 480/A, 560/A
FLEX 10K/A/B <sup>(1)</sup>	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.



# Device Part Numbers

## ◆ EPM7128STC100-7

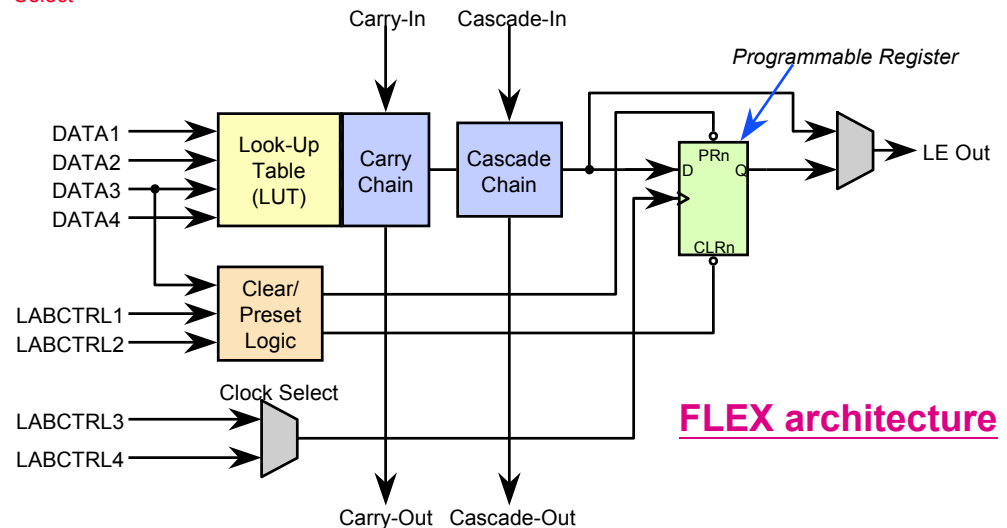
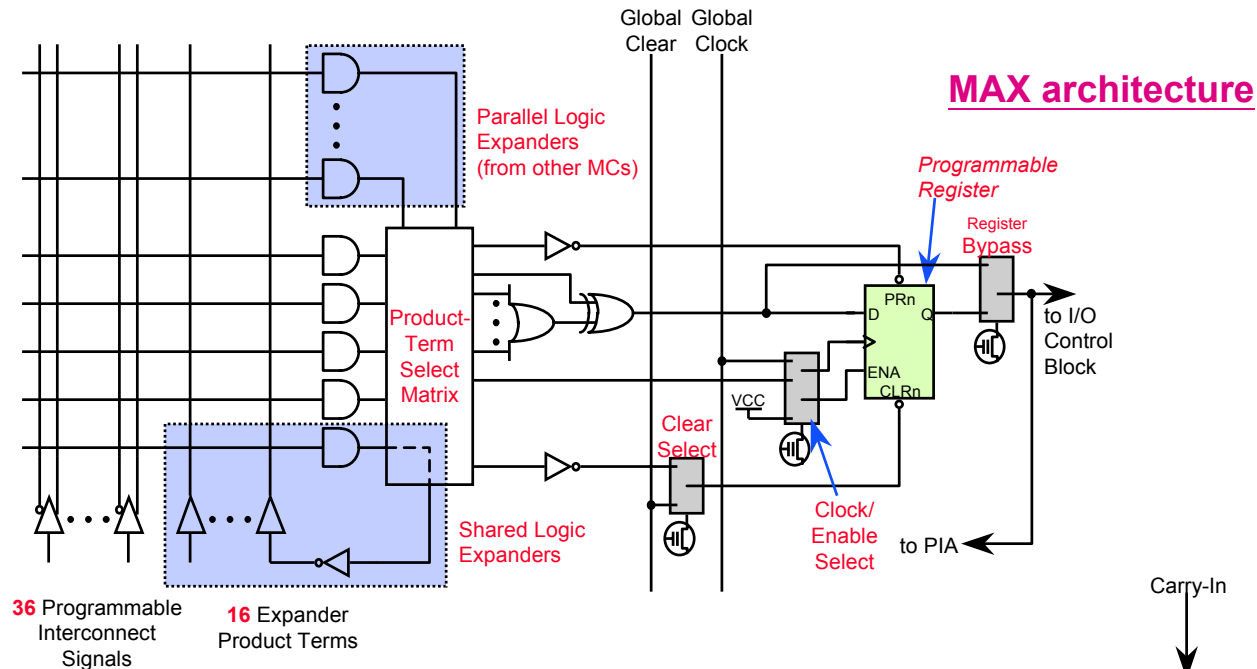
- EPM = Family Signature (**E**rasable **P**rogrammable **MAX** device)
- 7128S = Device type (128 = number of macrocells)
- T = Package type (L = PLCC, T = TQFP...)
- C = Operating temperature (Commercial, Industrial)
- 100 = Pin count (number of pins on the package)
- -7 = Speed Grade in nsec
- Suffix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

## ◆ Another Example:

- EPM7064SLC44-5
  - EPM7064S in a commercial-temp, 44 pin PLCC package with a 5 ns speed grade



# MAX & FLEX Architectures - (1)





# MAX & FLEX Architectures - (2)

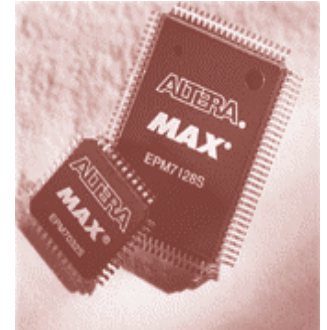
## ◆ Choose the appropriate architecture

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

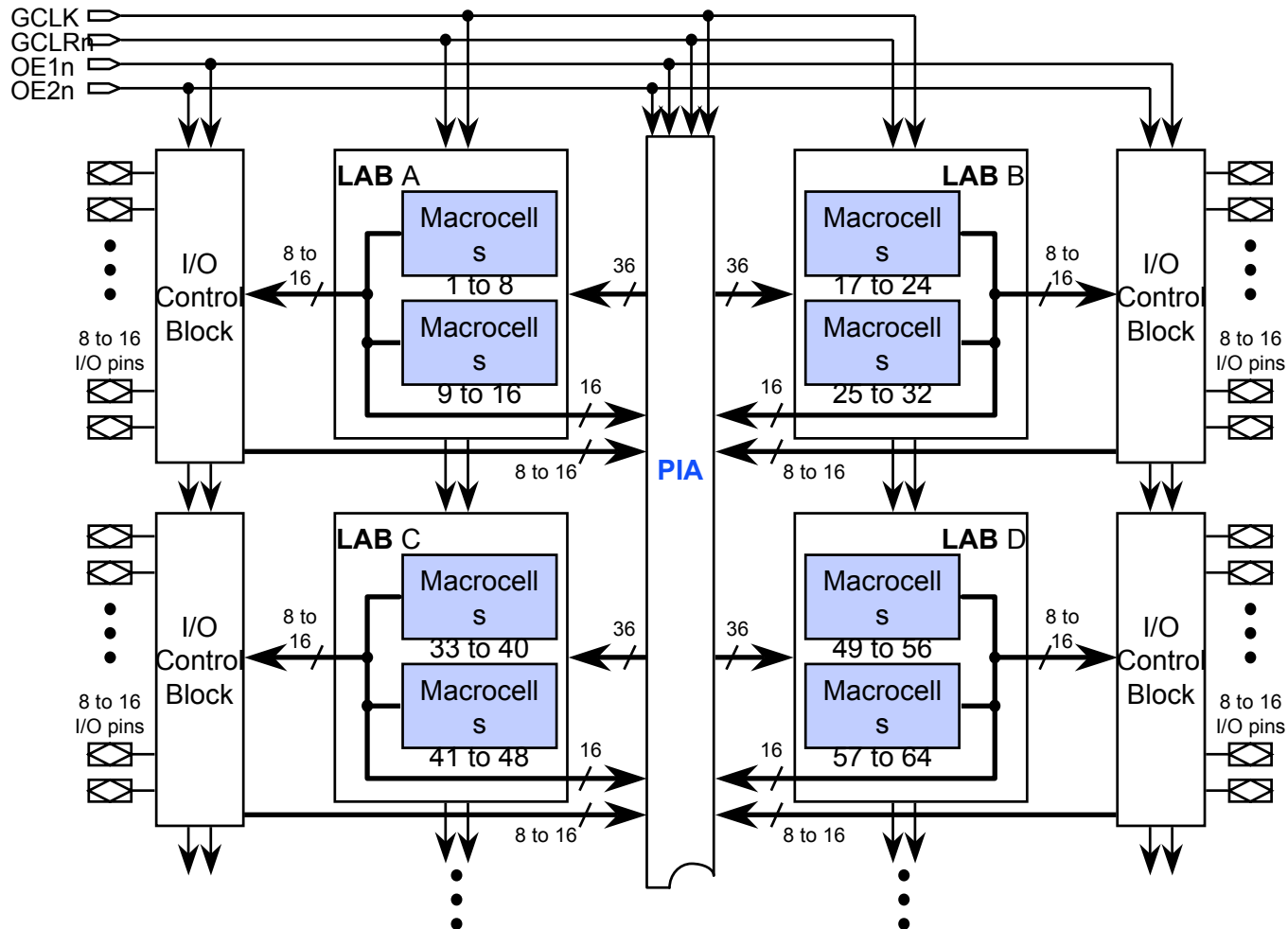
- More output enable control signals & more global clocking capabilities
- Fast input registers
- Programmable output **slew-rate control**
- More interconnect resources

## ◆ 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

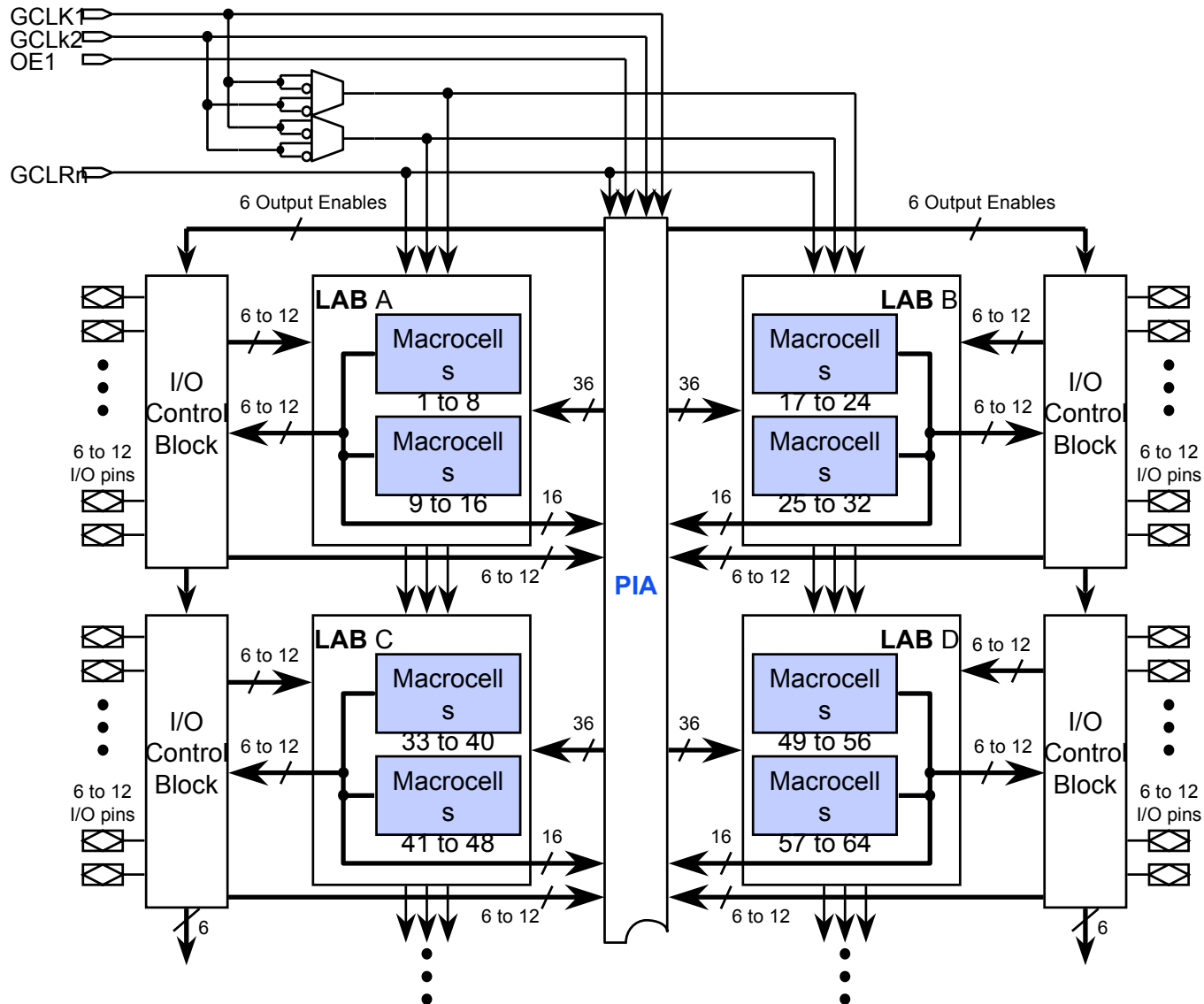


# MAX 7000 Architecture



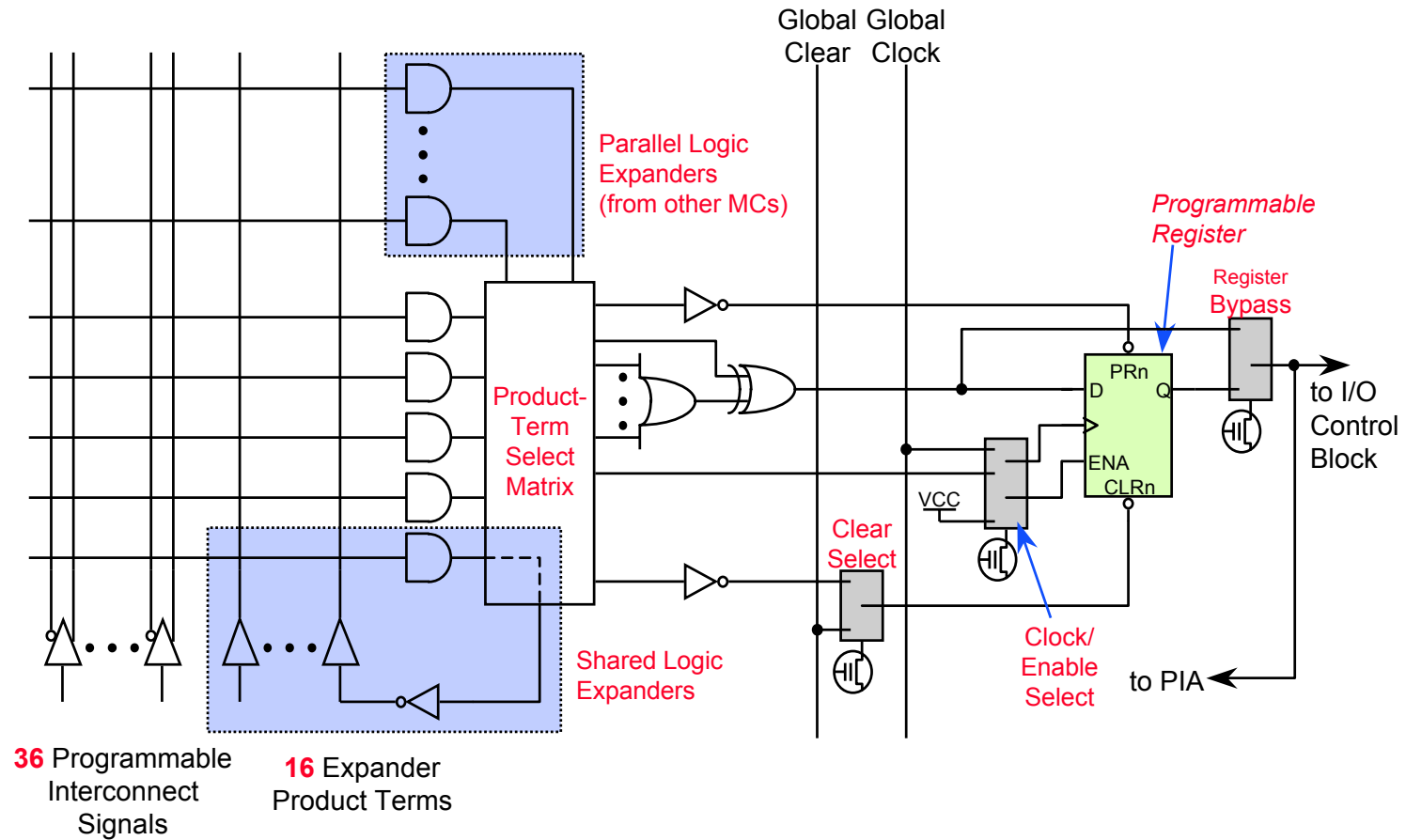


# MAX 7000E/S Architecture



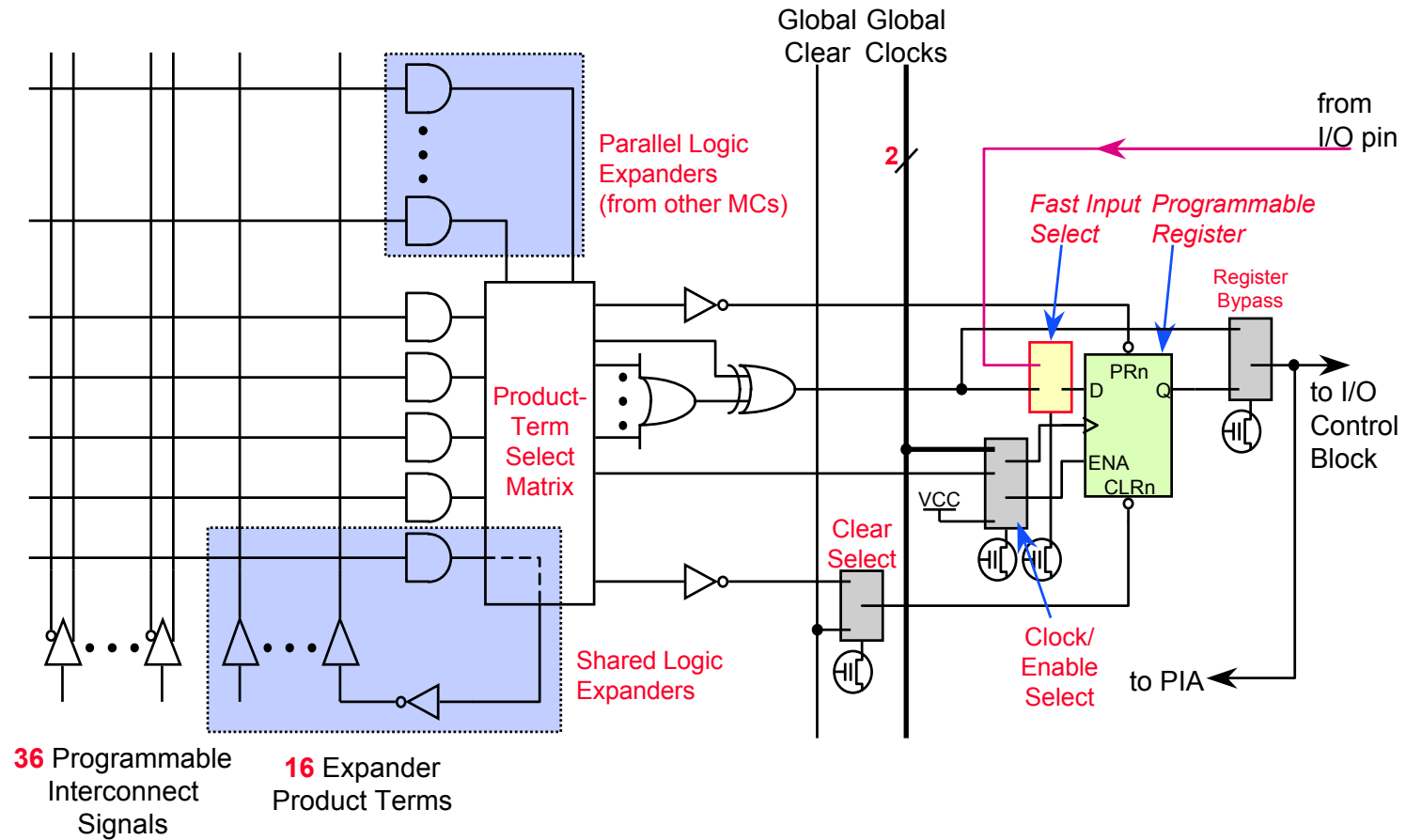


# MAX 7000 Macrocell



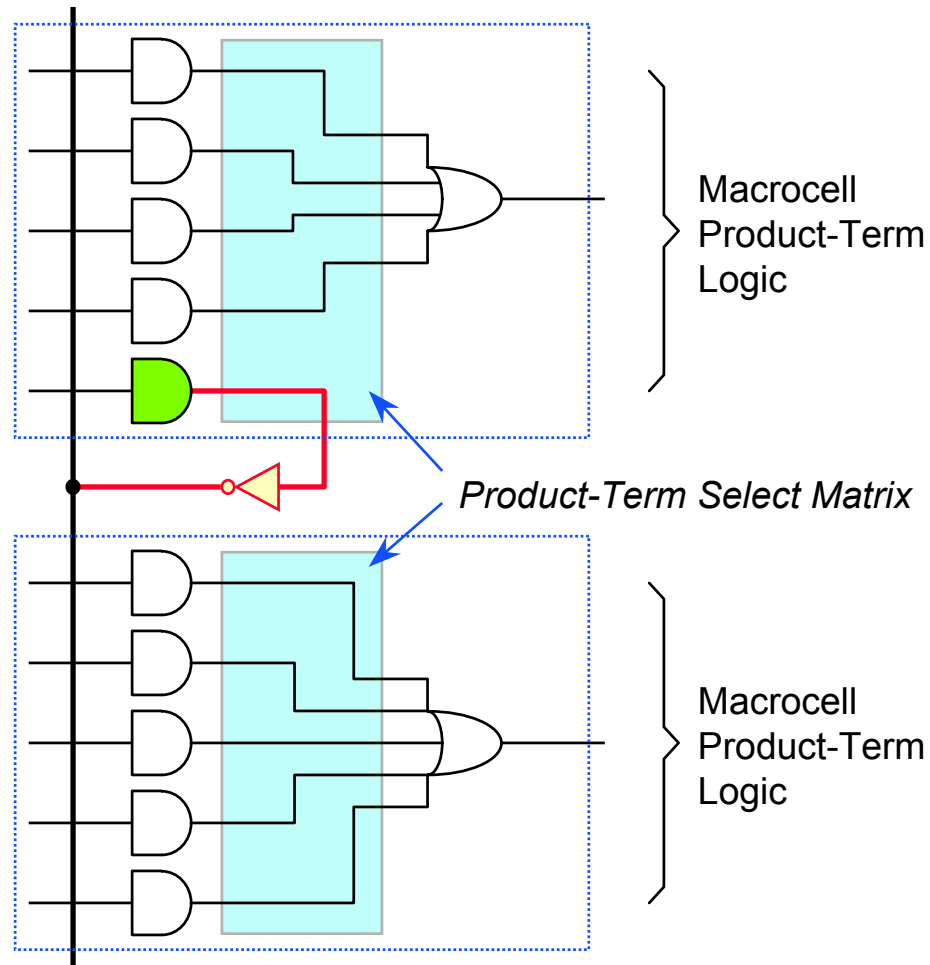


# MAX 7000E/S Macrocell



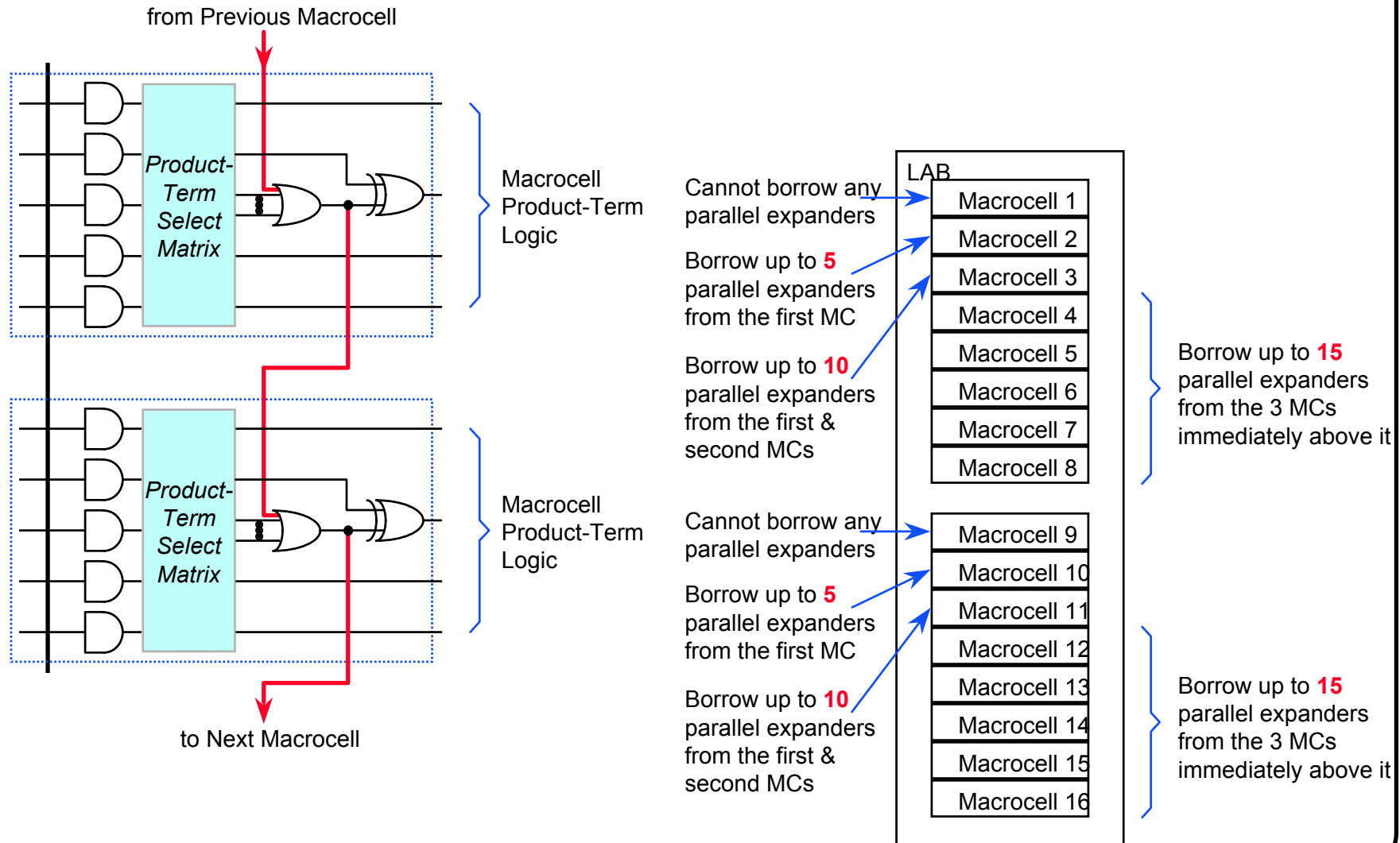


# Shareable Expanders



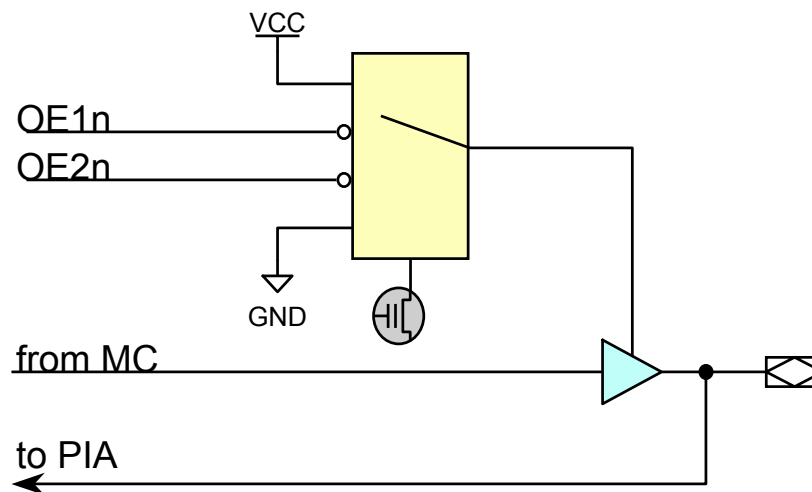


# Parallel Expanders



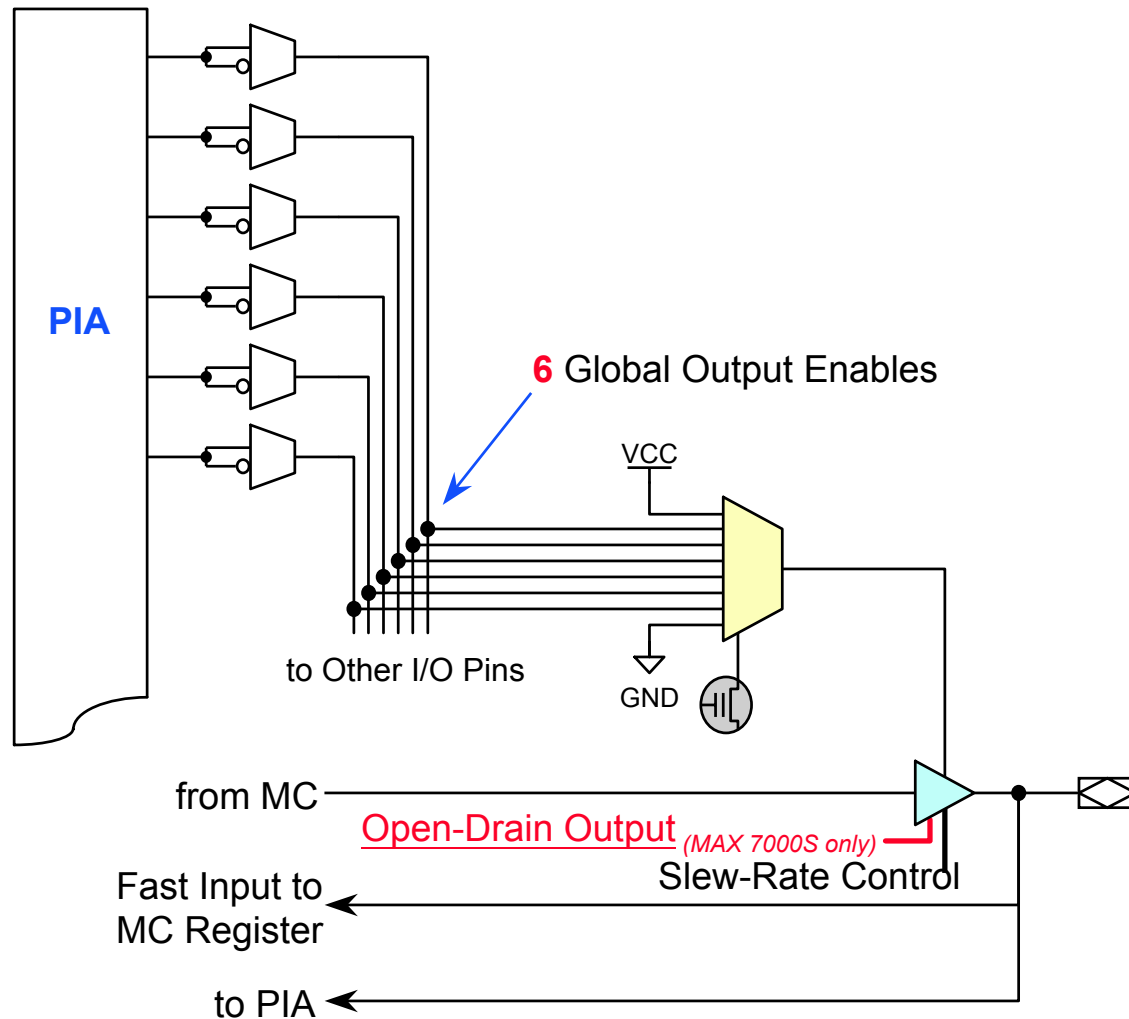


# MAX 7000 I/O Control Block





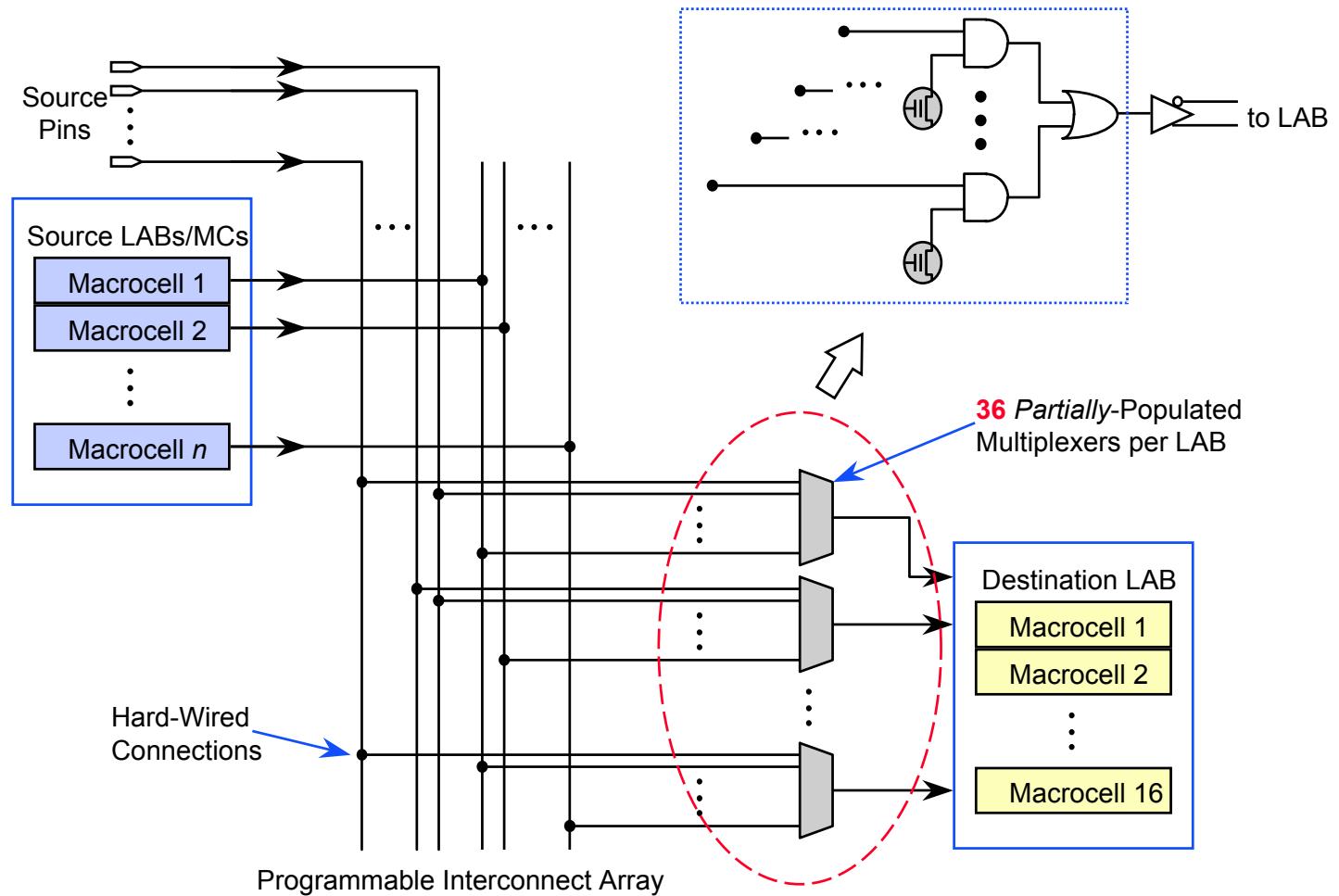
# MAX 7000E/S I/O Control Block





























# MAX 7000/E/S PIA

(Programmable Interconnect Array)





# MAX Vertical Migration

Device	44-Pin PLCC	44-Pin TQFP	48-Pin 0.8-mm BGA	84-Pin PLCC	100-Pin FineLine BGA™	100-Pin TQFP	144-Pin TQFP	168-Pin 0.8-mm BGA	208-Pin PQFP	256-Pin FineLine BGA
EPM7032										
EPM7064										
EPM7128										
EPM7256										
EPM7512										

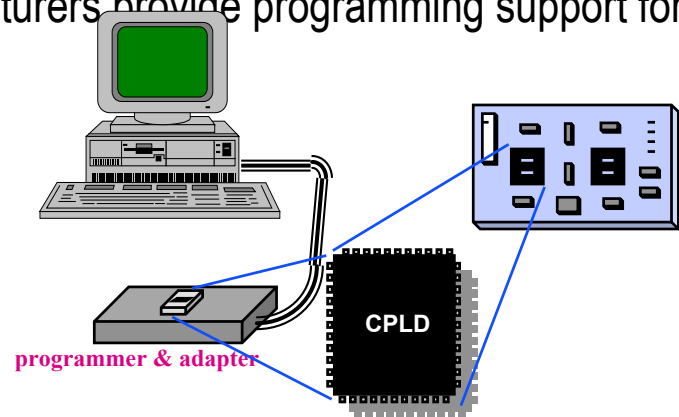


# MAX 7000/E/S Device Programming

## ◆ Program the device with external hardware

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

## ◆ MAX 7000S ISP

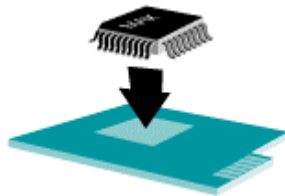




# What's ISP?

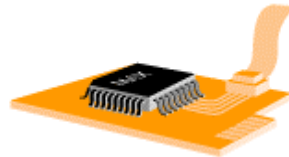
## ◆ ISP: In-System Programming

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



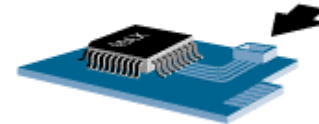
### Mount Unprogrammed

- \* Eliminates handling of devices
- \* Prevents bent leads



### Program In-System

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

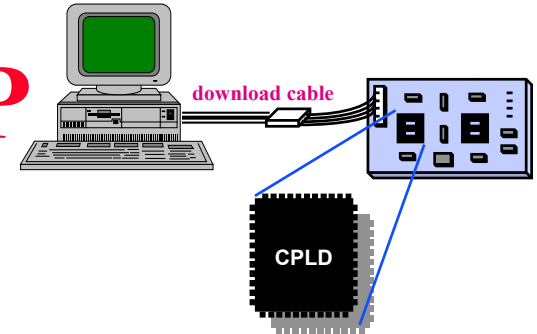


### Reprogramm in the Field

- \* No need to return system for upgrades
- \* Add enhancements quickly & easily



# MAX 7000S ISP

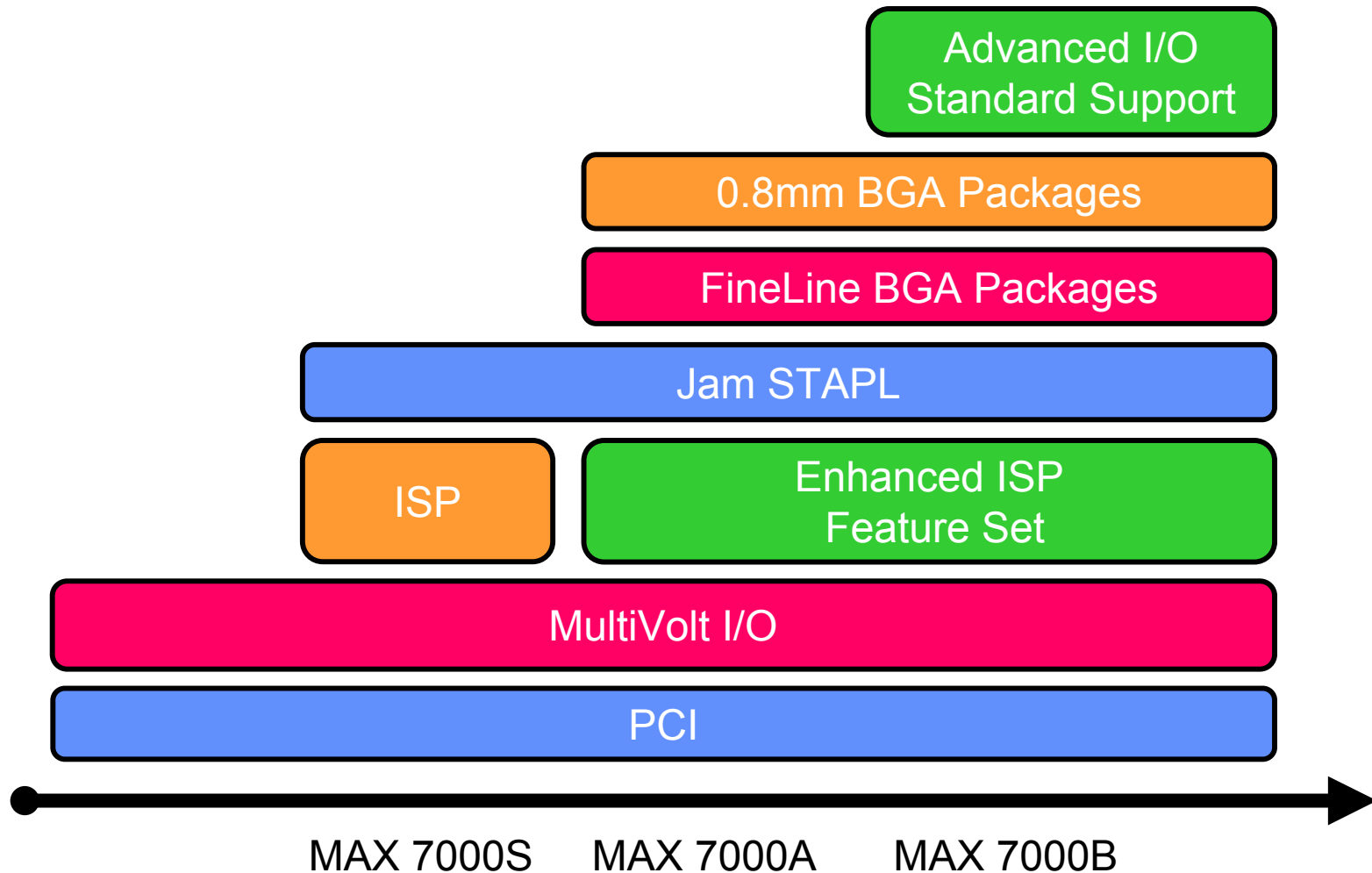


## ◆ MAX 7000S ISP

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



# Summary of MAX features





# 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



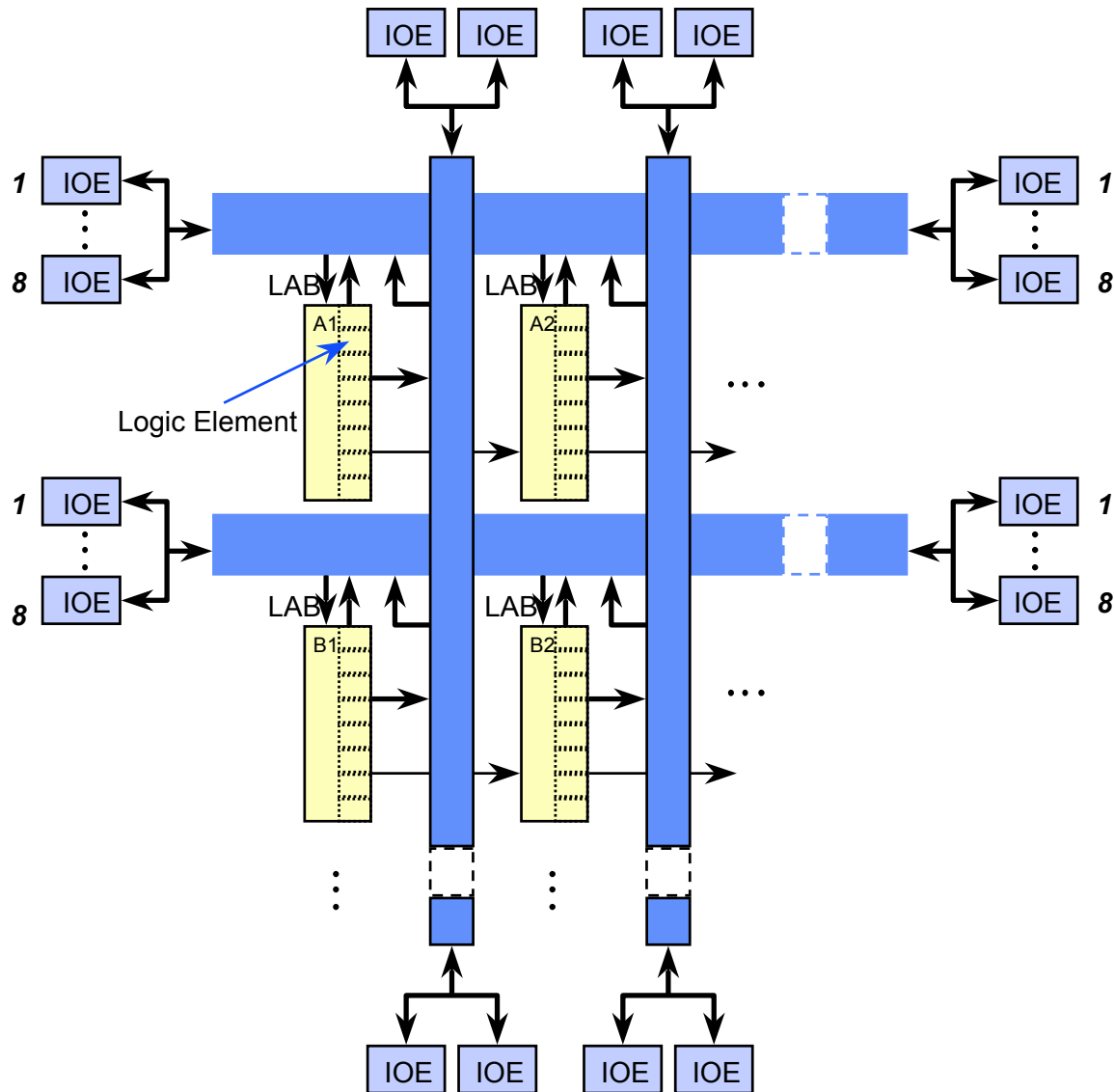
# FLEX 8000A Features

## ◆ 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

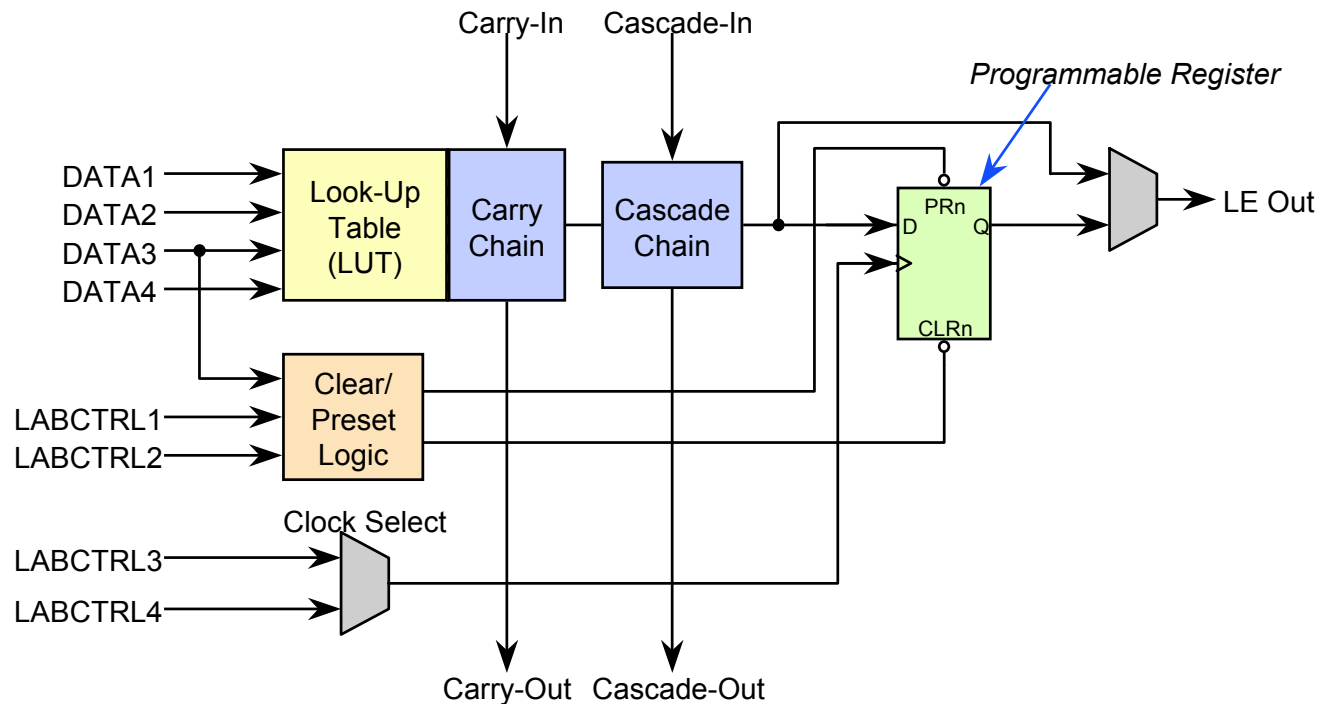


# FLEX 8000A Architecture



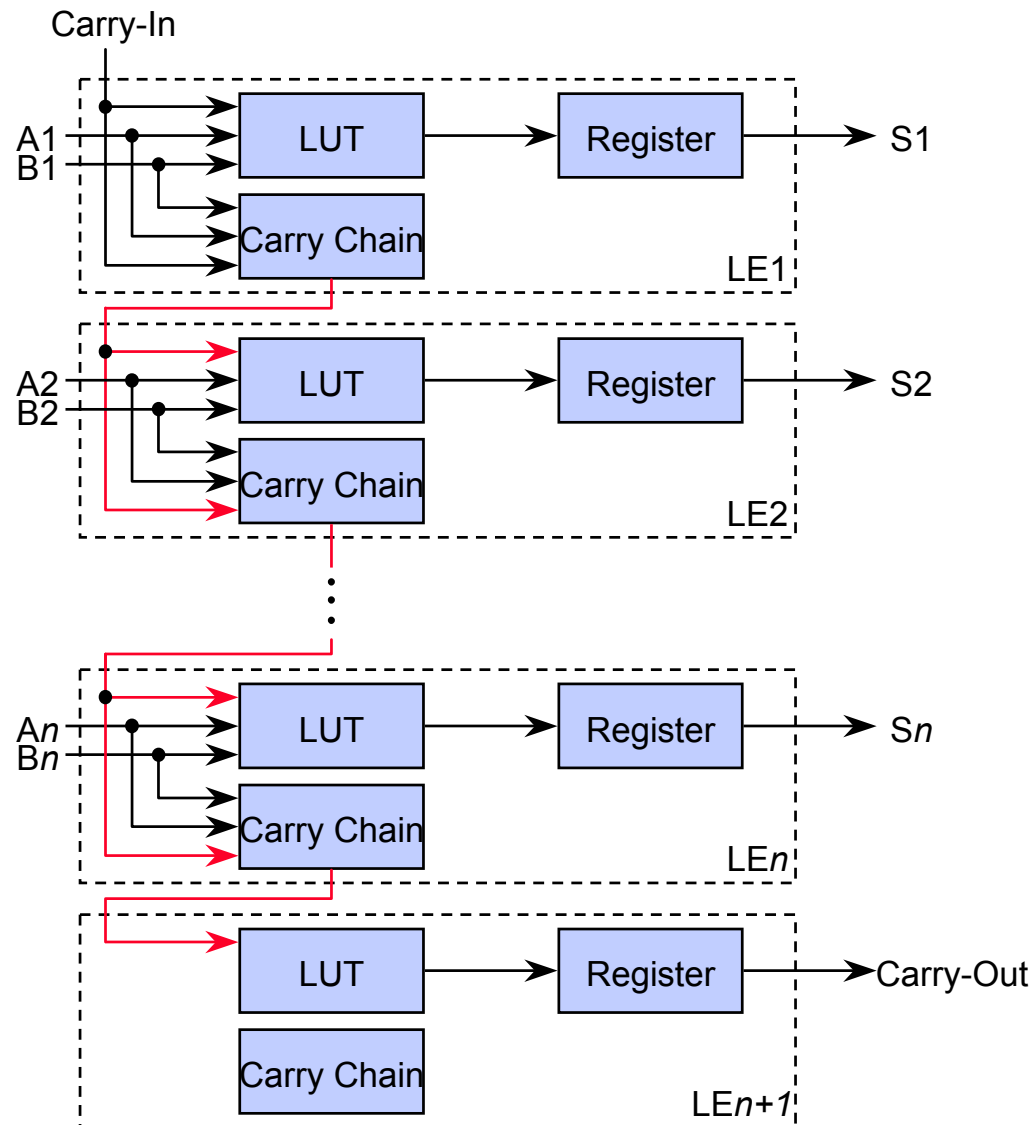


# FLEX 8000A Logic Element





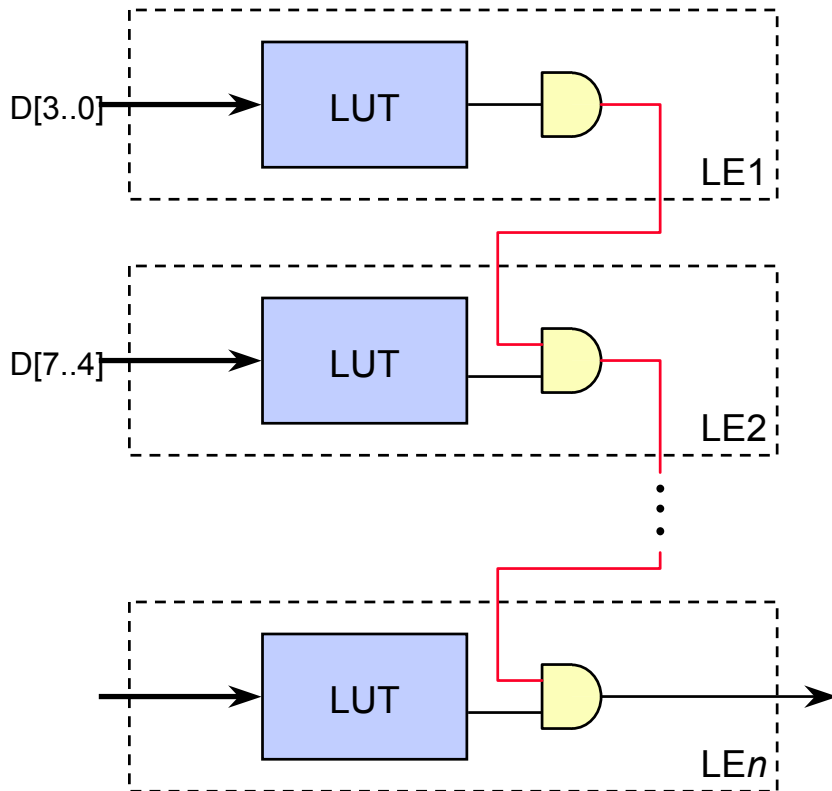
# Carry Chains



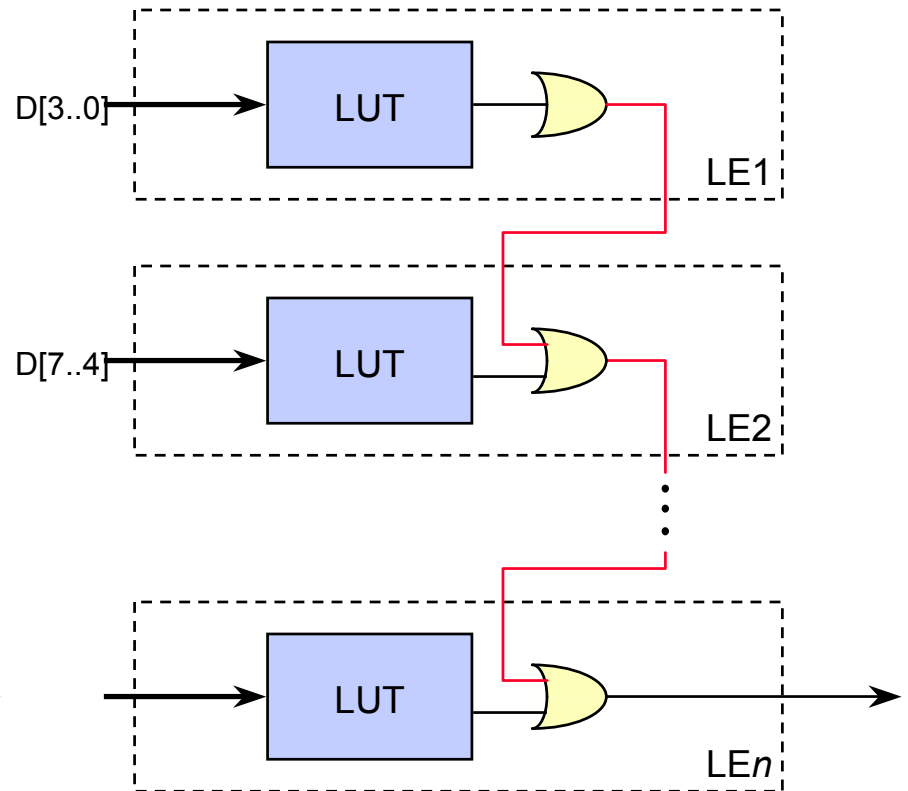


# Cascade Chains

AND Cascade Chain

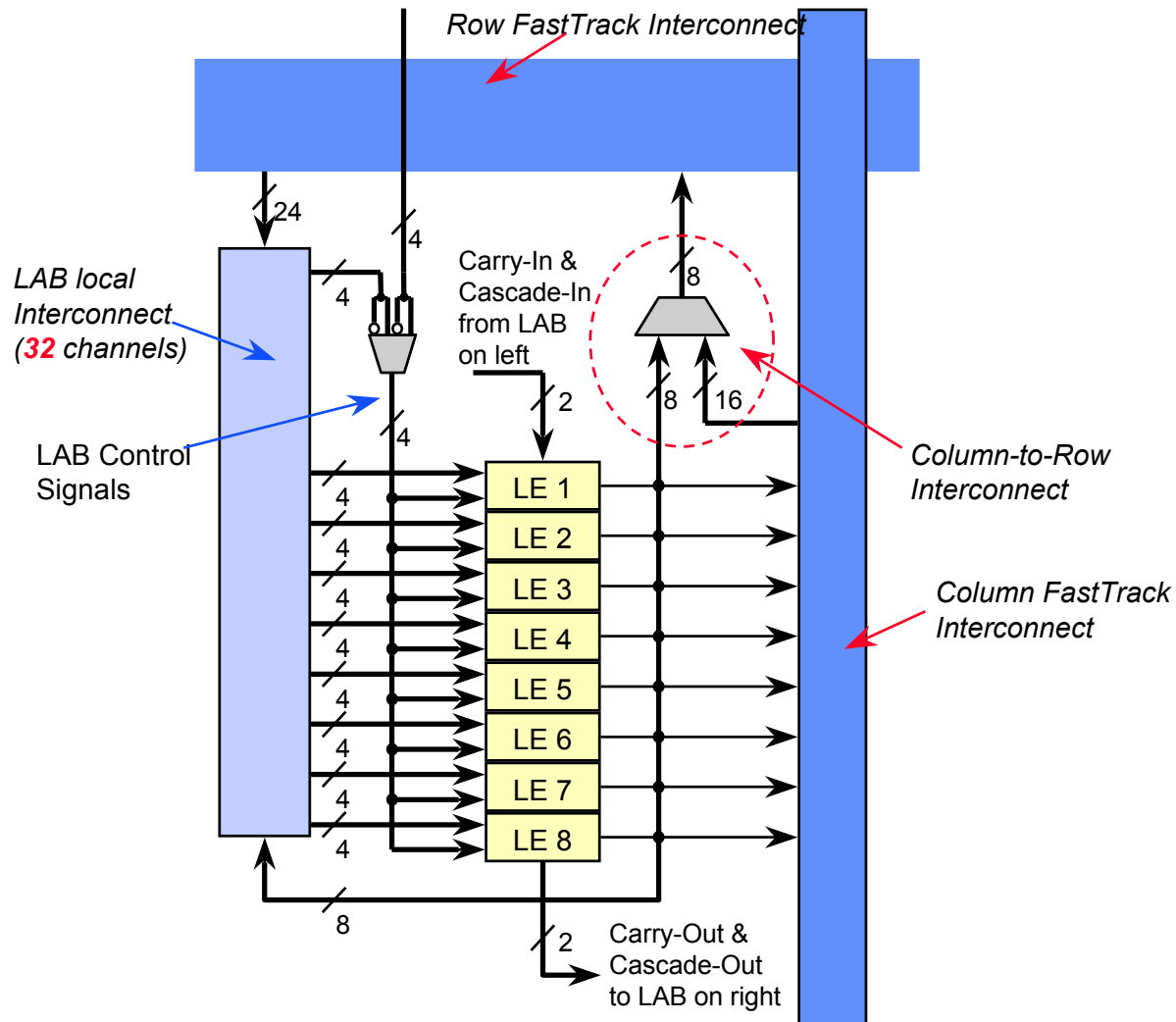


OR Cascade Chain



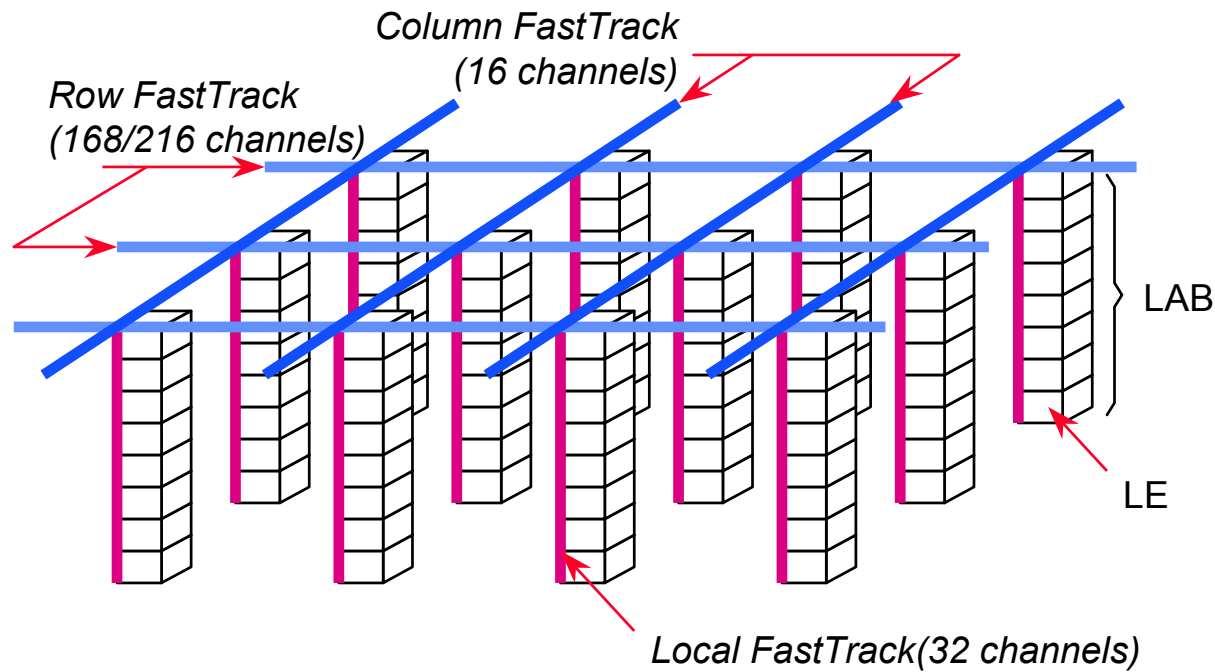


# FLEX 8000A Logic Array Block



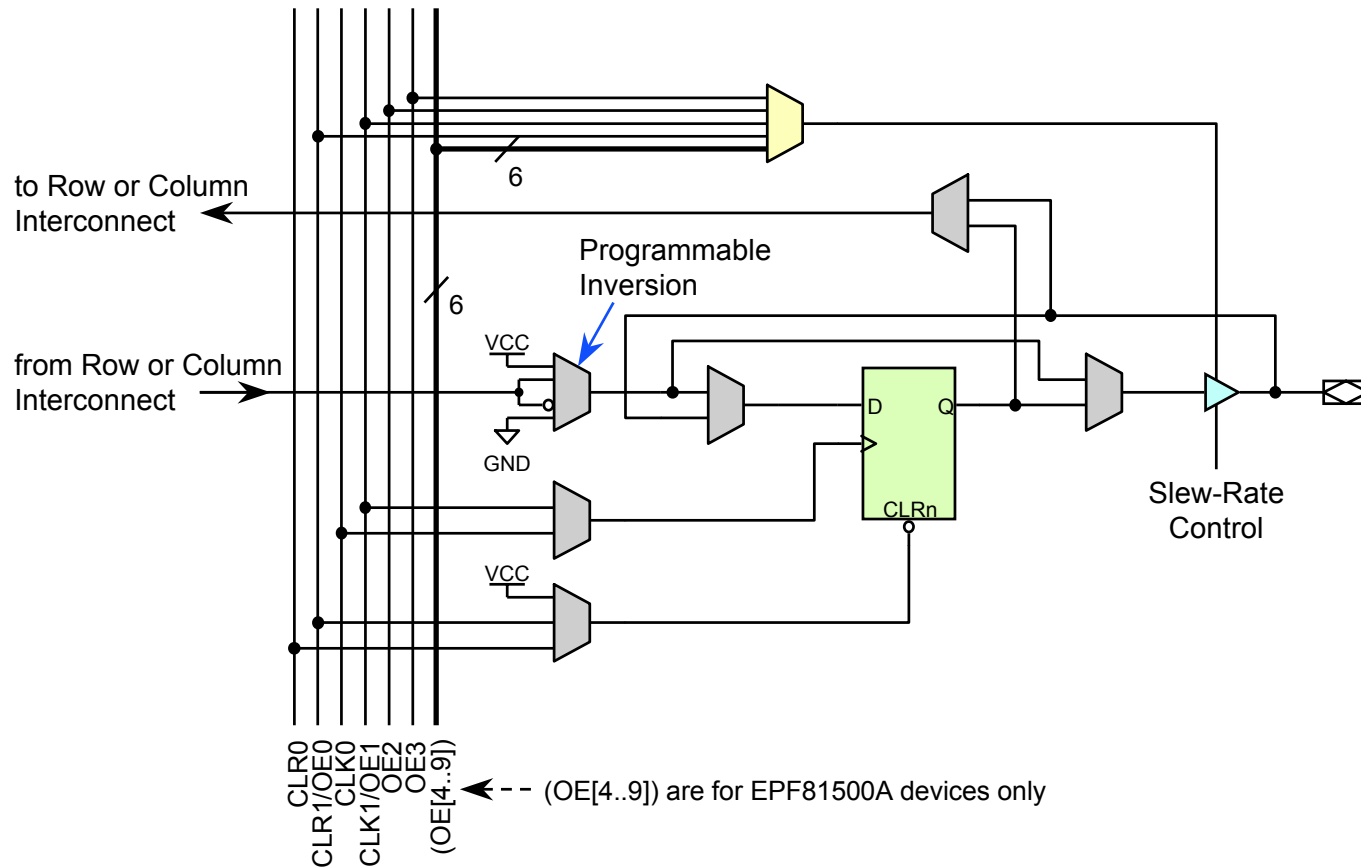


# FLEX 8000A FastTrack Interconnect





# FLEX 8000A I/O Element





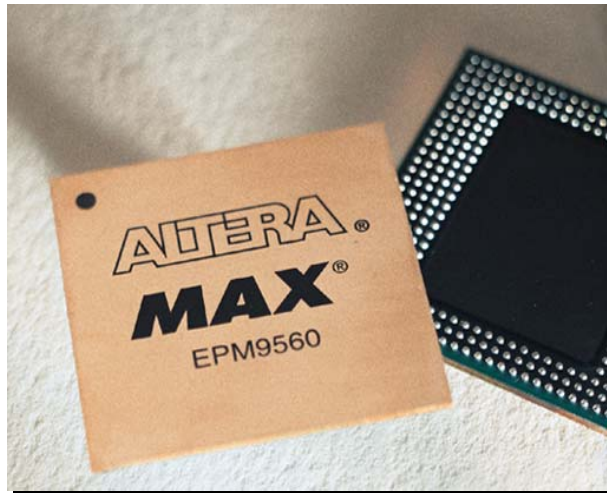
# FLEX 8000A Configuration

## ◆ 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
<b>AS</b> (Active Serial)	Serial configuration EPROM
<b>APU</b> (Active Parallel Up)	Parallel EPROM
<b>APD</b> (Active Parallel Down)	Parallel EPROM
<b>PS</b> (Passive Serial)	Serial data path (e.g. serial download cable)
<b>PPS</b> (Passive Parallel Synchronous)	Intelligent host
<b>PPA</b> (Passive Parallel Asynchronous)	Intelligent host





# MAX 9000A Devices





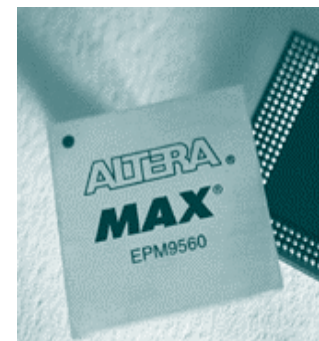
# MAX 9000A KEY FEATURE

## ◆ MAX 9000A main features...

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



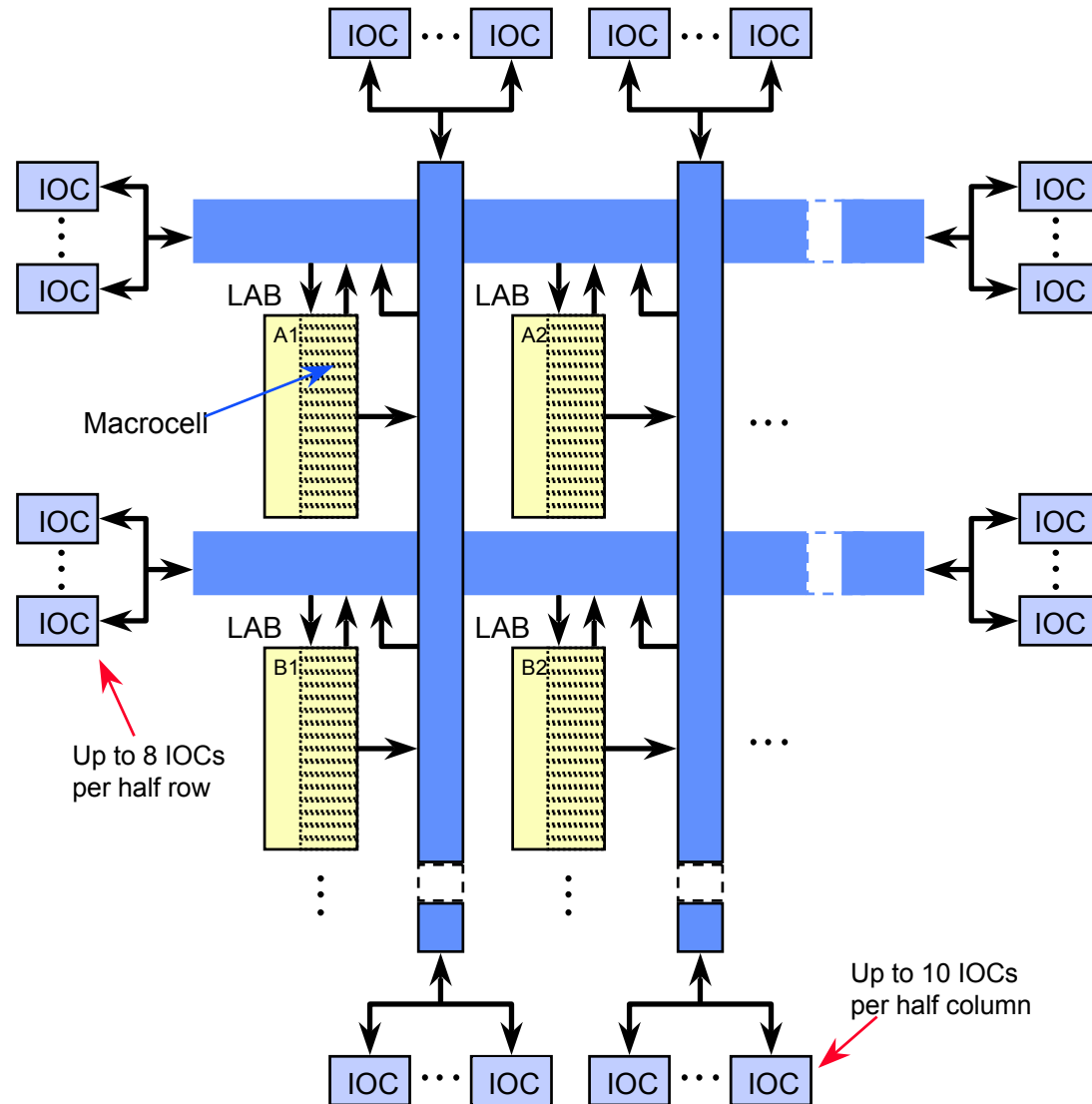
# MAX 9000A Family



Feature	EPM9320A	EPM9400	EPM9560A
Macrocells	320	400	560
Max. # FF	484	580	772
Packages	84 PLCC 208 RQFP 280 PGA 356 BGA	84 PLCC 208 RQFP 240 RQFP	208 RQFP 240 RQFP 304 RQFP 280 PGA 356 BGA

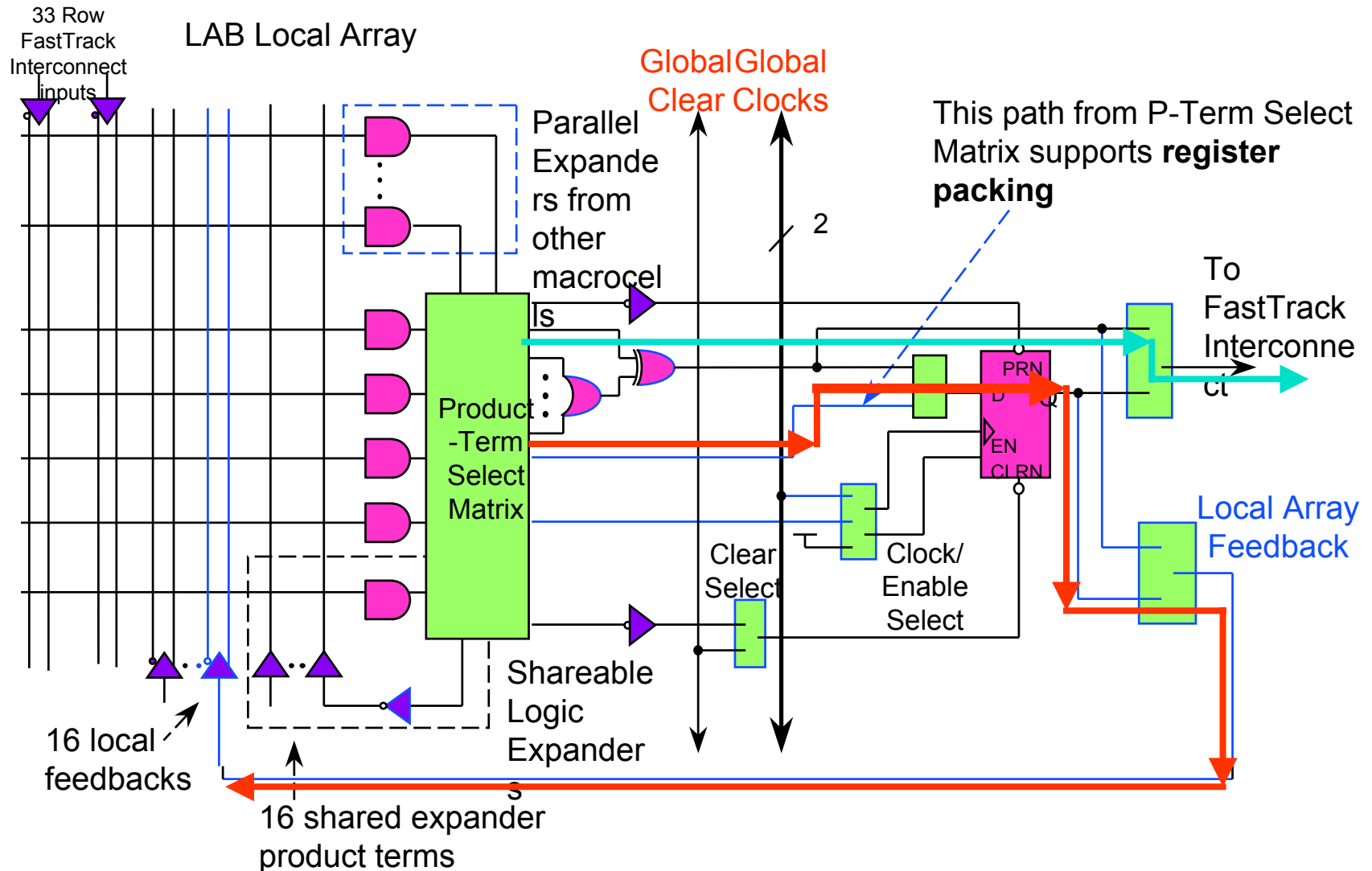


# MAX 9000 Architecture



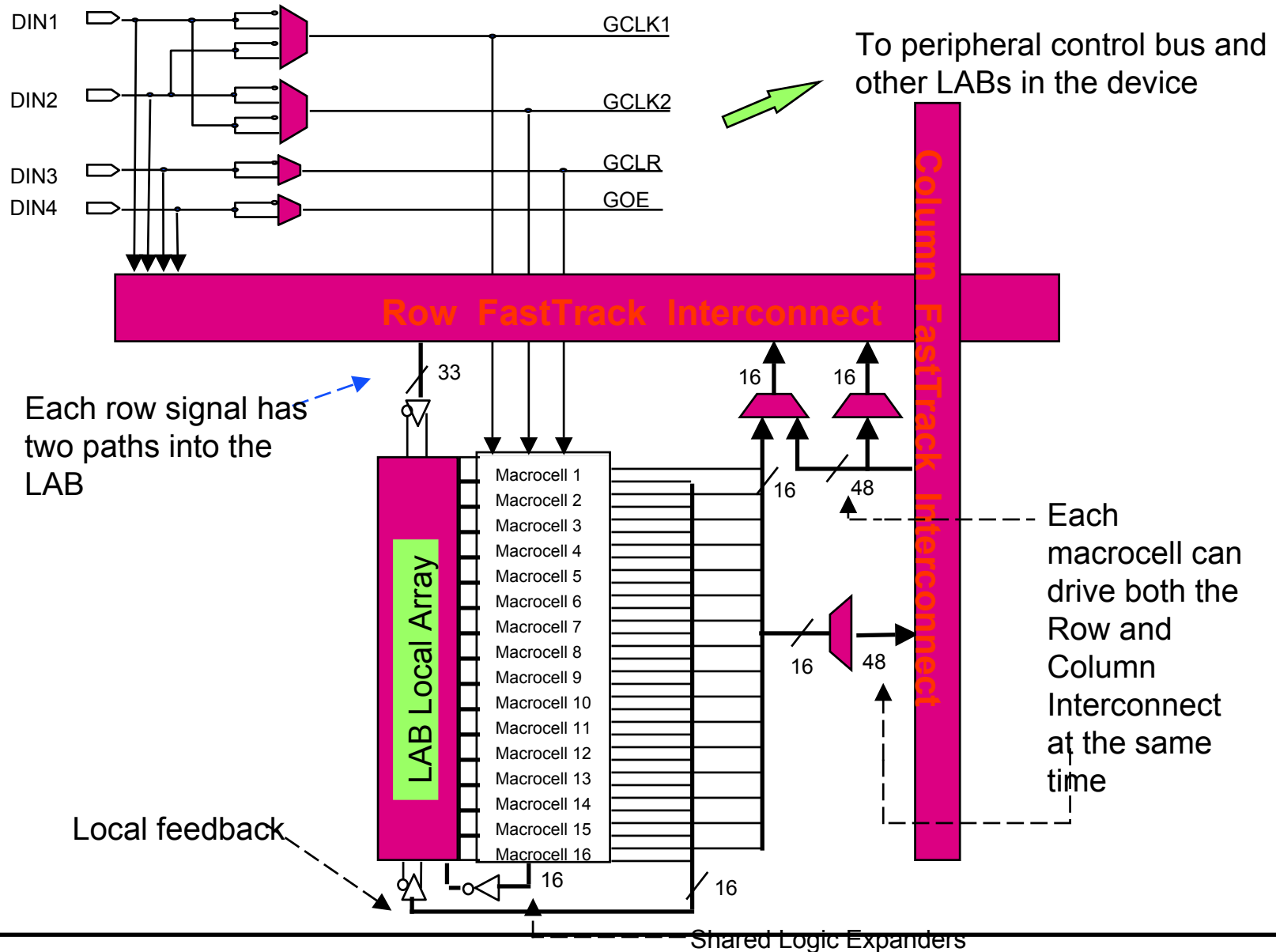


# MAX 9000A Macrocell



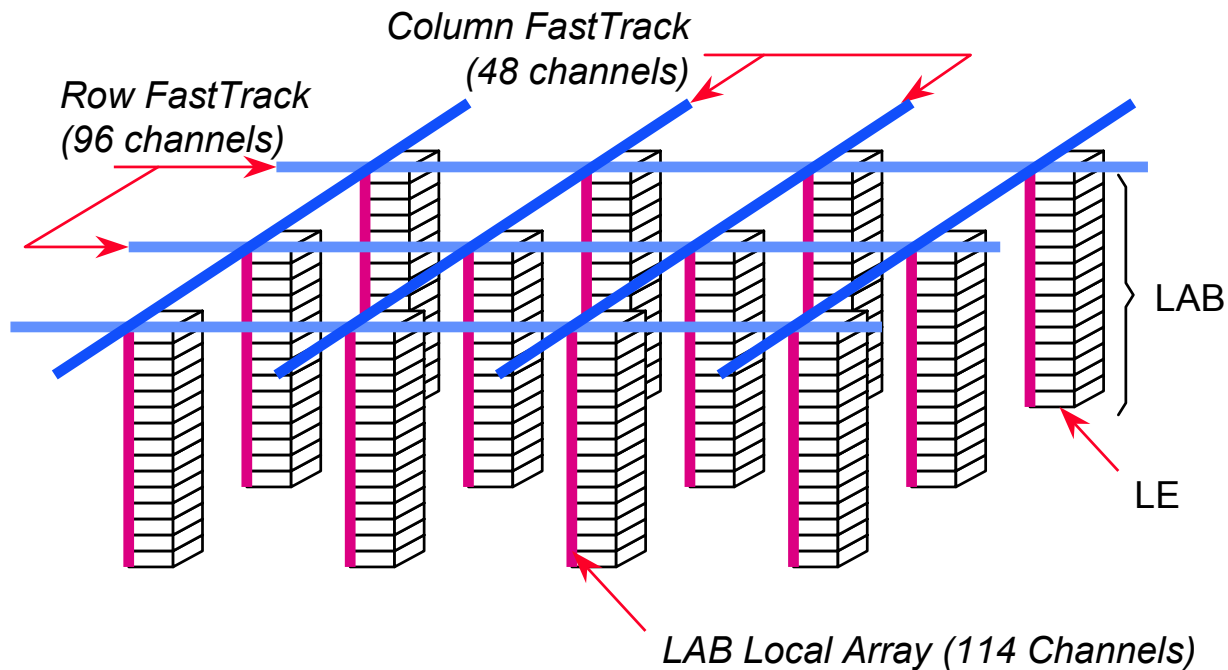


# MAX 9000A Logic Array Block



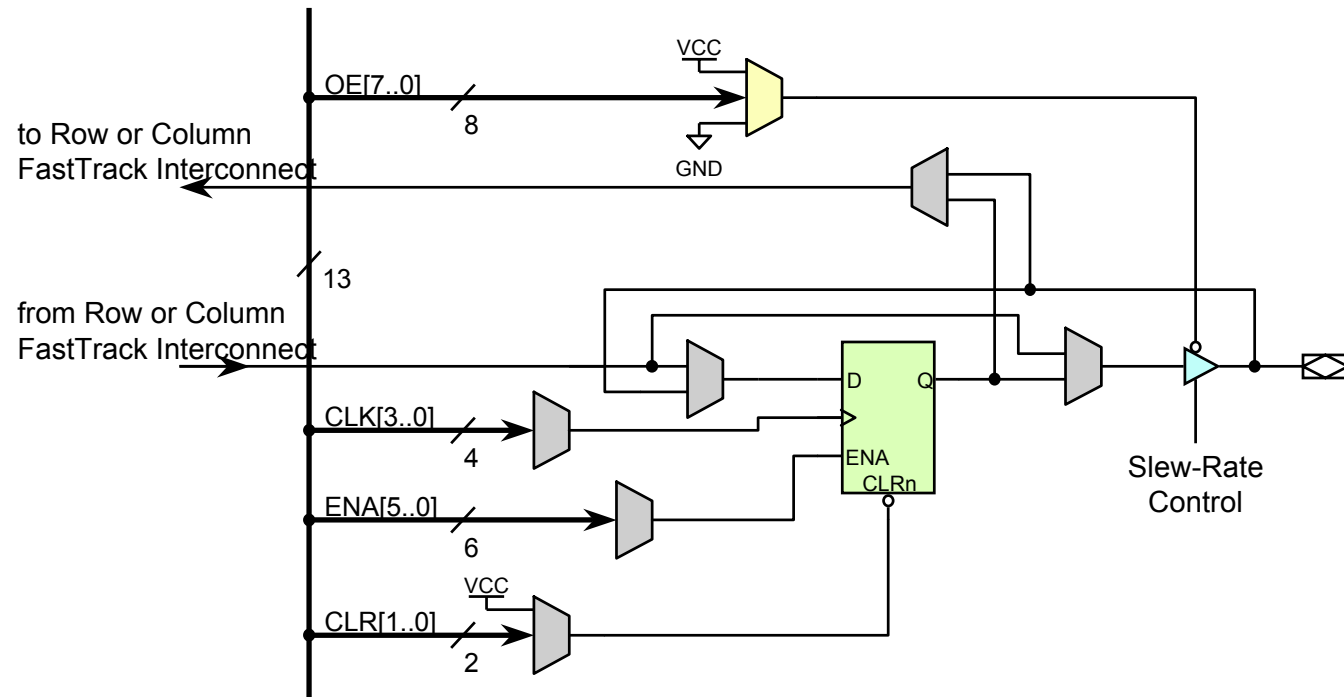


# MAX 9000 FastTrack Interconnect





# MAX 9000 I/O Cell



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

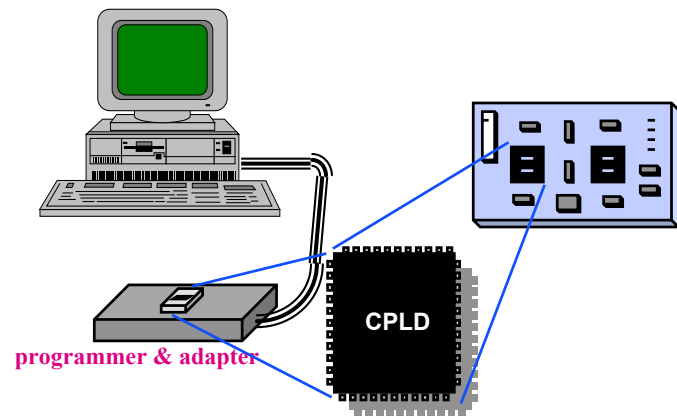


# MAX 9000 Device Programming

## ◆ 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

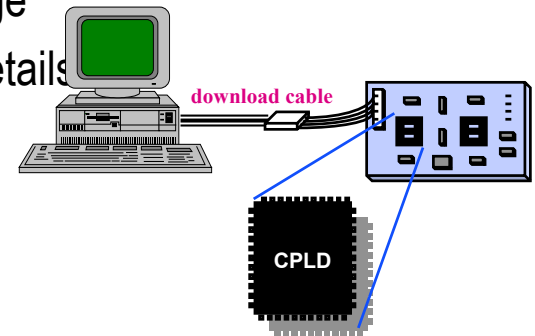




# MAX 9000 ISP

## ◆ 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*





# MAX Supports Jam STAPL for ISP

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

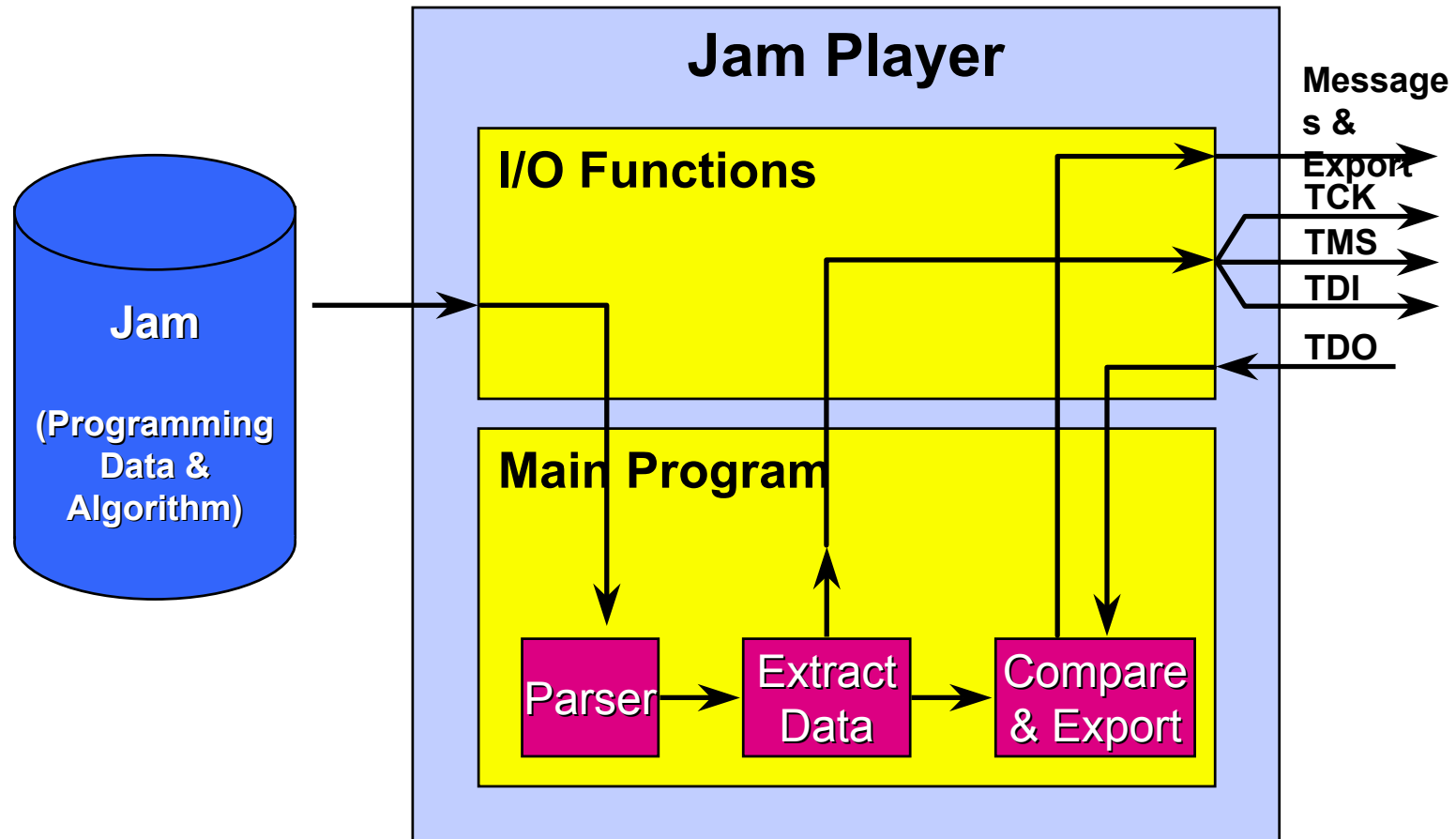


<http://www.jamisp.com>



# Embedded Programming using Jam Player

## Generating Programming Signals from the Jam File





# MAX Family Slew Rate Control

◆ **For designs without the above elements (or during prototype stages), the board may not be able to support the fast switching outputs of Altera devices**

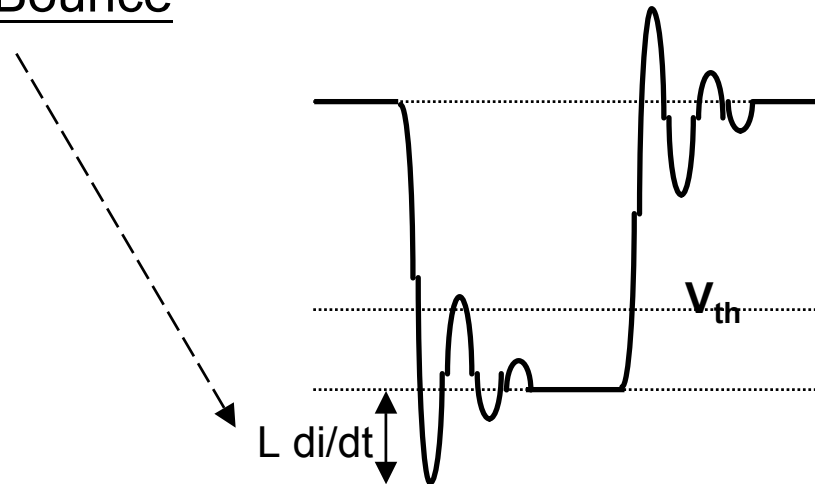
- Faster switching outputs cause higher transient currents in outputs as they discharge load capacitance
- These higher currents can cause ground bounce (ringing)



◆ The magnitude of this ringing is  $V = L \, di/dt$

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

Ground Bounce





# Programmable Speed/ Power Control

- ◆ **MAX devices offer low-power OR high speed operation through the Turbo Bit logic option**
- ◆ **Power dissipation can be reduced by 50% or more**
- ◆ **This is controllable for the entire device OR on a macrocell by macrocell basis**
  - The user can have a section of the design operating in high performance (Turbo Bit = on) and in the same device, other sections may be operating in low power (Turbo Bit = off)
  - MAX 7000 devices: macrocells running at low power (Turbo Bit = off) incur a delay  $t_{LPA}$  (8 ns for -5 speed grade) for the  $t_{LAD}$ ,  $t_{LAC}$ ,  $t_{IC}$ ,  $t_{ACL}$ ,  $t_{EN}$ ,  $t_{SEXP}$  parameters
  - MAX 9000 devices: macrocells running at low power (Turbo Bit = off) incur a delay  $t_{LPA}$  for the LAB local array delay ( $t_{local}$ )

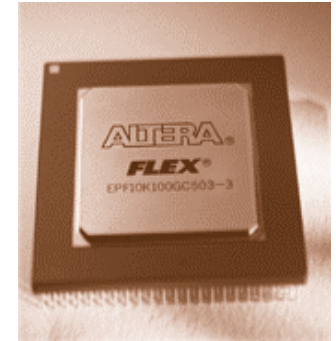




# FLEX 10K Devices



# FLEX 10K Families



Features	EPF10K10 EPF10K10A	EPF10K20	EPF10K30 EPF10K30A	EPF10K40	EPF10K50 EPF10K50V	EPF10K70	EPF10K100 EPF10K100A	EPF10K130V	EPF10K250A
Typical Gates	10,000	20,000	30,000	40,000	50,000	70,000	100,000	130,000	250,000
Logic Elements	576	1,152	1,728	2,304	2,880	3,744	4,992	6,656	12,160
RAM Bits	6,144	12,288	12,288	16,384	20,480	18,432	24,576	32,768	40,960
Registers	720	1,344	1,968	2,576	3,184	4,096	5,392	7,120	12,624
Max. User I/O	134	189	246	189	310	358	406	470	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<sub>(for EPF10K100GC503-3DX device only)</sub>
- Flexible package options
  - Pin-compatibility with other FLEX 10K devices in the same packages



# Altera 10KE Device

## BREAKTHROUGH PERFORMANCE

- ★ DESIGNED FOR PCI
- ★ 100-MHz SYSTEM SPEED
- ★ 150-MHz FIFOs

## Next- Generation Packaging

- ◆ 1.0-mm FineLine BGA™ Packages
- ◆ Requires Half the Board Area
- ◆ Minimizes Cost

## Advanced Process Technology

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

## Embedded Architecture Evolution

- ◆ Dual-Port RAM
- ◆ 4-Kbit EAB with x16 Width
- ◆ PCI-Compliant I/O

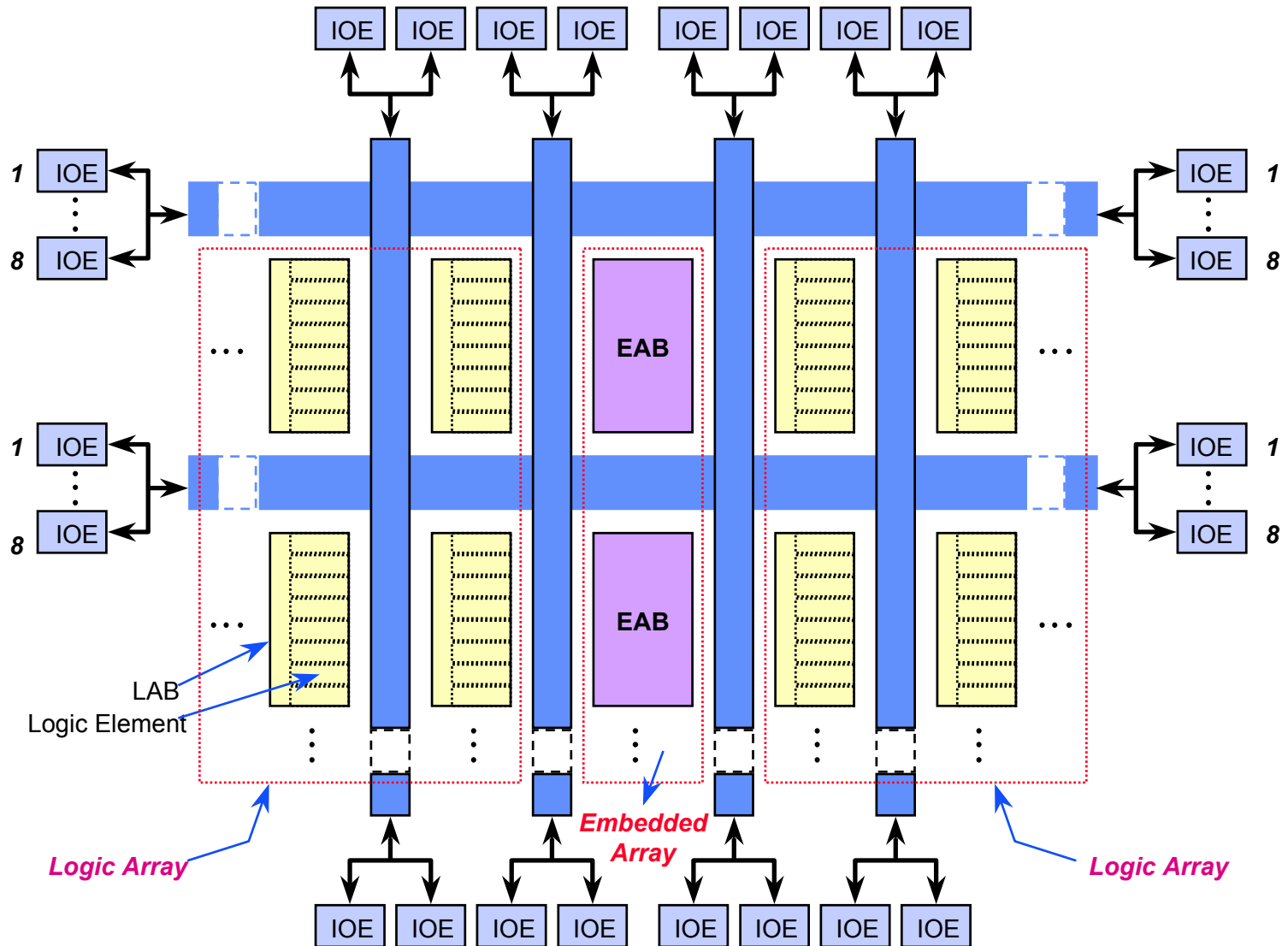


# Flex10KE Family Member

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



# FLEX 10K Architecture





# What is the EAB?

## ◆ What is the EAB?

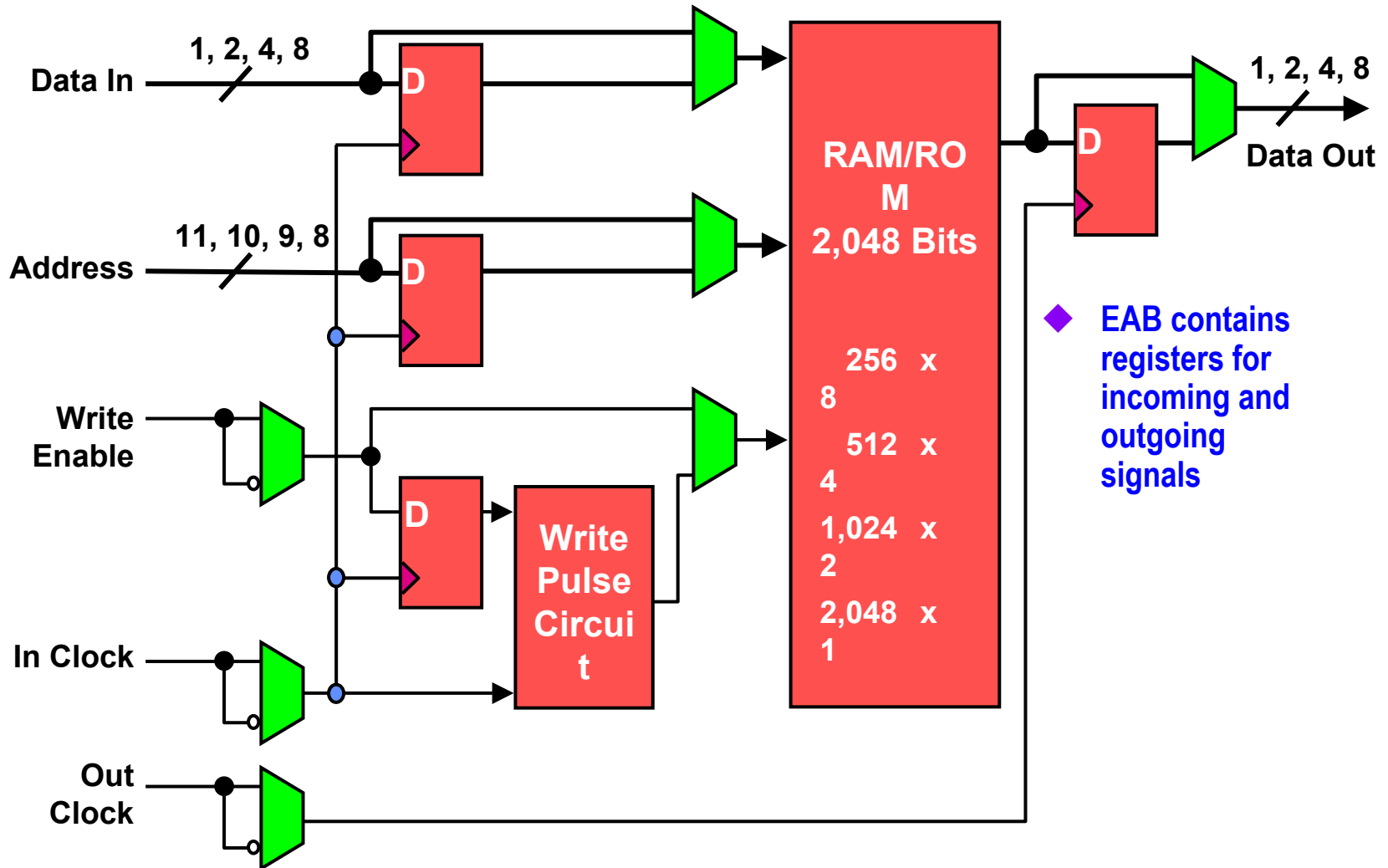
- Larger block of RAM embedded into the PLD
- Can be preloaded with a pattern
- EAB size is flexible - 256x8 / 512x4 / 1024x2 / 2048x1
- You can combine EABs to create larger blocks
- Using RAM does not impact logic capacity

## ◆ EAB as logic

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

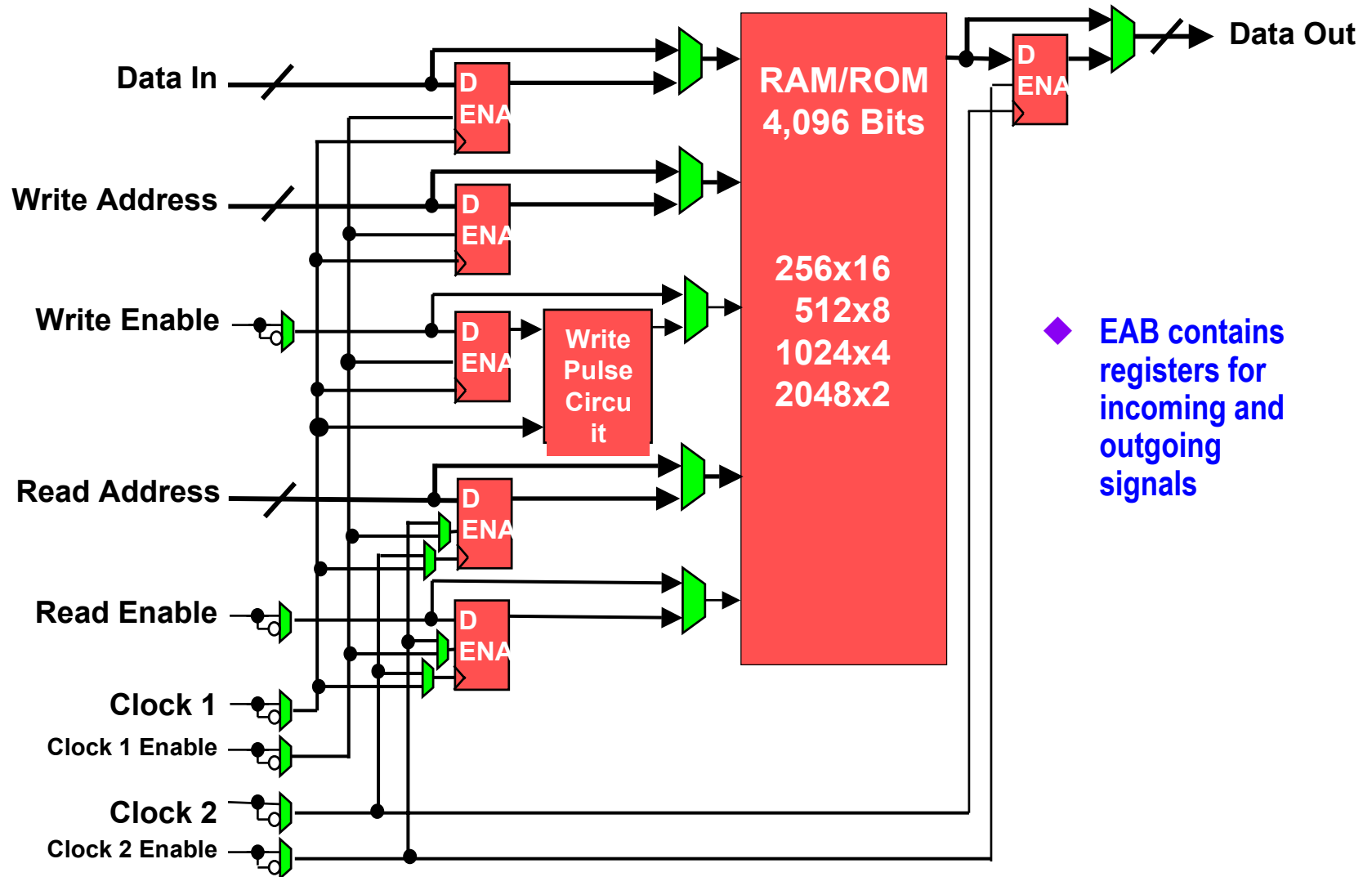


# FLEX 10K/V/A EAB



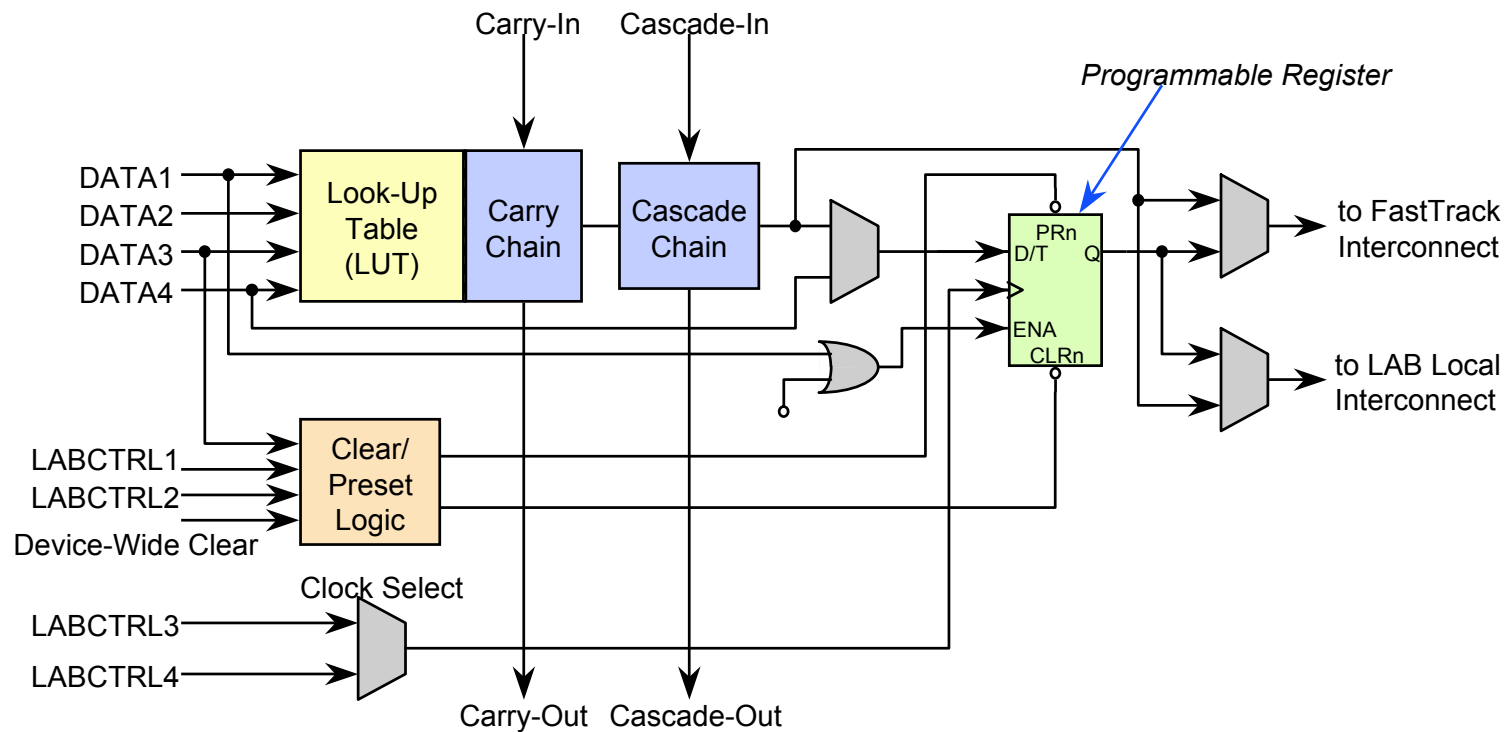


# 10KE EAB



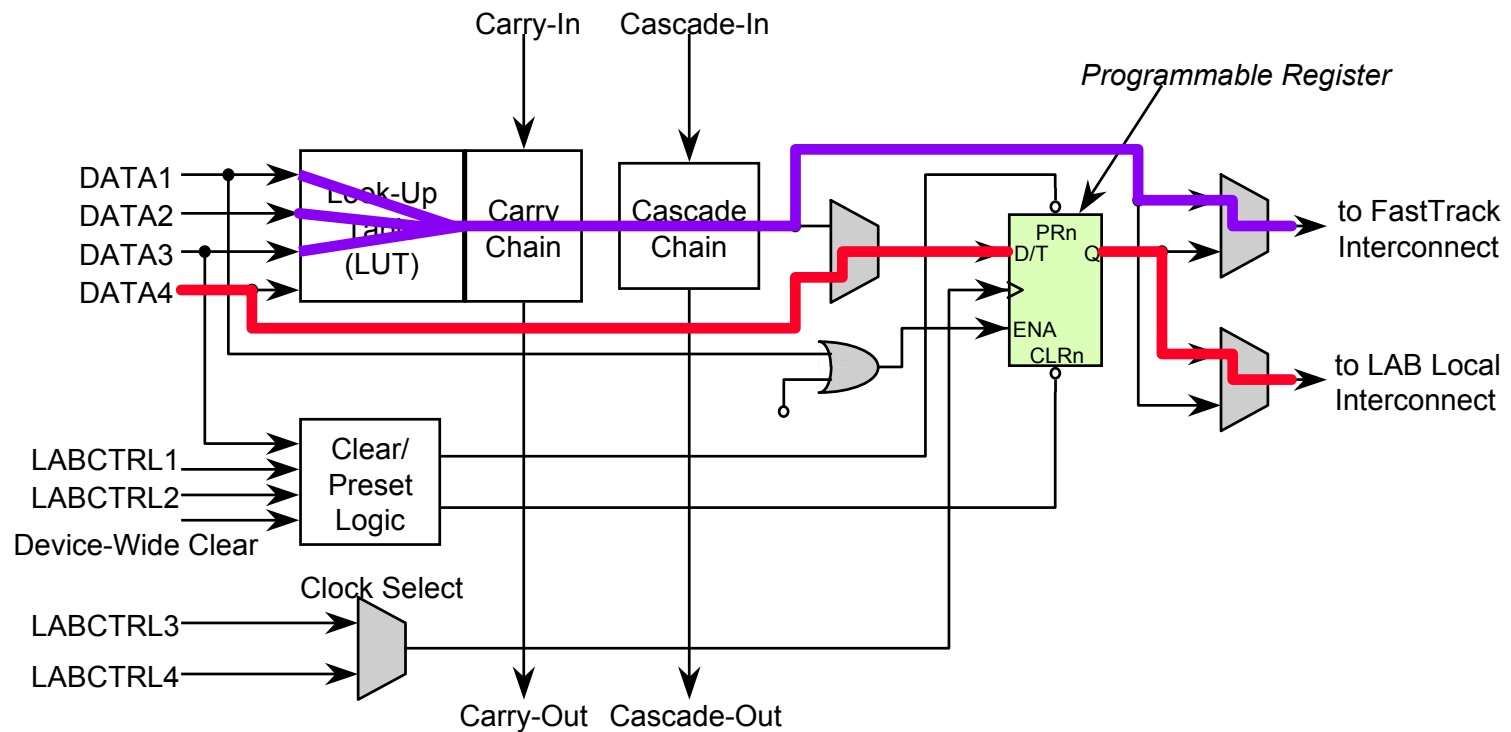


# FLEX 10K Logic Element



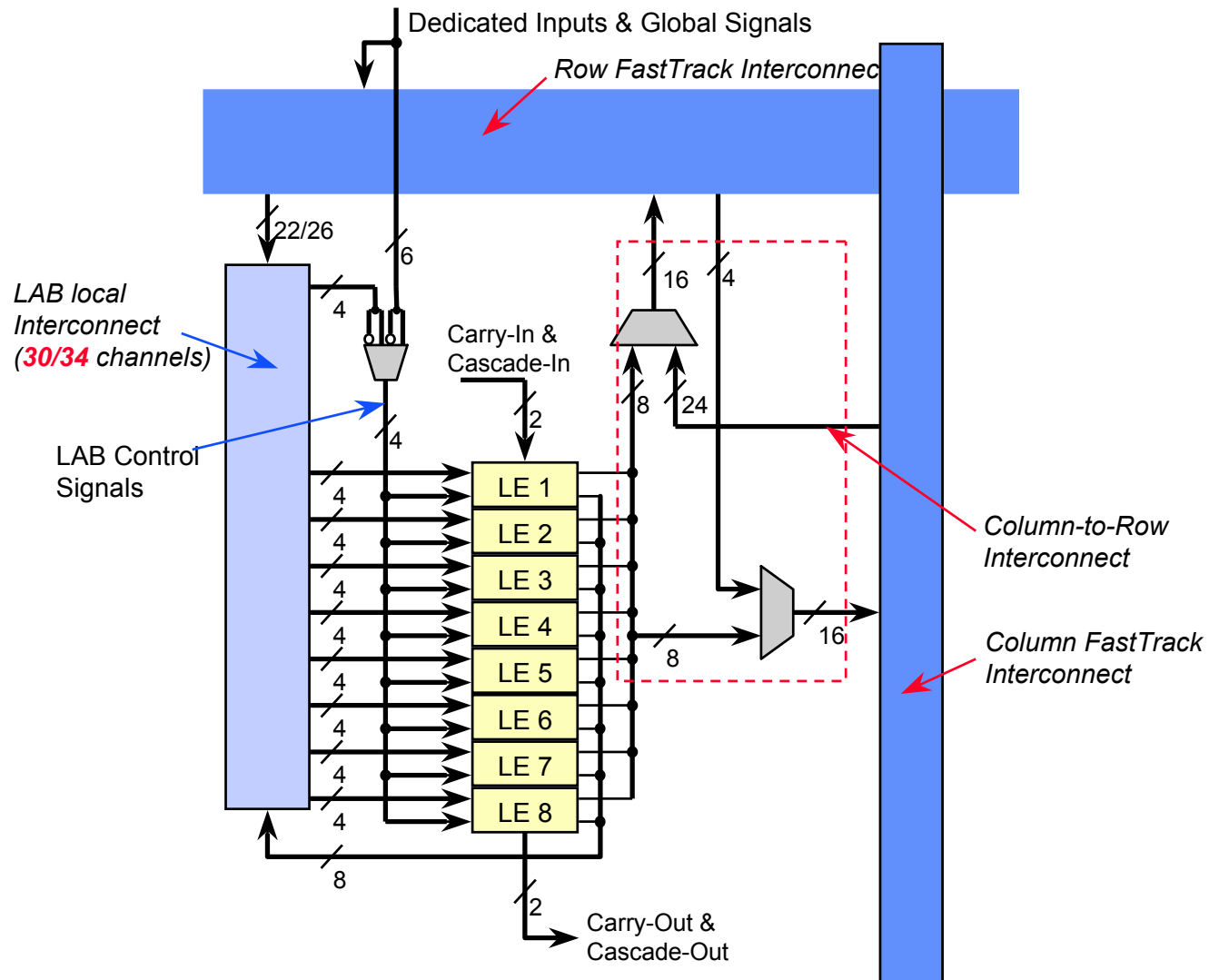


# FLEX 10K Register Packing



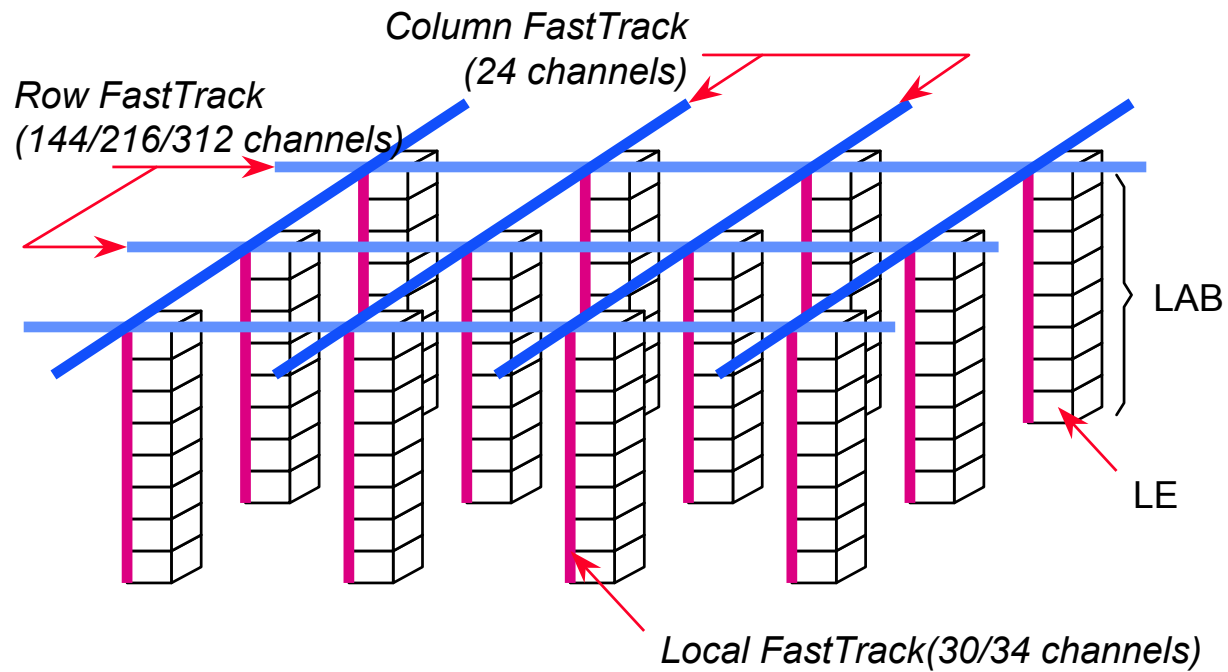


# FLEX 10K Logic Array Block



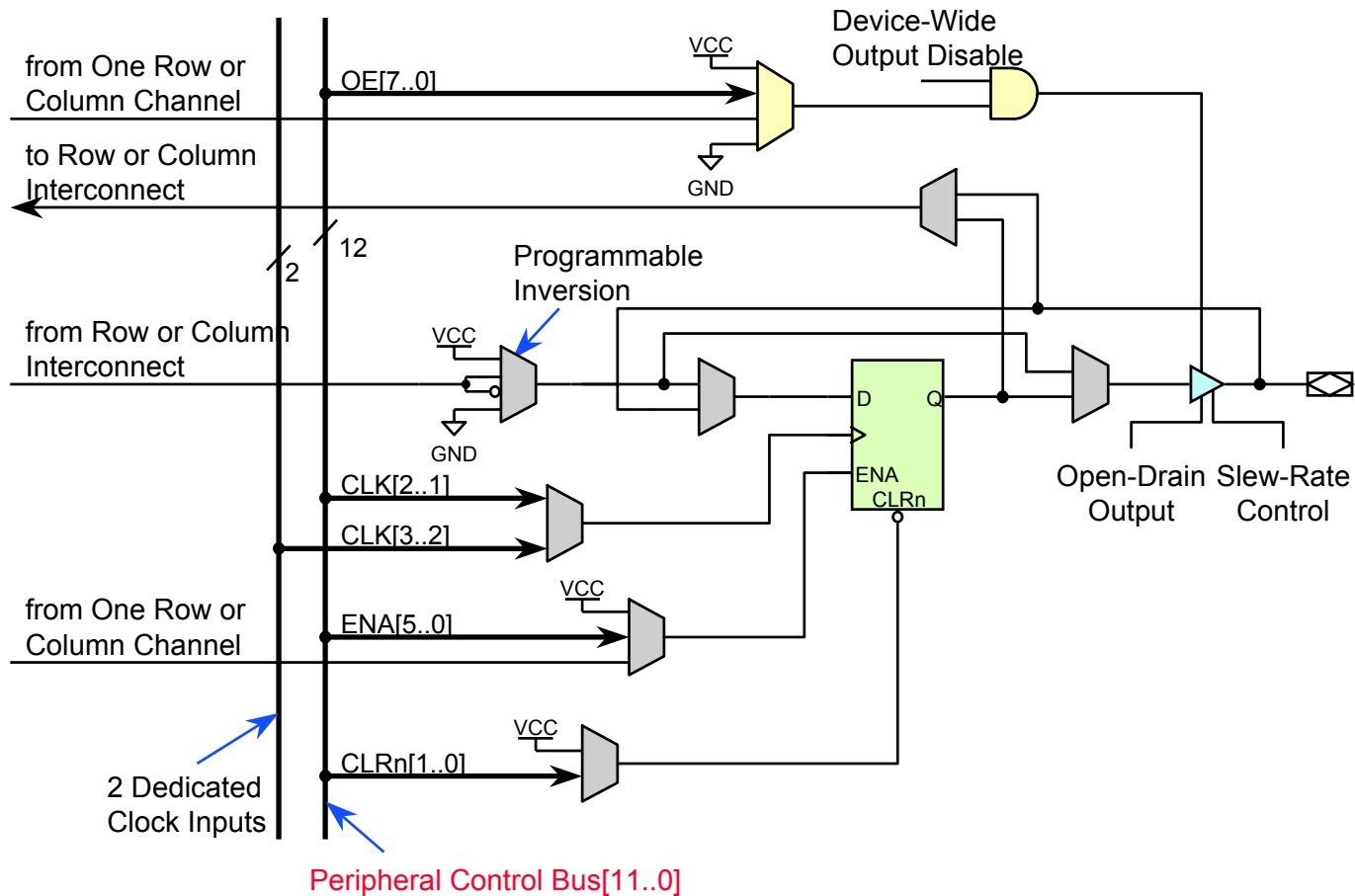


# FLEX 10K FastTrack Interconnect





# FLEX 10K I/O Element

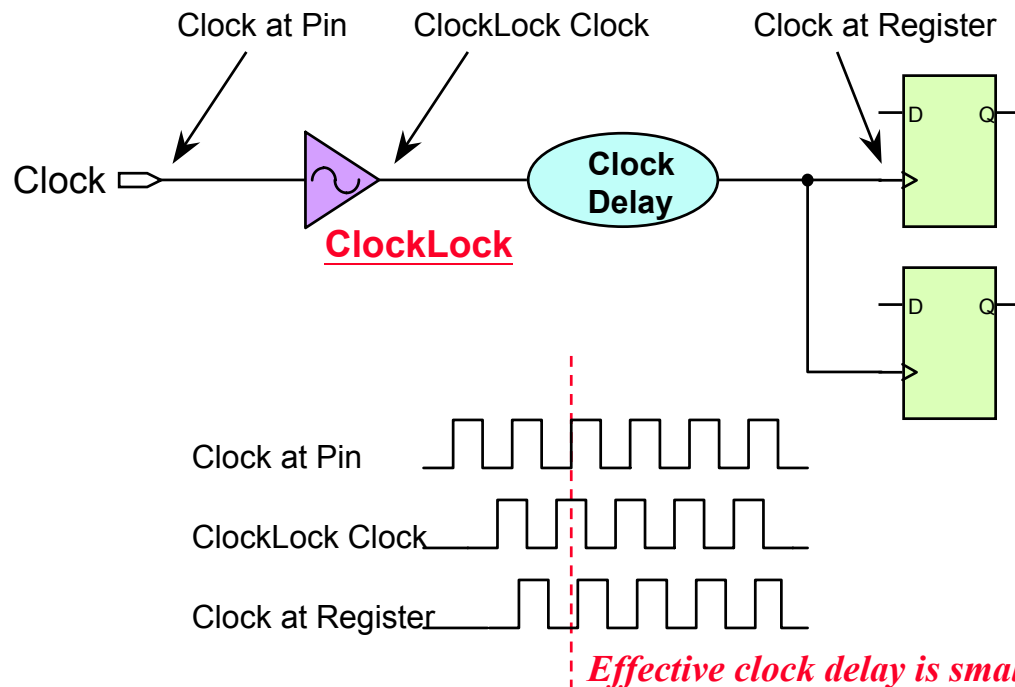




# ClockLock Feature

## ◆ 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
<b>PS</b> (Passive Serial)	Altera's EPC1 configuration EPROM, BitBlaster or ByteBlaster download cable, serial data source
<b>PPS</b> (Passive Parallel Synchronous)	Intelligent host, parallel data source
<b>PPA</b> (Passive Parallel Asynchronous)	Intelligent host, parallel data source
<b>JTAG</b>	JTAG controller



# Configuration Application Notes, Data Sheets

## ◆ Application Notes

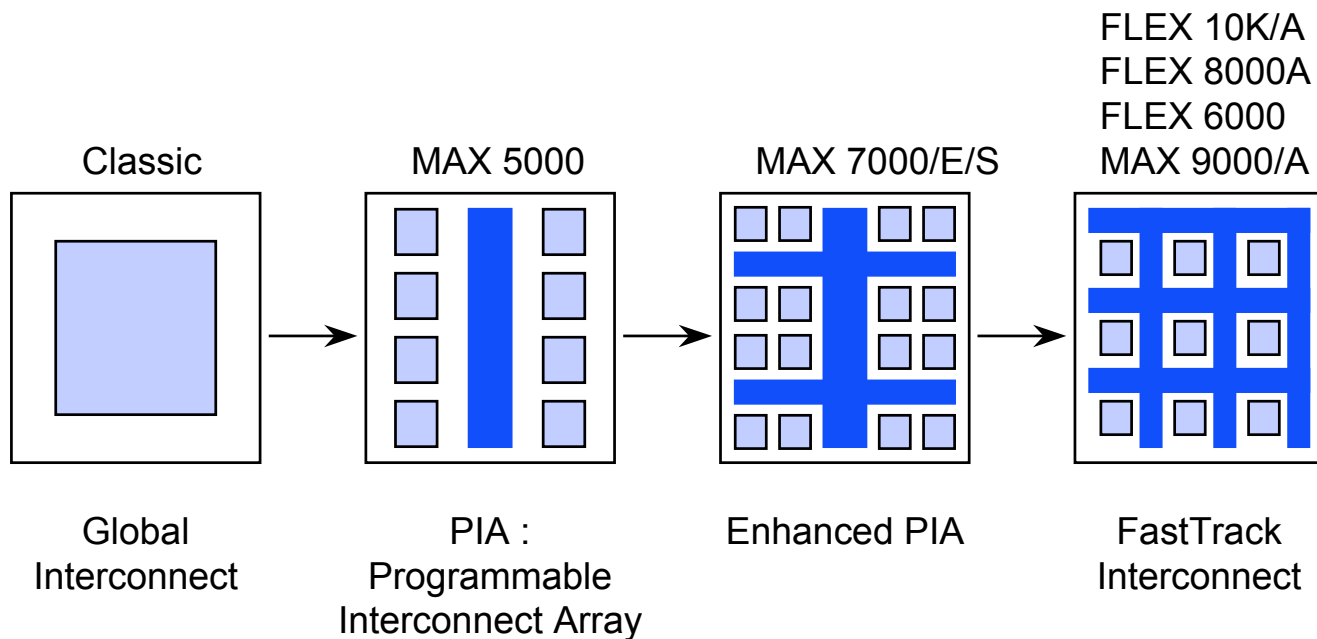
- [AN 33](#): Configuring FLEX 8000 Devices
- [AN 38](#): Configuring Multiple FLEX 8000 Devices
- [AN 87](#): Configuring FLEX 6000 Devices

## ◆ Data Sheets

- BitBlaster Serial Download Cable
- ByteBlasterMV Parallel Port Download Cable
- Configuration Devices for FLEX Devices
- Altera Programming Hardware



# Altera Architecture Evolution





# FLEX 6000 Device Family

## ◆ FLEX 6000 main features

- OptiFLEX™ Architecture
- Gate Count from 10,000 to 24,000 Gates
- 5.0 V, 0.5  $\mu$ m, TLM, SRAM Process  
(FLEX6000A 3.3 V, 0.35  $\mu$ m)
- 125-MHz Performance (16-Bit Counter)
- PCI-Compliant
- Pin Migration
- One Output Enable per Pin
- MultiVolt™ I/O
- High-Pin-Count TQFP, PQFP & BGA Packages

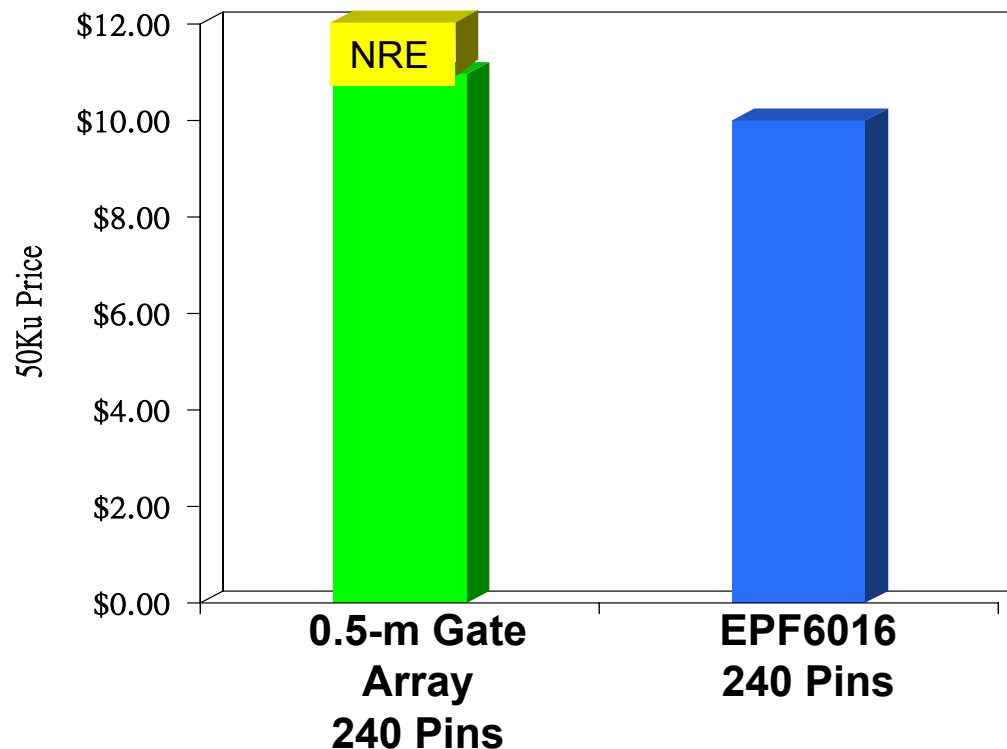


# Pricing vs. Gate Arrays

## ◆ Competitive with Gate Array Unit Cost

## ◆ Benefits of Programmable Logic

- Faster to Market
- Low Risk
- No NRE
- No Re-Spin Cost
- Short Lead Times
- Low Inventory Cost



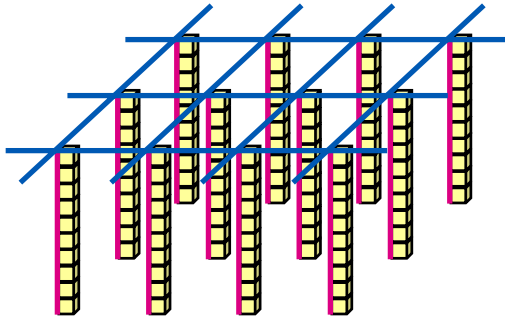
**FLEX 6000 Provides Low-Cost Flexibility**

Source: Dataquest/Altera  
Mid-1999 Price Projections

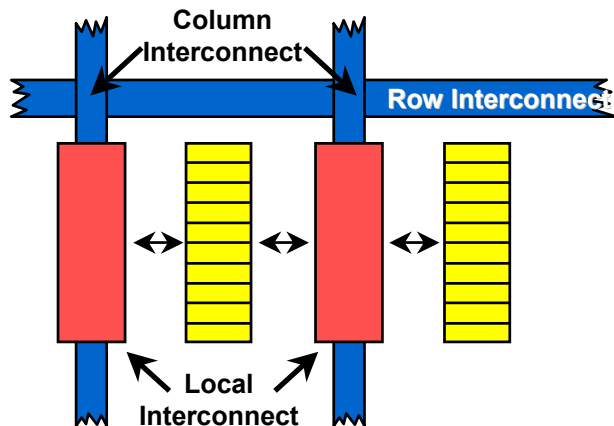


# Appendix: FLEX 6000 Architecture

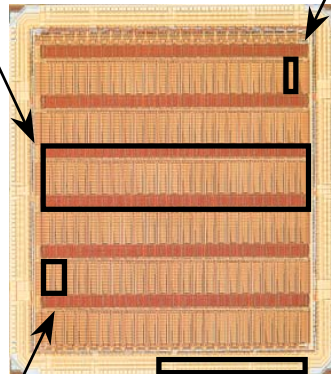
## FastTrack™ Interconnect



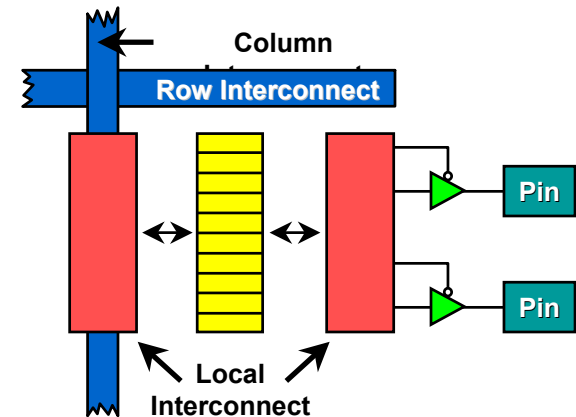
## Interleaved LABs



## FLEX 6000 Die



## FastFLEX™ I/O



## μPitch™ Technology

3.2 mil (81 μm)



Bond Pads

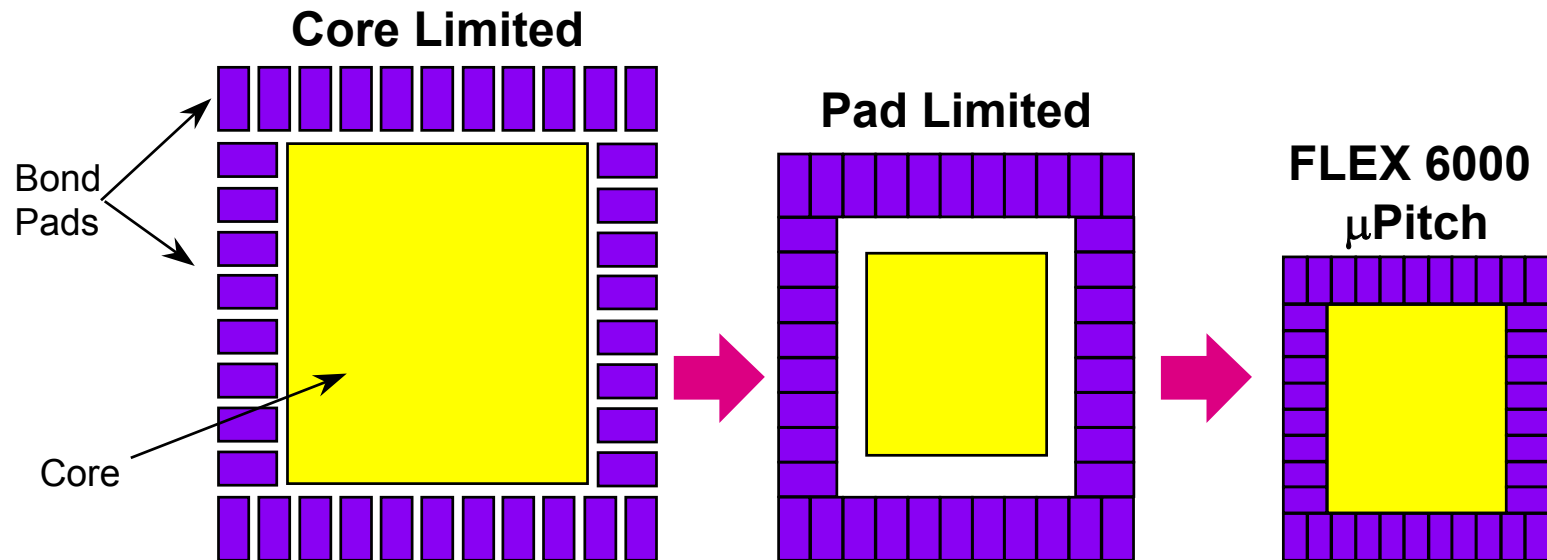


# What is $\mu$ Pitch Technology?

## ◆ OptiFLEX Leverages Most Advanced Bond Pad Pitch in the PLD Industry

- FLEX 6000 81 mm
- FLEX 6000A 75 mm

## ◆ Maximum Die Size Reduction for Lowest Possible Cost





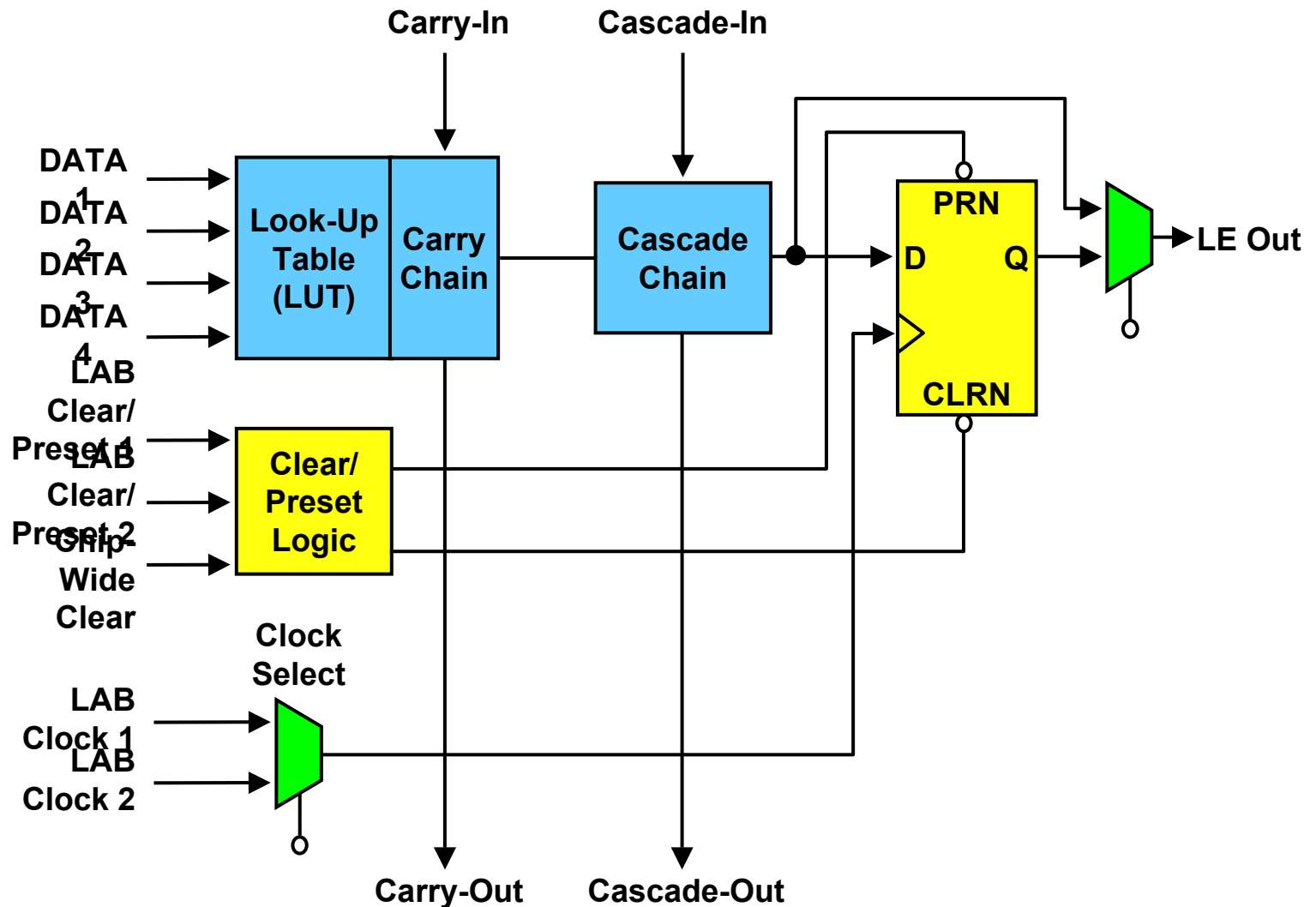
# FLEX 6000 Family

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

Now



# FLEX 6000 Logic Element





# **Low Power/ MultiVolt™ Design**



# Low-Power/MultiVolt Design

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



# 2.5-V Power Advantage

## ◆ 0.25- $\mu$ m Process Reduces Power by 54%

### ◆ Example

- 50-MHz Design Uses 821 mW in EPF10K30A Device
- Uses 379 *mW* in EPF10K30E Device

### ◆ Benefits

- Smaller Power Supply
- Simpler Cooling System
- Less Heat Buildup



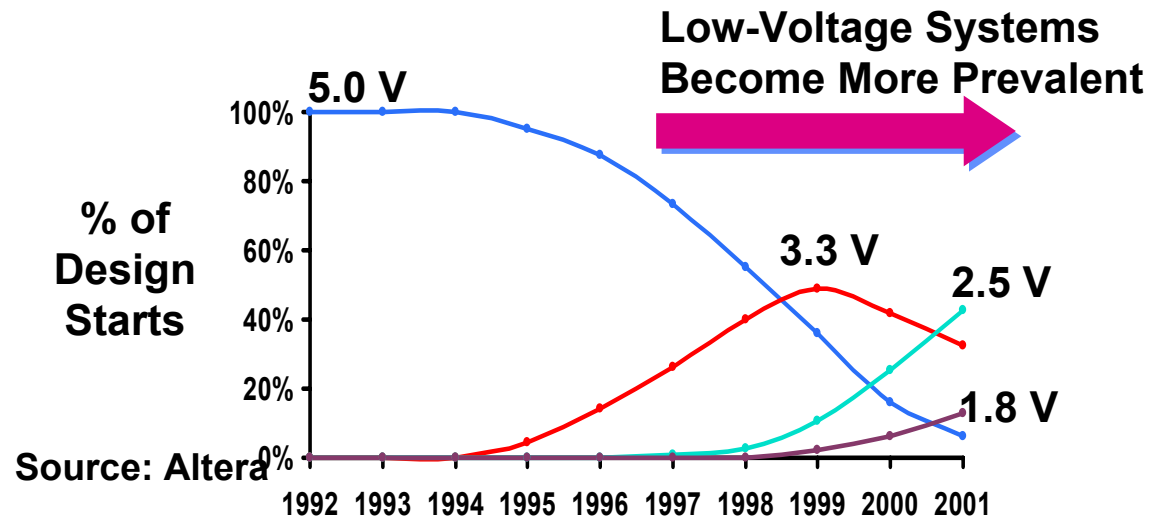
# Designing for 2.5-V Power Supply

## ◆ 2.5-V Devices Becoming Common

- Memory, Microprocessors

## ◆ What if FLEX 10KE Device Is Only 2.5-V Device?

- Generate 2.5-V Supply from 3.3-V or 5.0-V Supply





# Voltage Regulator Options

## ◆ Use Voltage Regulator to Generate 2.5-V Supply

- Linear
- Switching

## ◆ Selecting a Voltage Regulator

## ◆ Example: EPF10K200E Design



# Linear Regulators

## ◆ Advantages

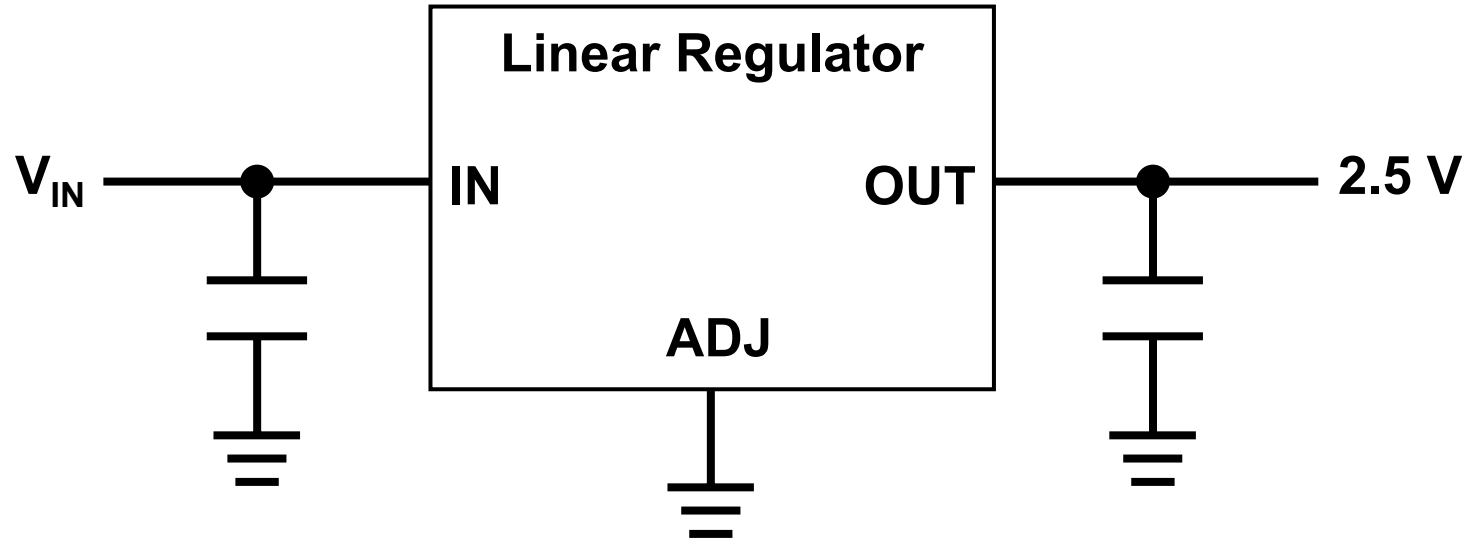
- Component Count
- Cost
- Board Area
- No EMI Radiation
- Tight Voltage Regulation

## ◆ Disadvantages

- Efficiency
- Power Dissipation



# Linear Regulator





# Switching Regulators

## ◆ Advantages

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

## ◆ Disadvantages

- EMI
- More Complex
- Component Count
- Board Area
- Cost



# EPF10K200E Design Example

Design Requirement	Value
VCCIO Supply Level	3.3 V
VCCINT Supply Level	2.5 V
$f_{\text{MAX}}$	100 MHz
Output Pins	350
Utilization	100%
Supply Voltages Available on the Board	3.3 V, 5.0 V

- ◆ Calculated  $I_{\text{OUT}} = 6.3 \text{ A}$
- ◆ Linear Solution (LT1580CT)
  - Efficiency = 75%
- ◆ Switching Solution (LTC1649)
  - Efficiency  $\geq 90 \%$



# Interfacing 2.5-V PLD to System

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

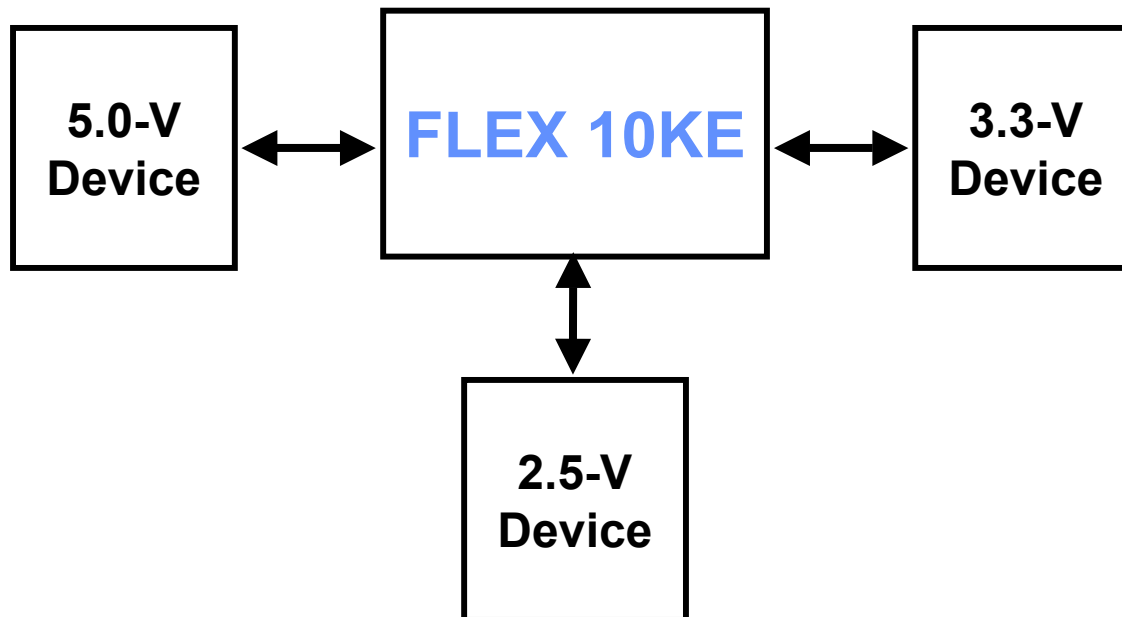
***What's The Solution?***



# FLEX 10KE & Multi-Voltage Boards

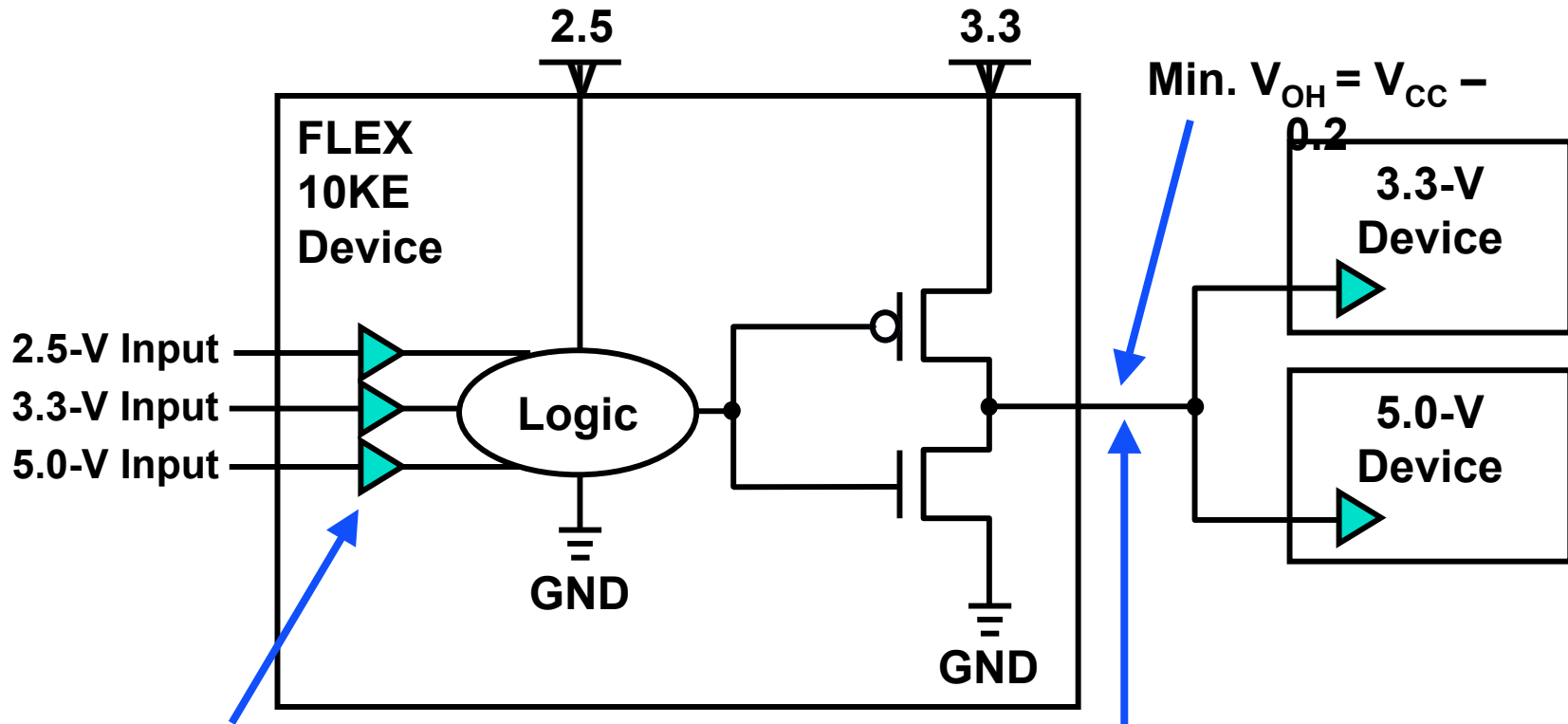
## ◆ FLEX 10KE Interfaces with Multiple Voltage Levels

- MultiVolt™ I/O Feature
- 2.5-V, 3.3-V, 5.0-V I/O
- 3.3-V PCI





## 3.3-V I/O with 2.5-V Logic

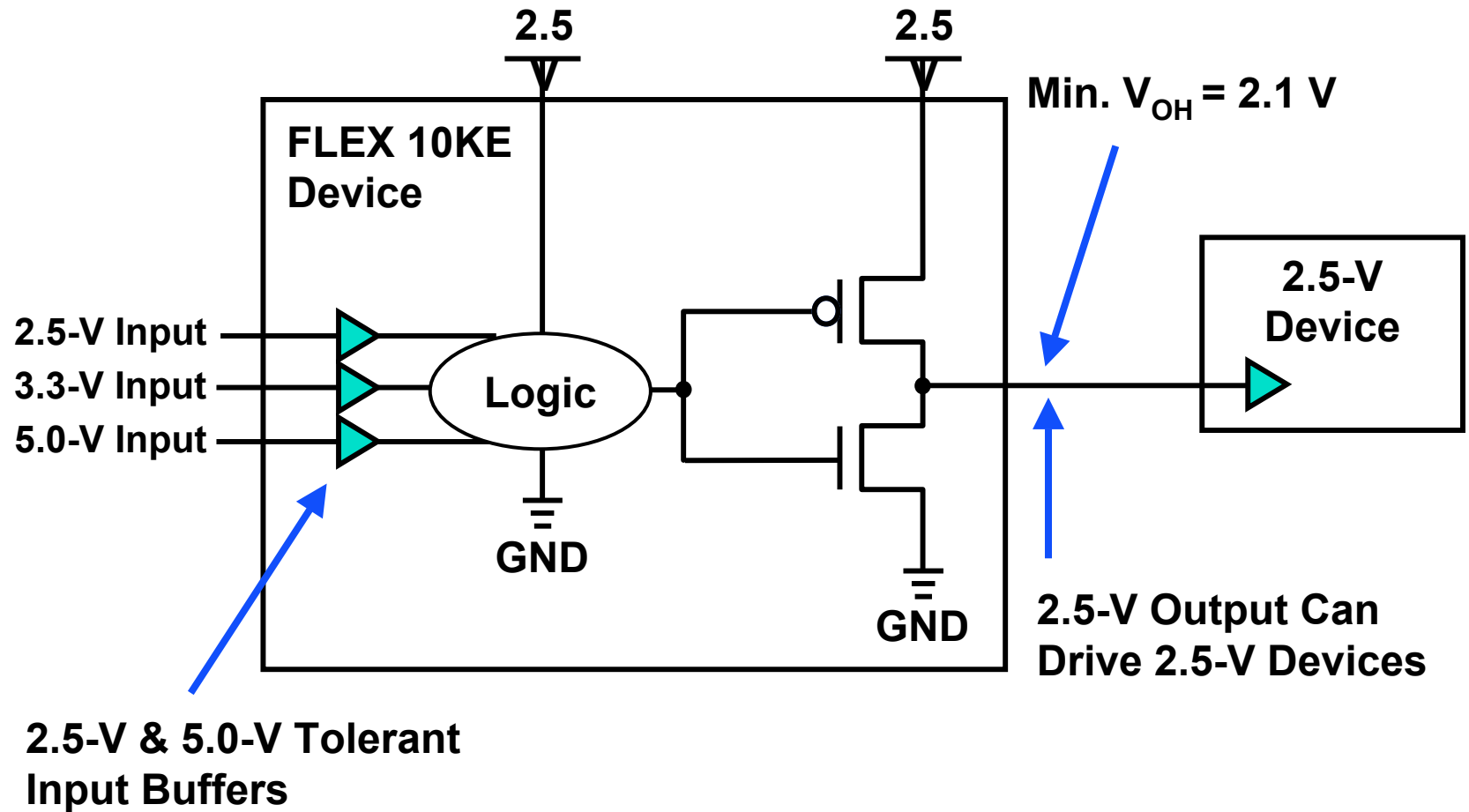


## 2.5-V & 5.0-V Tolerant Input Buffers

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



# 2.5-V I/O with 2.5-V Logic

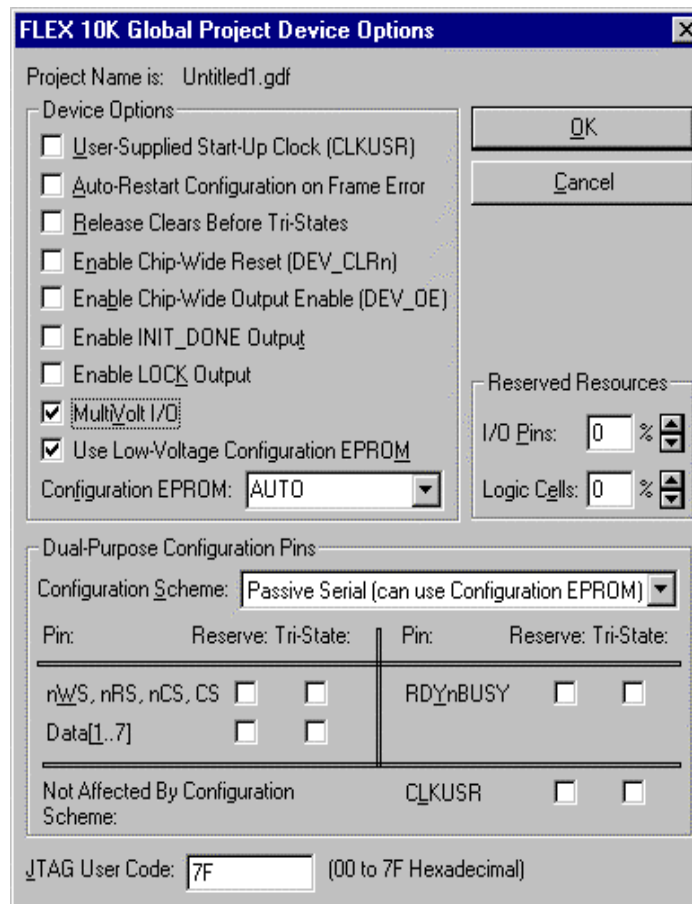




# Simulating Timing of MultiVolt I/O

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

Turn on MultiVolt™  
I/O Setting when  
VCCIO Is Not Equal  
to VCCINT



**FLEX 10K Global Project Device Options**

Project Name is: Untitled1.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
- ☒ MultiVolt I/O
- ☒ Use Low-Voltage Configuration EPROM

Configuration EPROM: AUTO

Reserved Resources:

I/O Pins: 0 %

Logic Cells: 0 %

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 ☐ ☐

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



# FLEX 10KE MultiVolt I/O

## Summary

### ◆ Separate VCC Pins for Logic & I/O Pins

- Logic Driven by VCCINT
- I/O Pins Driven by VCCIO

### ◆ Connect VCCINT to 2.5-V Supply

### ◆ Connect VCCIO to 2.5-V or 3.3-V Supply

VCCINT	VCCIO	Drives			Driven by		
		2.5 V	3.3 V	5.0 V	2.5 V	3.3 V	5.0 V
2.5 V	3.3 V	Ok	Ok	Ok	Ok	Ok	Ok
2.5 V	2.5 V	Ok			Ok	Ok	Ok



# Altera's Multivolt Offering

Device	Technology	VccINT	VccIO	Drives(TTL)			Driven by		
				2.5	3.3	5	2.5	3.3	5
FLEX6000	0.5	5	5			Y		Y	Y
			3.3		Y	Y		Y	Y
FLEX6000A	0.35	3.3	3.3		Y	Y	Y	Y	Y
			2.5	Y			Y	Y	Y
FLEX8000A	.6,.5	5	5	Y		Y			Y
	.6,.5	5	5			Y		Y	Y
			3.3		Y	Y		Y	Y
EPF8282AV	.6,.5	3.3	3.3		Y	Y		Y	
FLEX10K	0.5	5	5			Y		Y	Y
			3.3		Y	Y	Y	Y	Y
FLEX10A	0.35	3.3	3.3		Y	Y	Y	Y	Y
			2.5	Y				Y	Y
EPF10KE	0.22	2.5	3.3		Y	Y	Y	Y	Y
			2.5	Y			Y	Y	Y



# Device feature summary

	3.3V	ISP	ICR	EAB	Open Drain	GCLK	Dedicated Input	OE
FLEX10K(A)	Y		Y	Y	Y	2(+4)	4	ALL
FLEX8K	Y		Y			(+4)	4	10
FLEX6K(A)	Y		Y			(+4)	4	ALL
MAX9000		Y				2		8
MAX7000S	Y	Y			Y	2		6

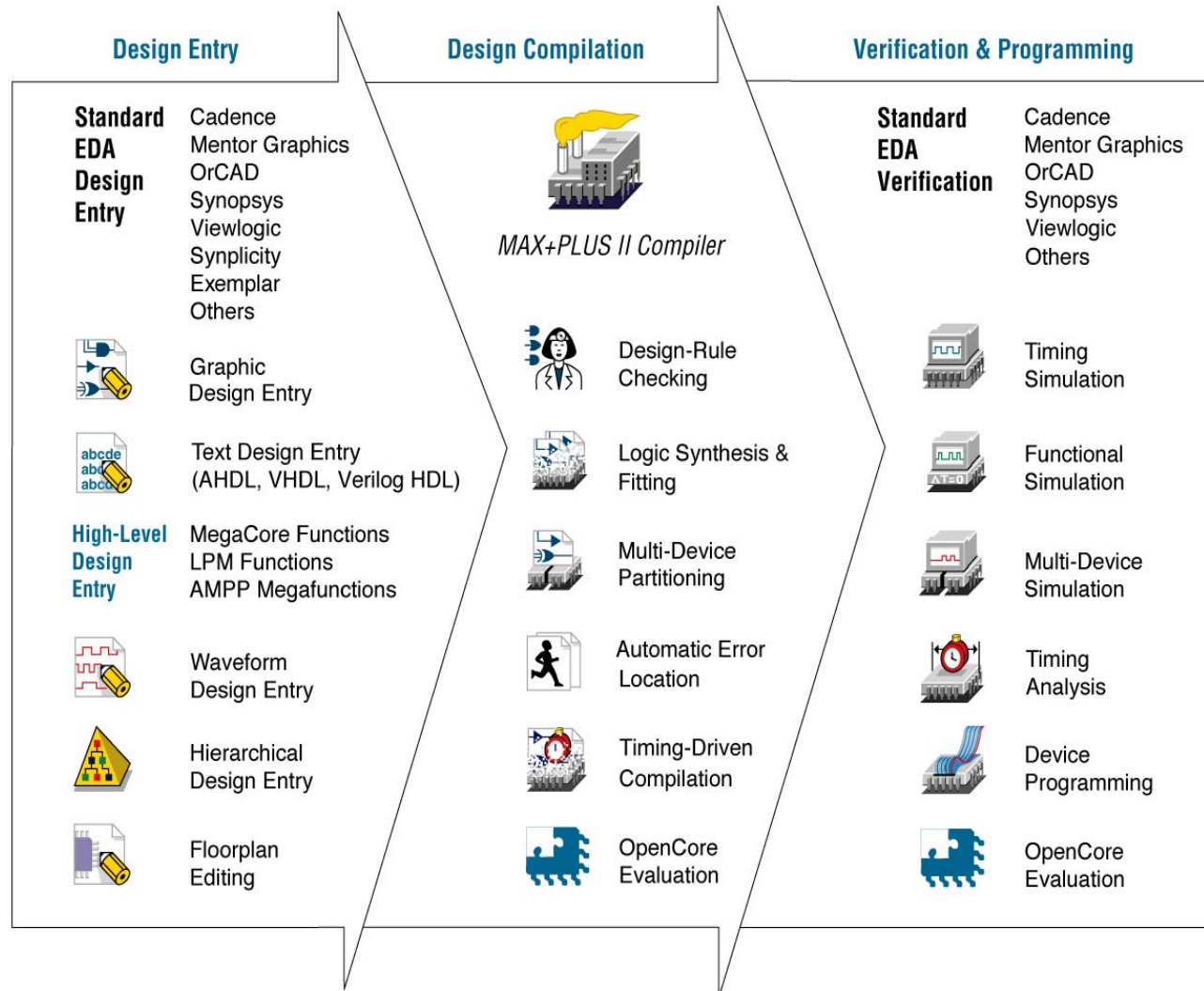
## Notes :

- Multi-Volt ; PCI ; Slew slow rate ; JTAG-BST ==> All Devices
- PLL = Phase Locked Loop (ClockBoost)
- ISP = In System Programmability
- ICR = In Circuit Reconfiguration
- OE = Output Enable



# MAX+PLUS II Can ... (An Introduction)

## ◆ Operate in a self-contained environment





# MAX+PLUS II Software Products

## ◆ Fixed-Node Subscription Products

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

## ◆ Floating-Node Subscription Products

- Licensed Using Windows NT and UNIX Servers
- **FLOATPC** for Windows 95/98 and Windows NT clients only.
- **FLOATNET** for Windows 95/98/NT and UNIX clients.

## ◆ MAX+PLUS II **BASELINE** Software

- entry-level version of the MAX+PLUS II software which is free of charge.



# MAX+plus II

## ◆ Supported Platforms\*

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

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

## ◆ Network licensing supported on both PC and Unix



# Design Flow & Altera Tools

## ◆ FPGA/CPLD Design Flow

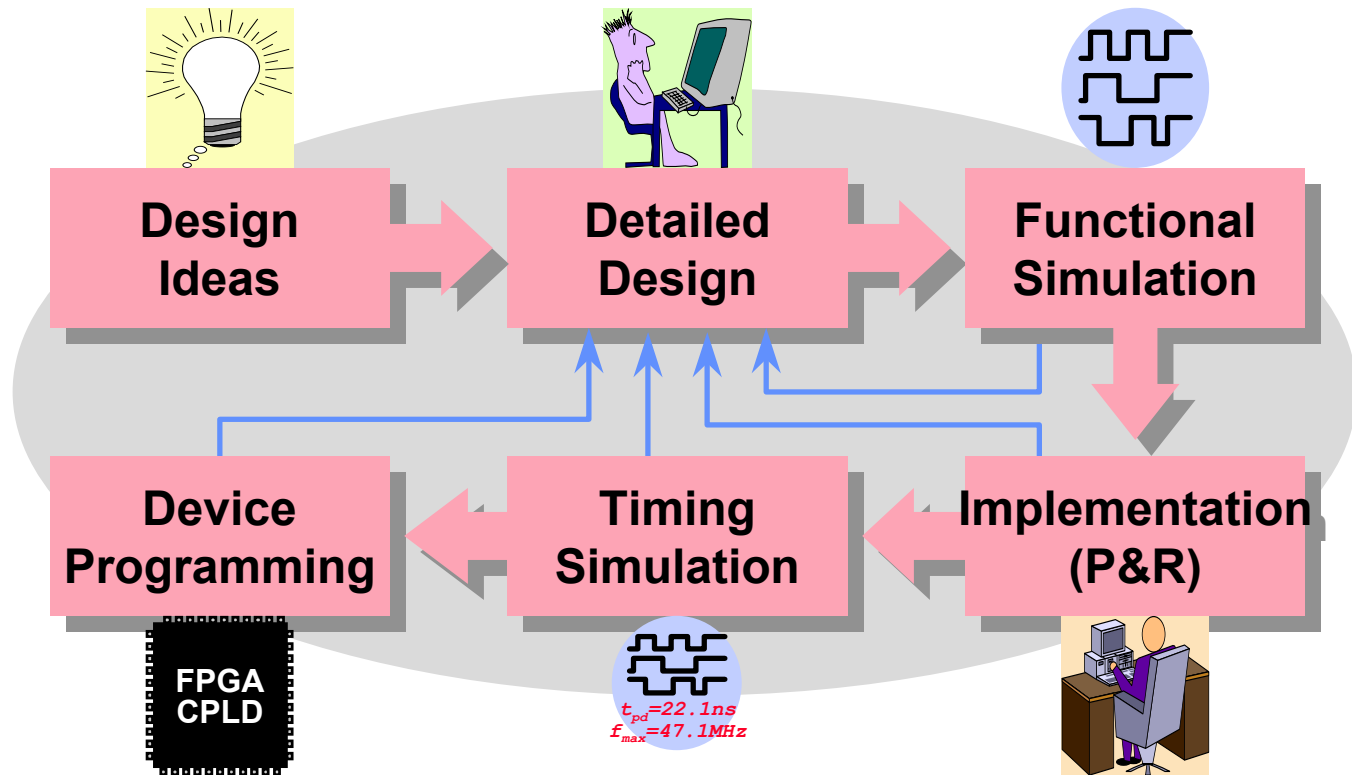
- Design Ideas
- Detailed Design
- Functional Simulation
- Synthesis & Implementation
- Timing Simulation
- Device Programming

## ◆ Altera MAX+PLUS II Development Software

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



# FPGA/CPLD Design Flow





# Design Ideas



## ◆ What are the main design considerations?

- Design feasibility?
- Design spec?
- Cost?
- FPGA/CPLD or ASIC?
- Which FPGA/CPLD vendor?
- Which device family?
- Development time?



# Detailed Design



## ◆ Choose the design entry method

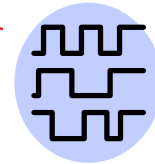
- Schematic
  - Gate level design
  - Intuitive & easy to debug
- HDL (Hardware Description Language), e.g. Verilog & VHDL
  - Descriptive & portable
  - Easy to modify
- Mixed HDL & schematic

## ◆ Manage the design hierarchy

- Design partitioning
  - Chip partitioning
  - Logic partitioning
- Use vendor-supplied libraries or parameterized libraries to reduce design time
- Create & manage user-created libraries (circuits)



# Functional Simulation



## ◆ Preparation for simulation

- Generate simulation patterns
  - Waveform entry
  - HDL testbench
- Generate simulation netlist

## ◆ Functional simulation

- To verify the functionality of your design only

## ◆ Simulation results

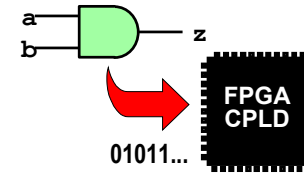
- Waveform display
- Text output

## ◆ Challenge

- Sufficient & efficient test patterns



# Design Implementation



## ◆ Implementation flow

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

## ◆ Implementation results

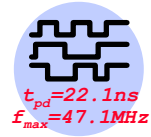
- Design error or warnings
- Device utilization
- Timing reports

## ◆ Challenge

- How to reach high performance & high utilization implementation?



# Timing Analysis & Simulation



## ◆ Timing analysis

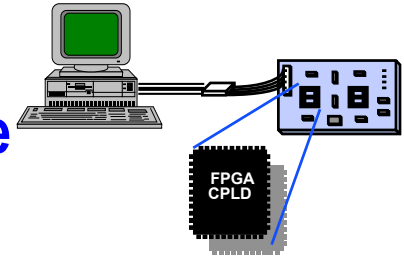
- Timing analysis is static, i.e., independent of input & output patterns
- To examine the timing constraints
- To show the detailed timing paths
- Can find the critical path

## ◆ Timing simulation

- To verify both the functionality & timing of the design



# Device Programming



## ◆ Choose the appropriate configuration scheme

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

## ◆ Finish the board design

## ◆ Program the device

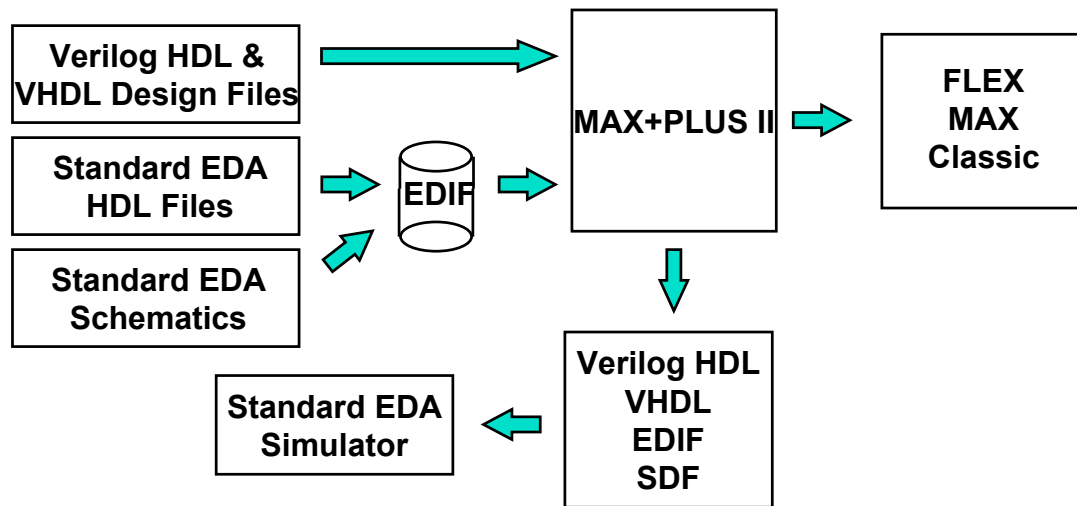
## ◆ Challenge

- Board design
- System considerations



# Altera Design Flow

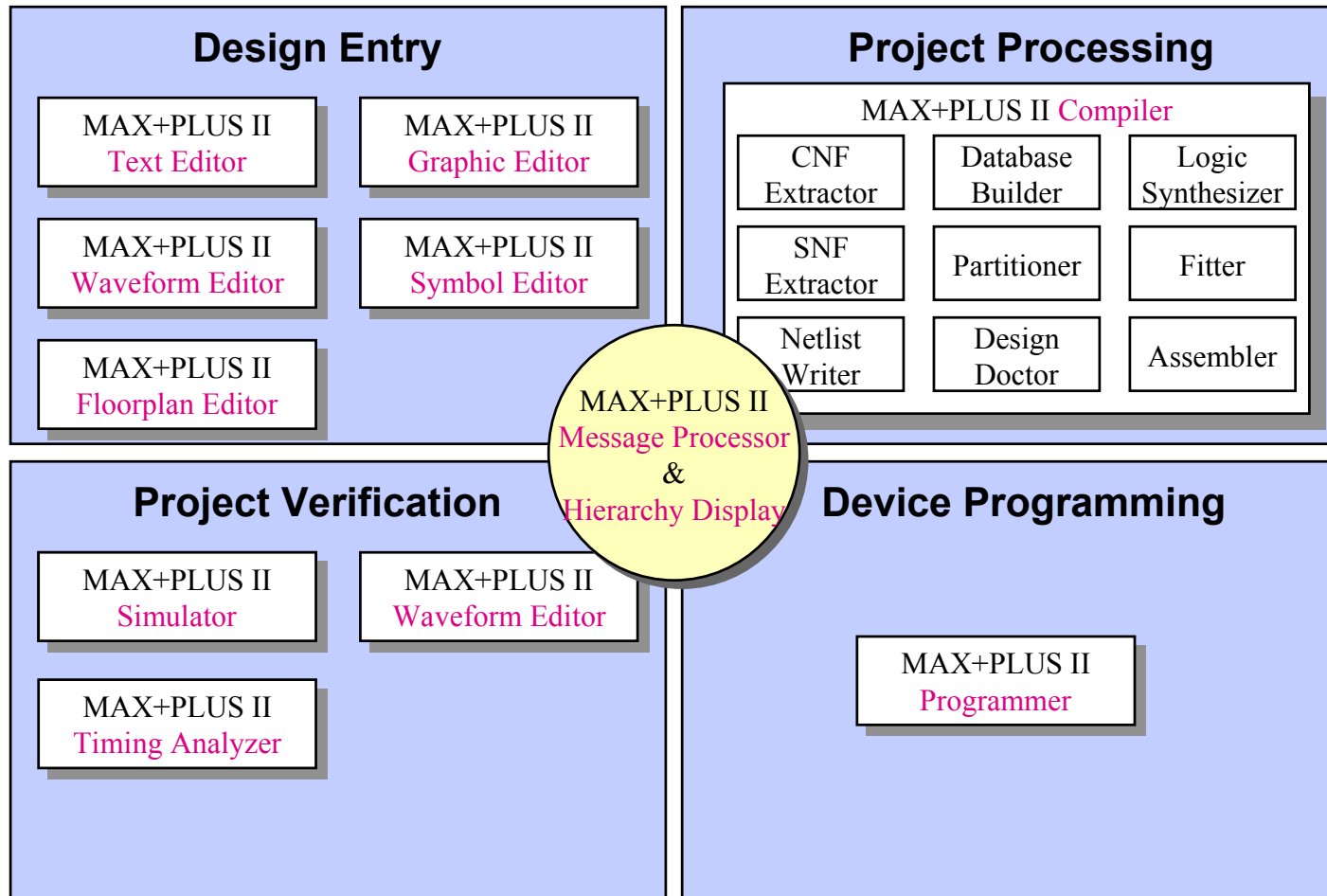
◆ Operate seamlessly with other EDA tools





# MAX+PLUS II

## Altera's Fully-Integrated Development System





# Design Entry

## ◆ MAX+PLUS II 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 interface with four windows open:

- 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 **get\_ticket**.
- speed\_ch.wdf - Waveform Editor:** Shows a timing diagram for signals **accel\_in**, **reset**, **clk**, **speed**, and **get\_ticket**. The **speed** signal is labeled **legal** and has a duration of **600.0ns**. The **get\_ticket** signal is labeled **0**.
- 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:** Shows the VHDL code for the **time\_cnt** component.

```
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







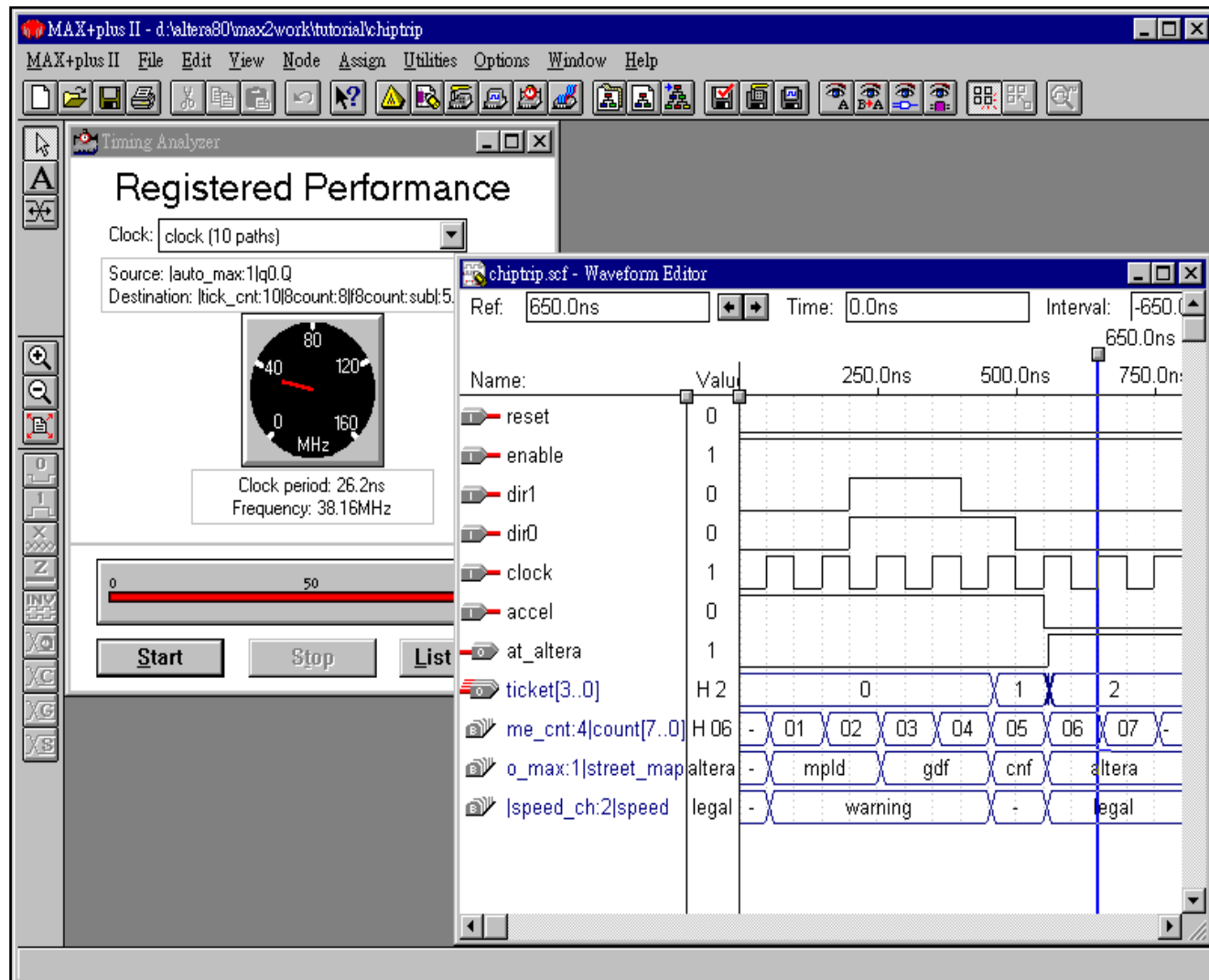
# Project Verification

## ◆ MAX+PLUS II tools for project verification

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



# MAX+PLUS II Project Verification

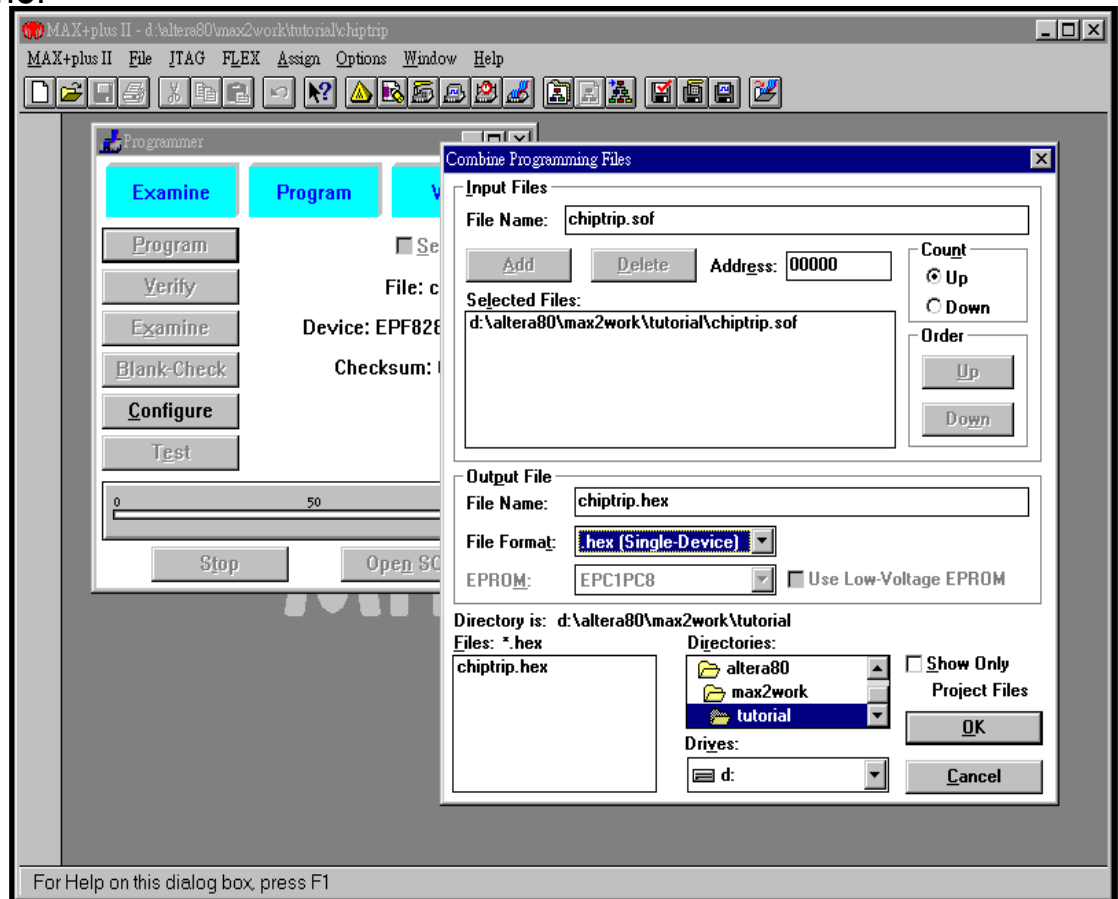




# Device Programming

## ◆ MAX+PLUS tool for device programming

- MAX+PLUS II Programmer





# MAX+PLUS II Features

## ◆ MAX+PLUS II, Altera's fully integrated design environment

- Schematic, text (AHDL), waveform design entry & hierarchy display
- Floorplan editing
- DRC, logic synthesis & fitting, timing-driven compilation
- Multi-device partitioning
- Automatic error location
- Functional simulation, timing simulation, and multi-device simulation
- Timing analysis
- Programming file generation & device programming
- EDA interface : industry-standard library support, EDA design entry & output formats (EDIF, Verilog & VHDL)
- [On-line help](#)



# Getting Started

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



# System Requirements

## ◆ The minimum system requirements

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

## ◆ Memory & disk space requirement

- Go to the [read.me](#) file for specific information about disk space & memory requirements in the current version of MAX+PLUS II
  - At least **64MB** physical RAM is recommended
  - Memory requirement depends on the selected device and the design complexity



# Installing MAX+PLUS II

## ◆ To install MAX+PLUS II from CD-ROM

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

## ◆ Additional Windows NT installation steps

- You must install *Sentinel driver* after running the install program
  - To detect the key-pro
- (Optional) ByteBlaster and Altera LP6 Logic Programmer Card drivers
  - Required only for ByteBlaster or LP6 users



# Starting MAX+PLUS II

## ◆ To start MAX+PLUS II...

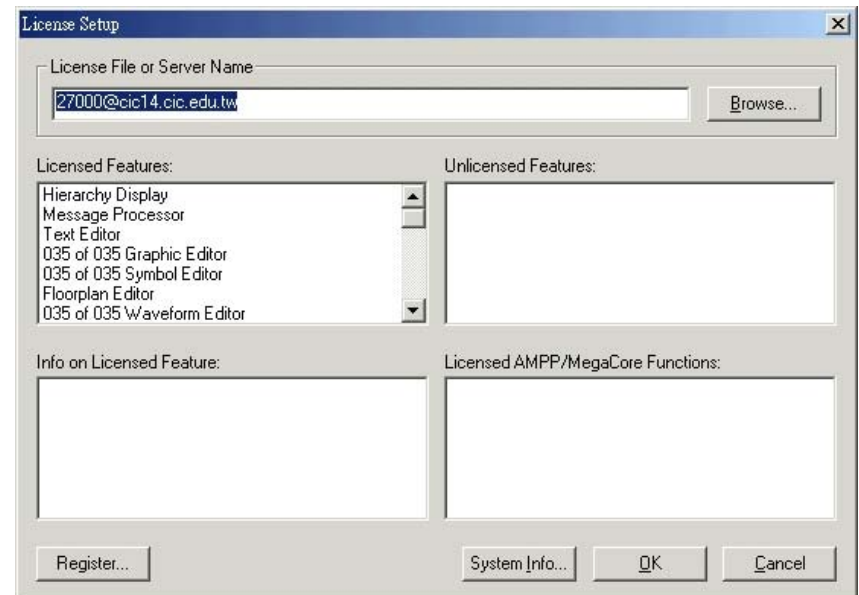
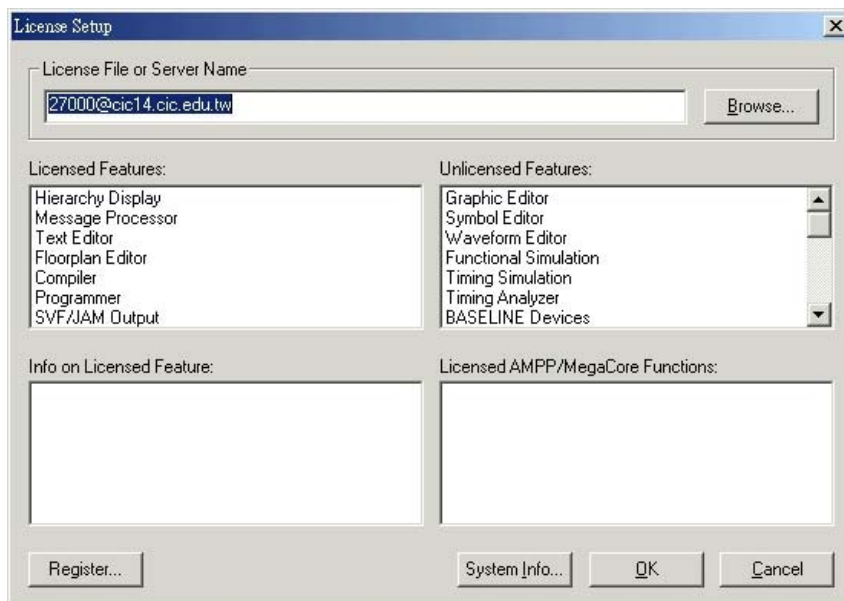
- Double click on the MAX+PLUS II icon





# Entering the Authorization Code

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





# MAX+PLUS II Operating Environment

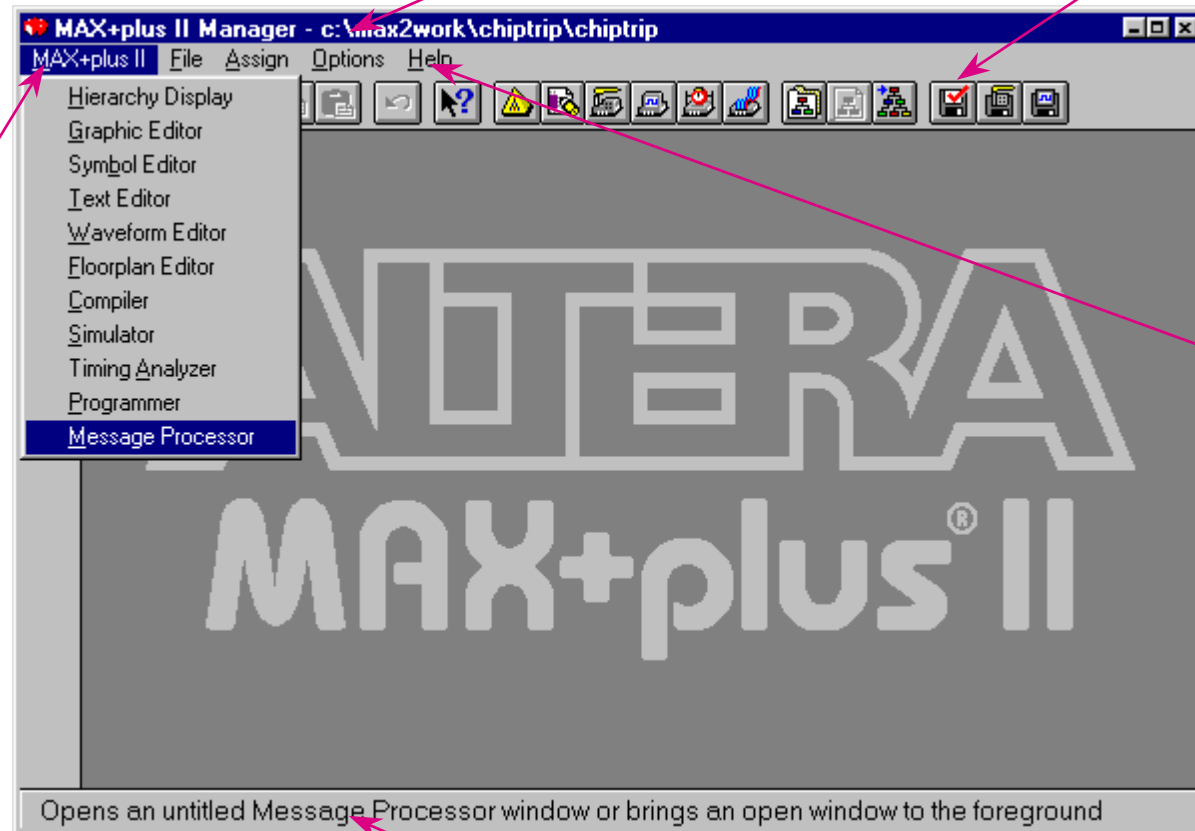
## ◆ MAX+PLUS II Manager

- Start-up window

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

Project Directory and Project name

Toolbar provides shortcuts for commonly used functions

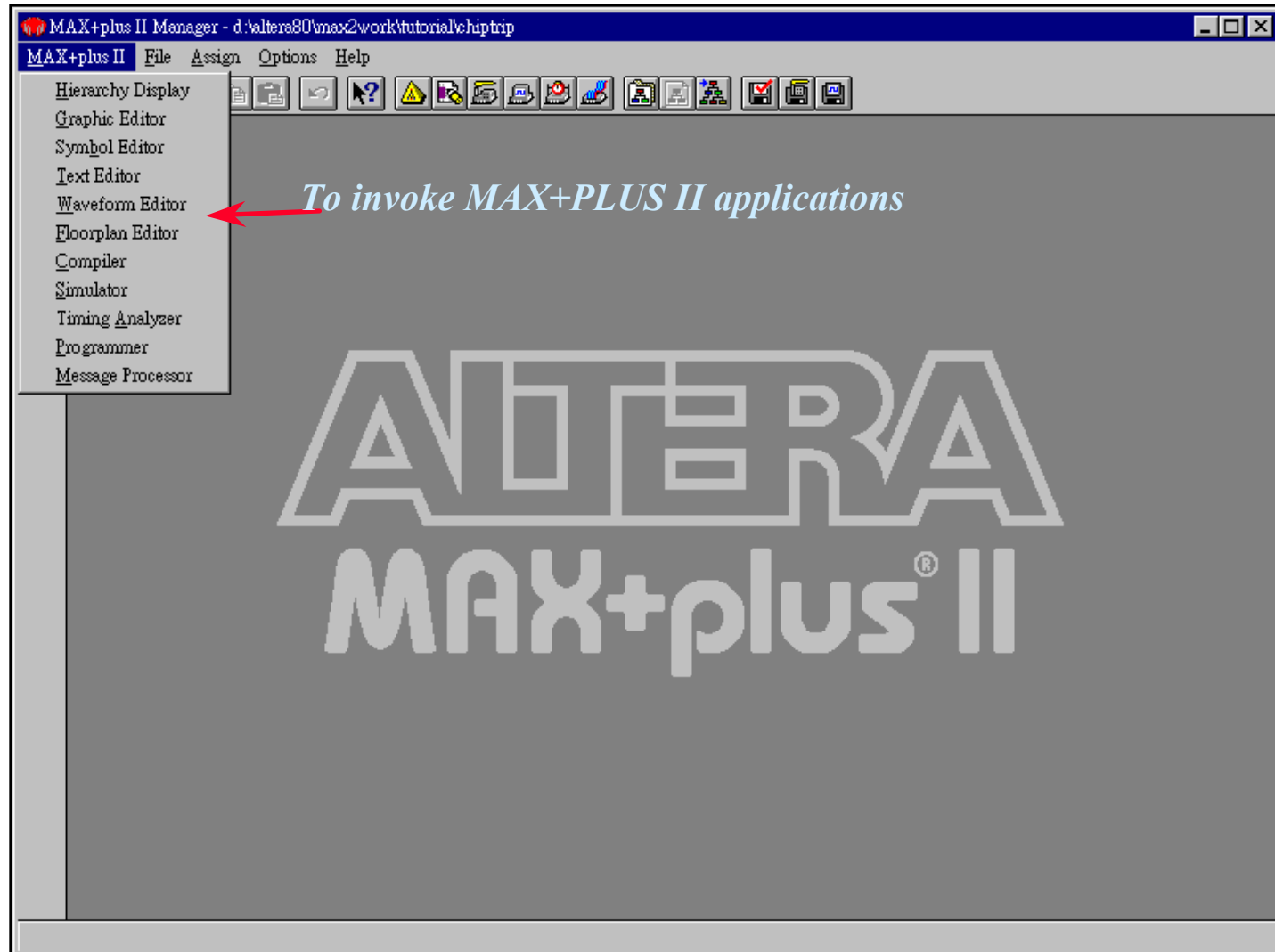


Help menu gives you access to on-line help

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

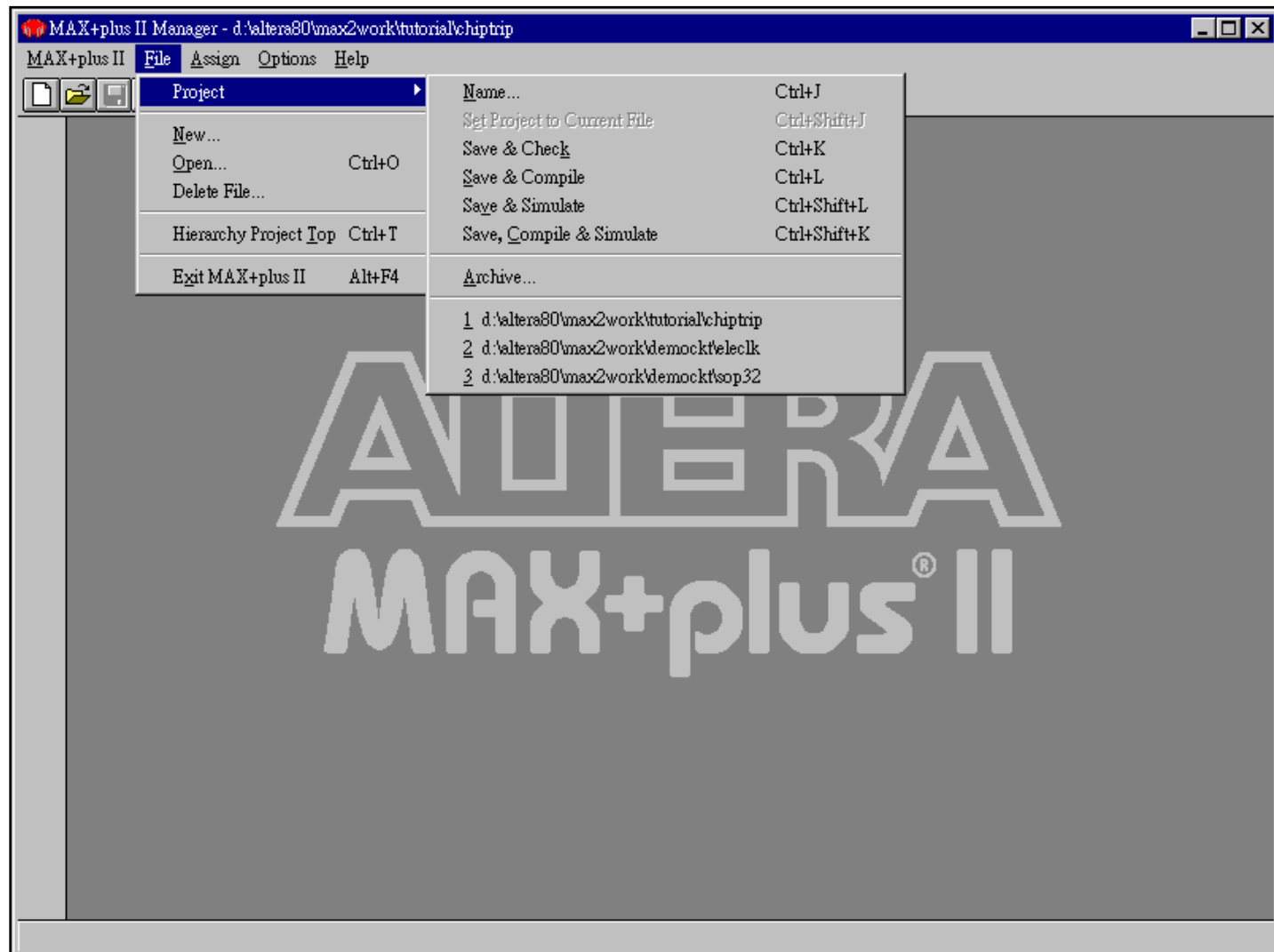


# MAX+PLUS II Menu



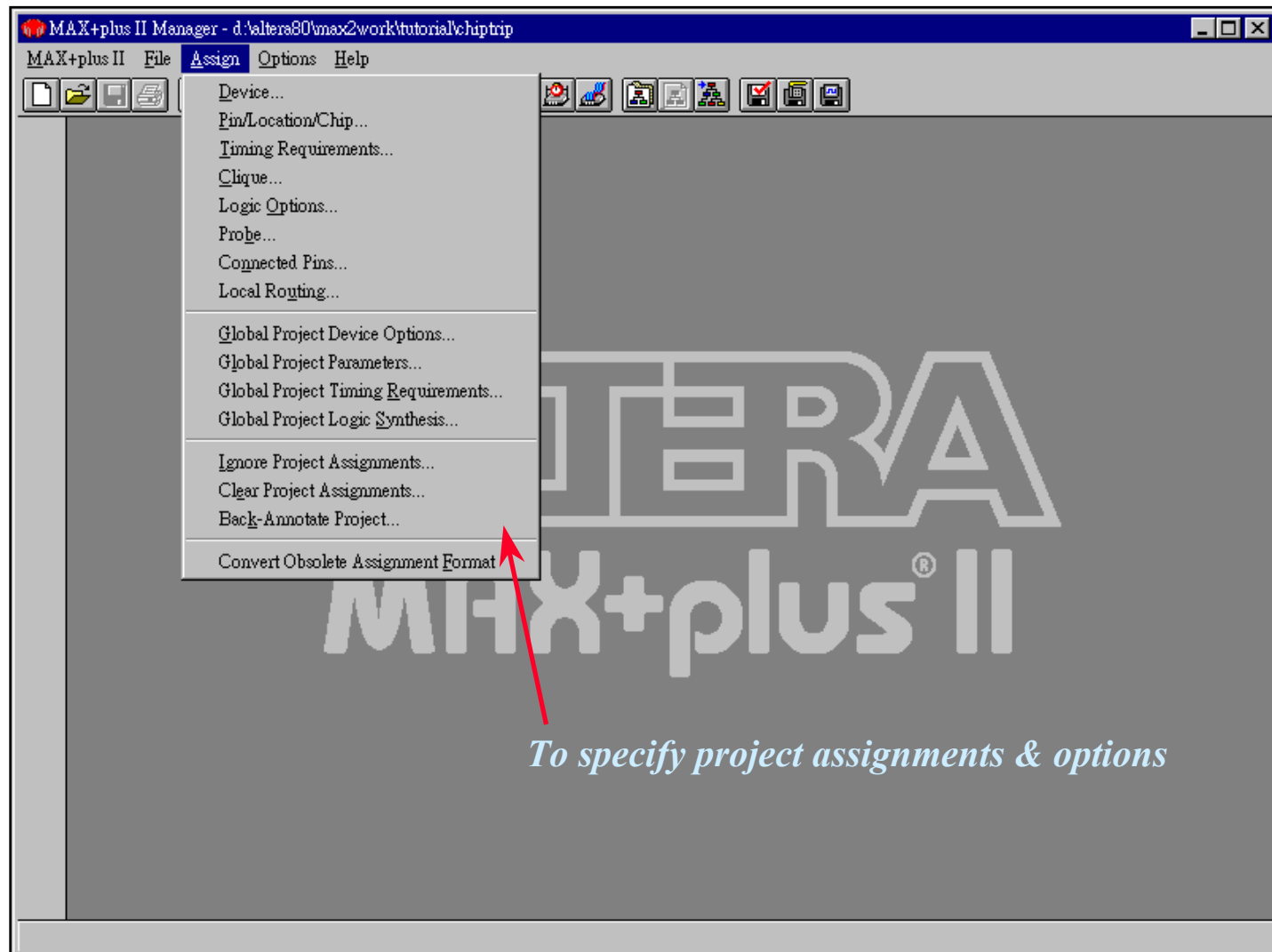


# File Menu



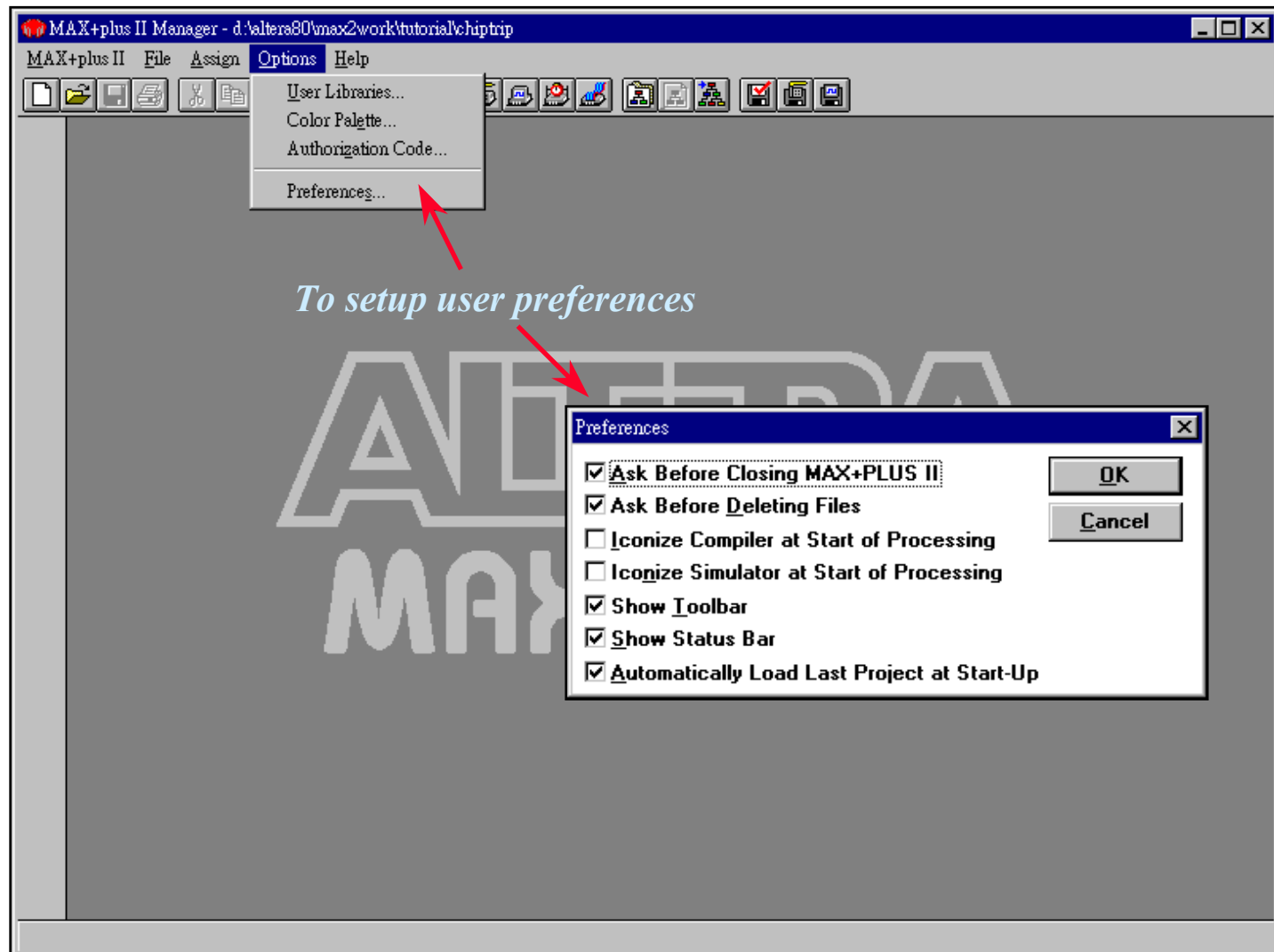


# Assign Menu



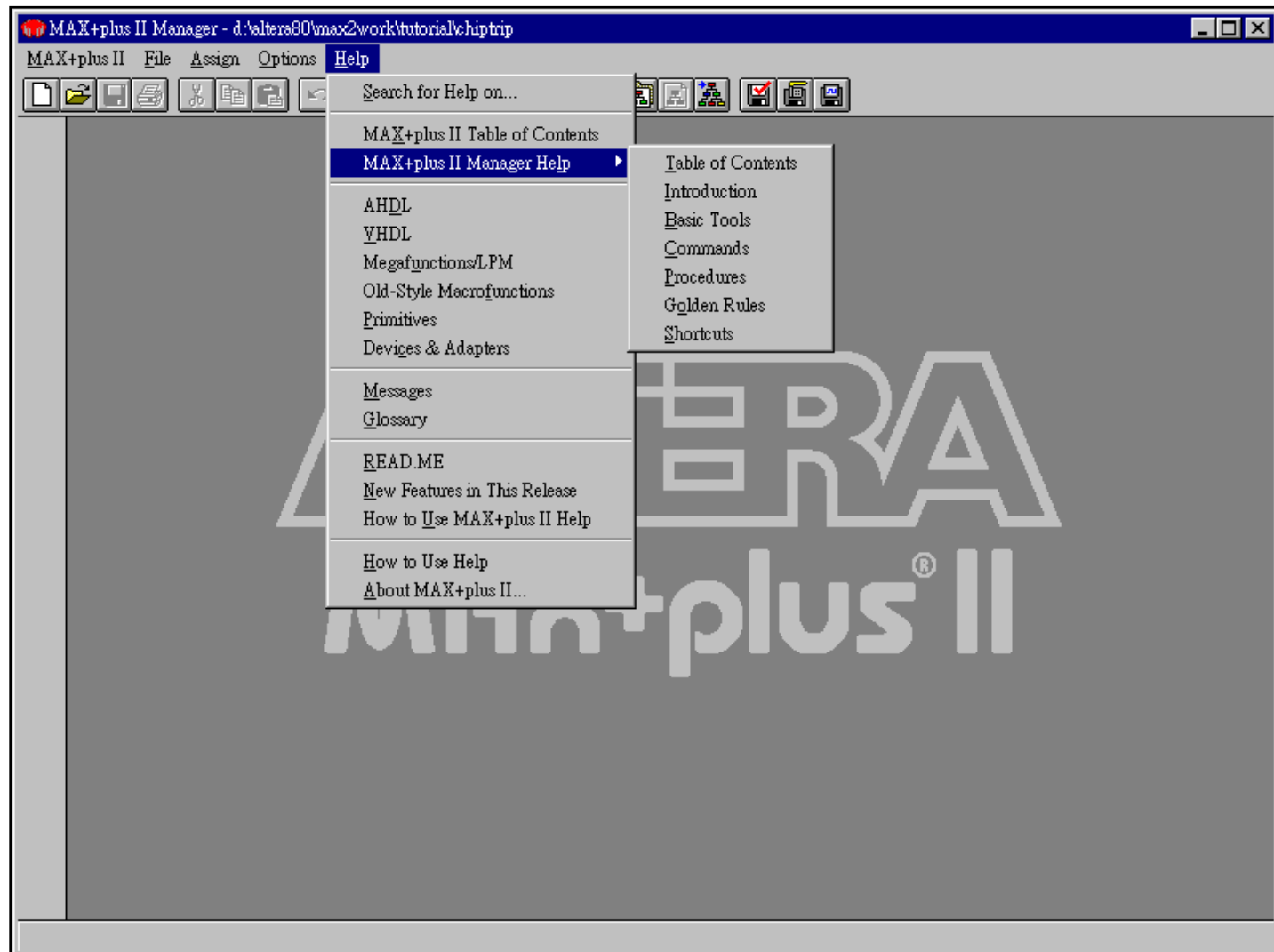


# Options Menu





# Help Menu

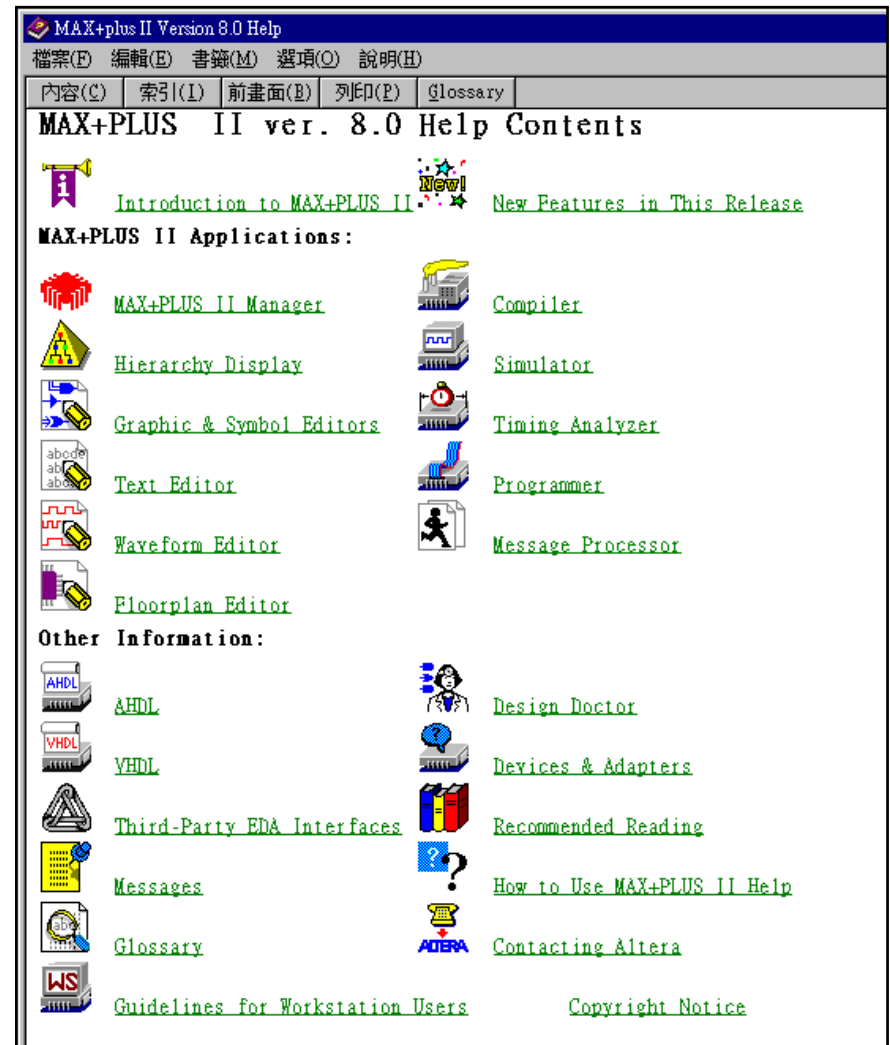




# MAX+PLUS II Help Contents

## ◆ 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

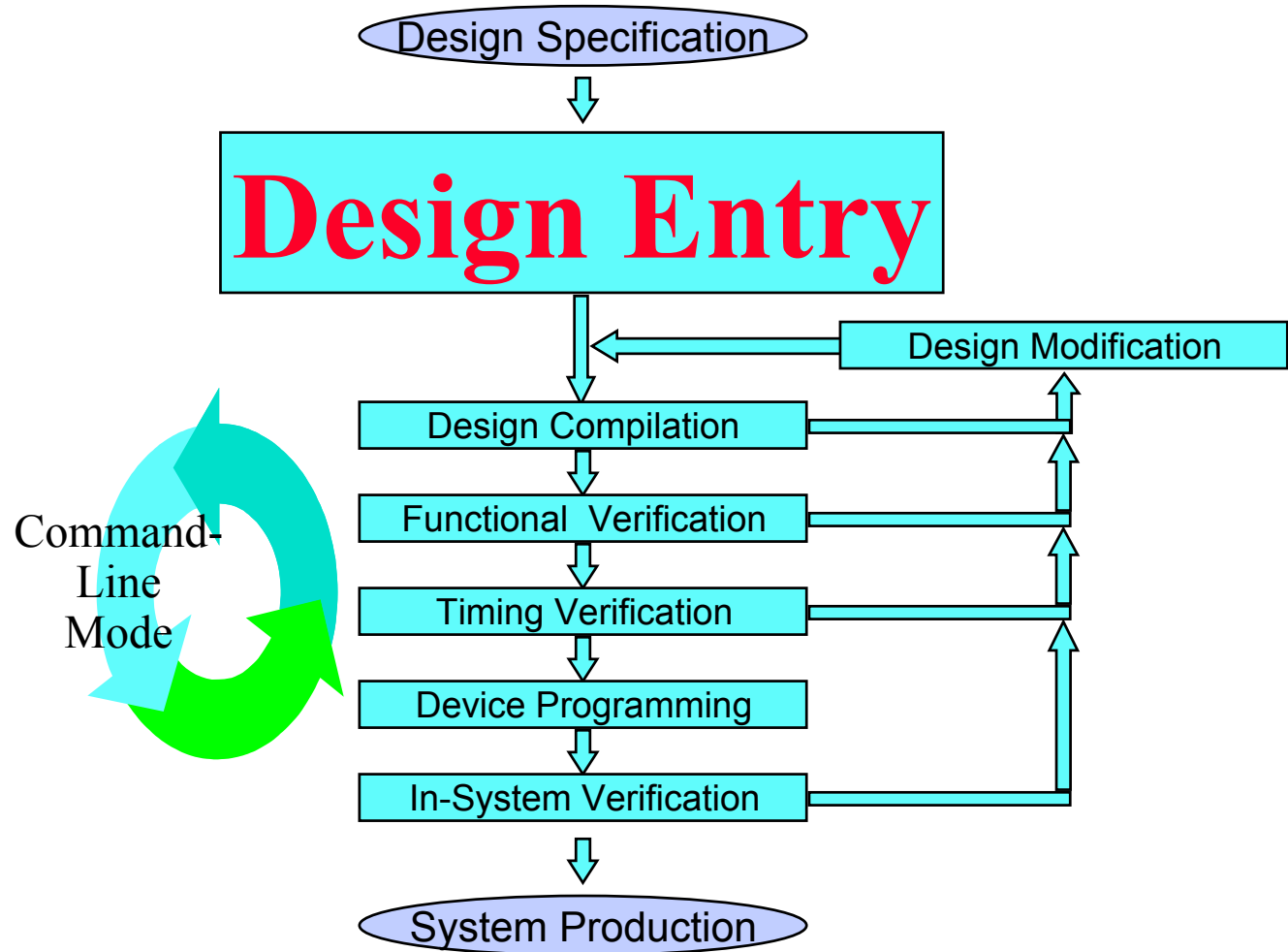




# **MAX+PLUS II**

## **Design Methodology**







# Design Entry Process

## ◆ Project Setup/Management

## ◆ Multiple design entry methods

- MAX+PLUS II
  - Graphic design entry
  - Text design entry
    - AHDL, VHDL, Verilog
  - Waveform design entry
- 3rd party EDA tools
  - EDIF, OrCAD schematics
- Add flexibility and optimization to the Design entry process by:
  - mixing and matching design files
  - using LPM and Megafunctions to accelerate design entry



# Project Setup/Management

## ◆ What is a Project?

- A design file
- A project is:
  - checked for design entry errors
  - compiled
  - simulated (functional or with timing)
  - analyzed for timing
  - used to generate programming file

## ◆ Projects can be archived

## ◆ To specify a project

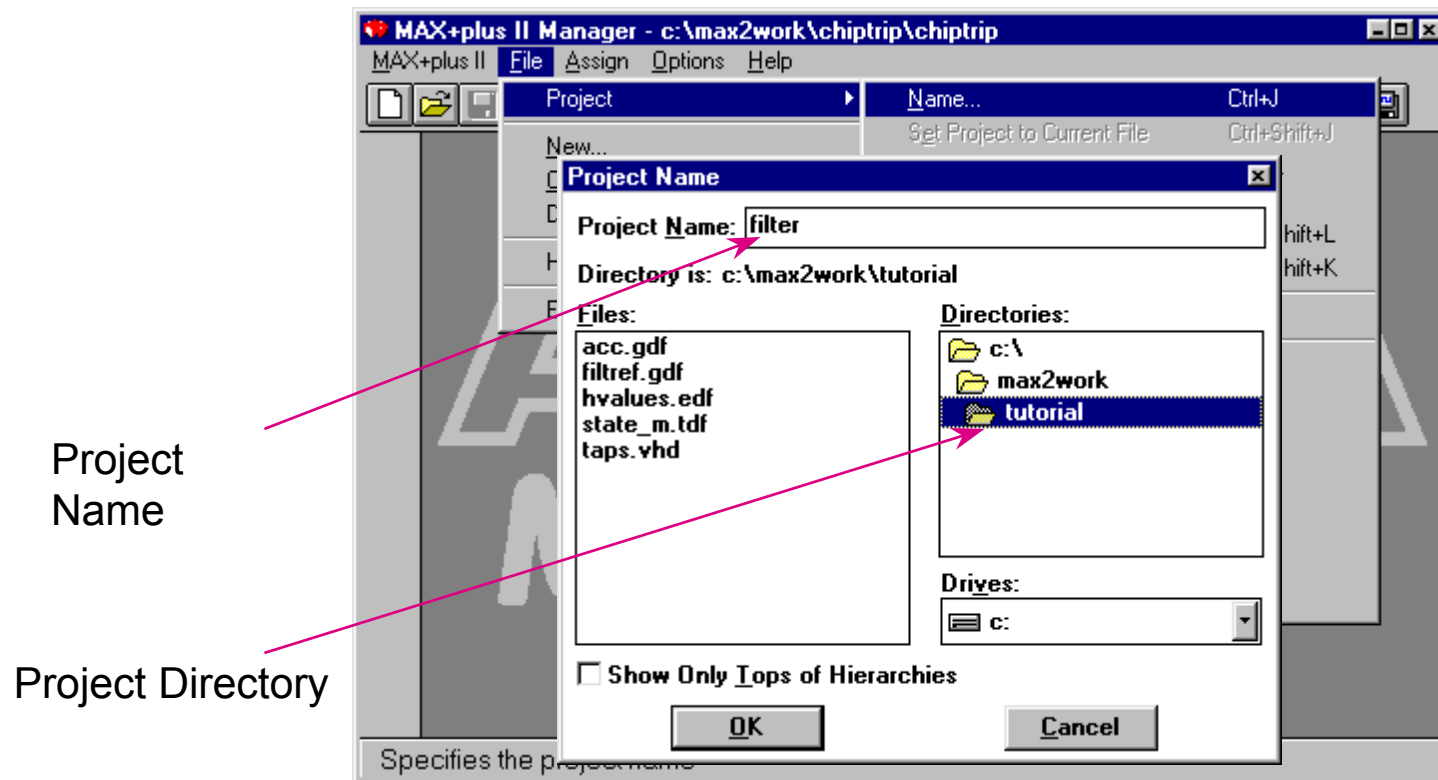
**Menu: File -> Project -> Name...** *(To specify an existing or new design file)*

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



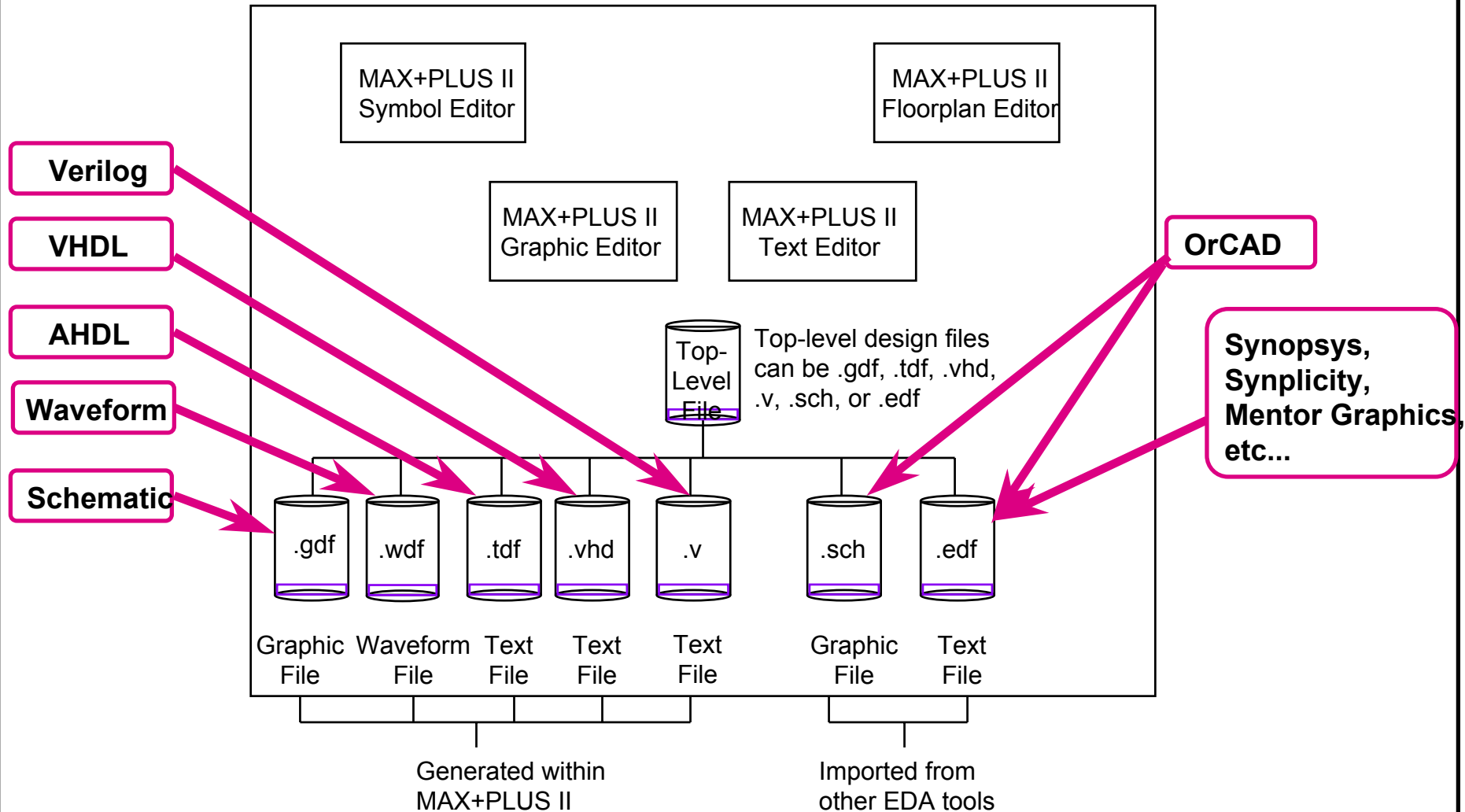
# Set Up A New Project

- ◆ Every design must have a project name
- ◆ Project name must match design file name





# Design Entry Files



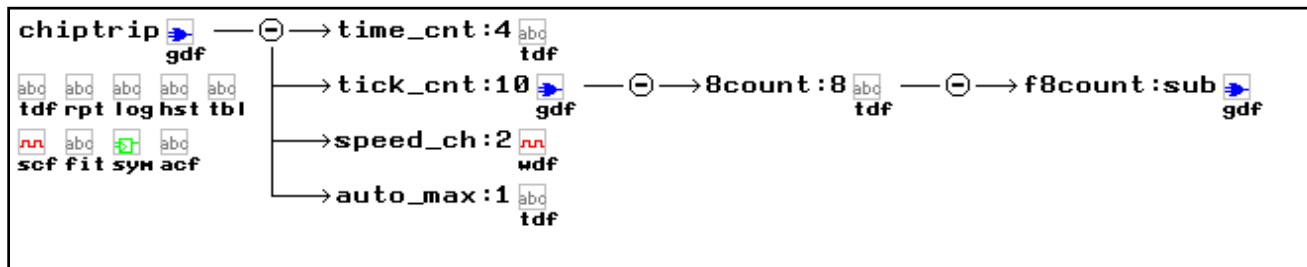


# Hierarchy Display

## ◆ MAX+PLUS II Hierarchy Display

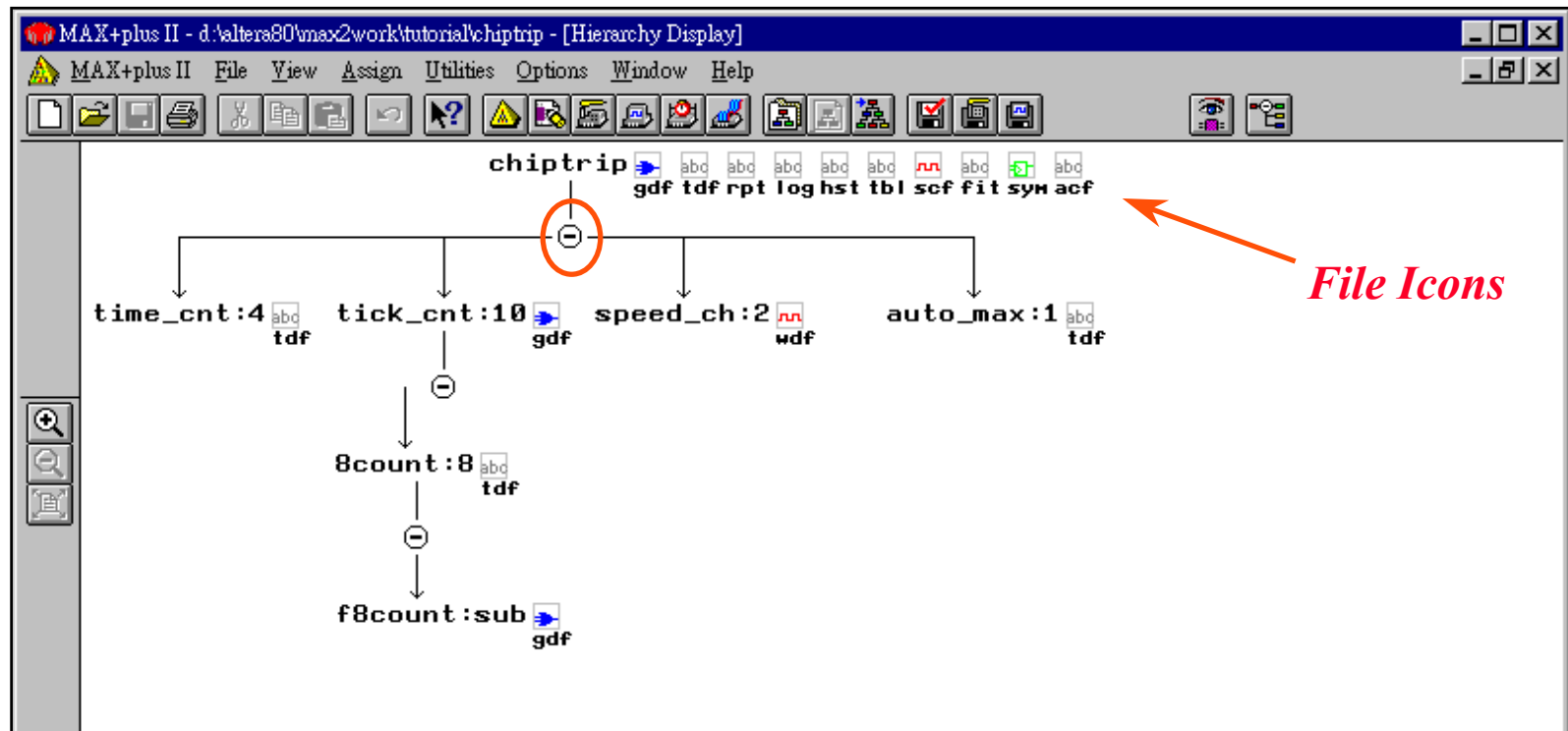
- The Hierarchy Display shows a hierarchy tree that represents the current hierarchy and allows you to open and close files in the hierarchy
- The hierarchy tree branches show a filename and file icon for each subdesign in the hierarchy, and it also shows ancillary files associated with the current hierarchy.
- 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



# Graphic Design Entry Process

## ◆ Add resource libraries to search list as needed

## ◆ Draw schematic

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

Note: **MAX+PLUS II DOES NOT AUTO SAVE**

## ◆ Save and check the design

- The file extension is .gdf
- Correct any errors with the aid of Message Processor

## ◆ Create symbol or include file for sub-design



# Resource Libraries

## ◆ **prim ( Altera primitives )**

- Basic logic building blocks

## ◆ **mf ( Macrofunction )**

- 7400 family logic

## ◆ **mega\_lpm ( LPMs )**

- Library of Parameterized Modules ( LPMs )
- Megafunctions are high level function module
  - busmux, ram elements, fifo's, etc...



# Value added Libraries

## ◆ MegaCores IP models you can try before purchase (download from [www.altera.com](http://www.altera.com))

- UARTs, FFT, PCI etc...

## ◆ AMPP ( Altera Megafunction Partners Program )

- Partners providing PCI, DSP,  $\mu$ Controllers, etc...

**Note: For the latest information on MegaCores or Megafunctions, refer to Altera's web site [www.altera.com](http://www.altera.com)**



# Add User Resource Libraries

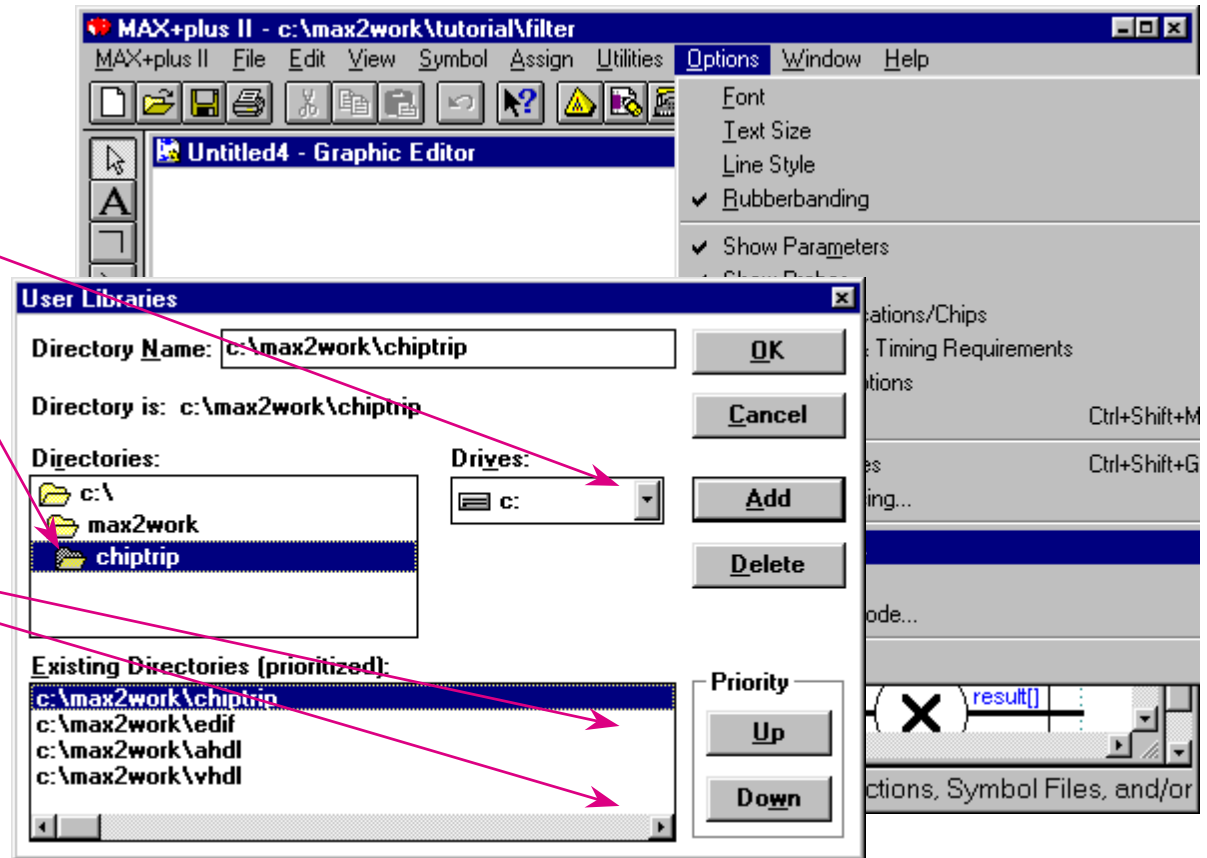
## ◆ Access user created libraries

- Add user library directories
- Set priorities

Select the library directory then click on Add

Library search priority can be changed.

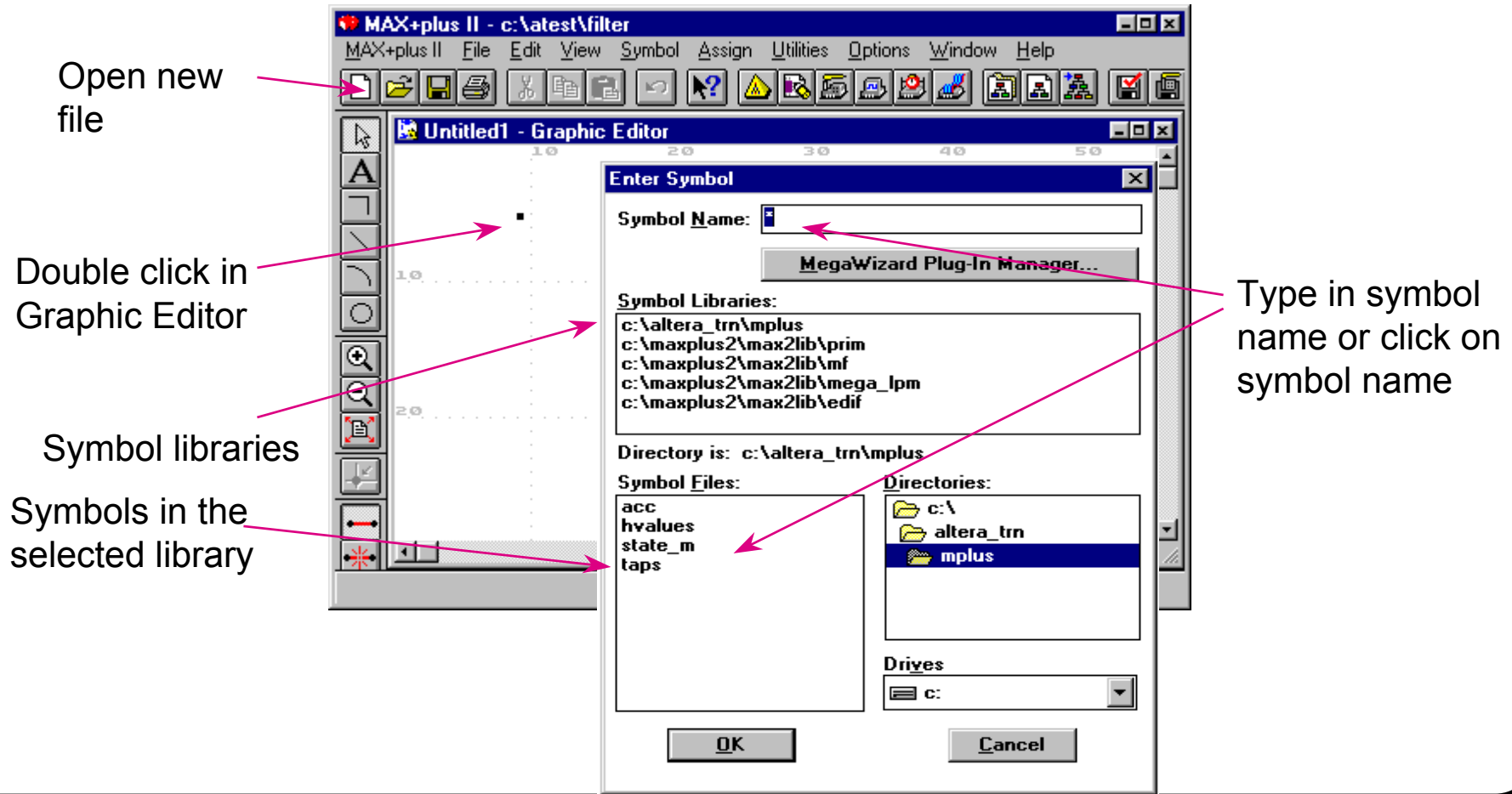
The Project directory has the highest priority, followed by the User Libraries, then by the Altera Libraries





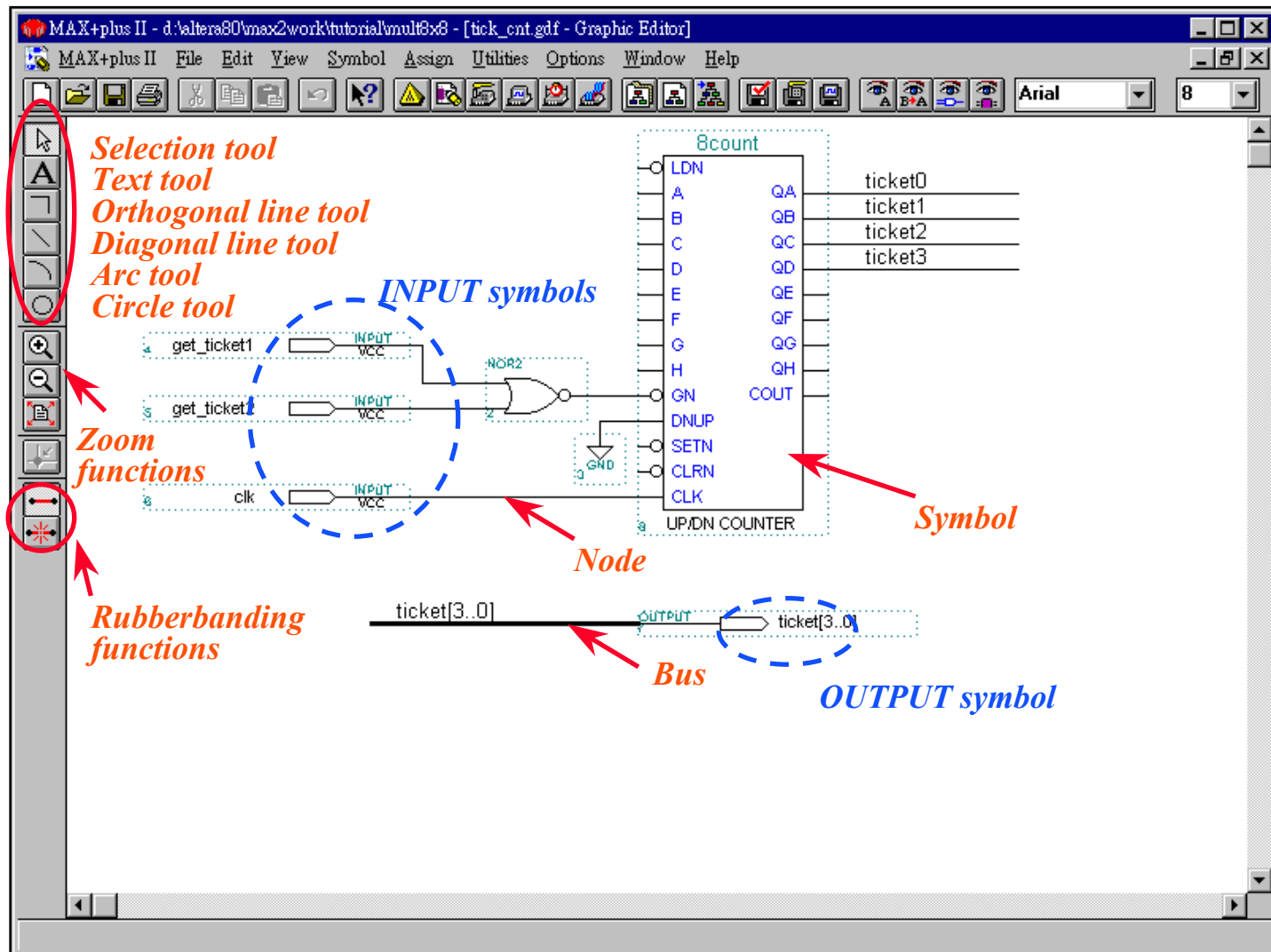
# Open New File & Enter Symbols

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





# Graphic Editor Window





# Making Connections

## ◆ Wire

- Single bit line

## ◆ Bus

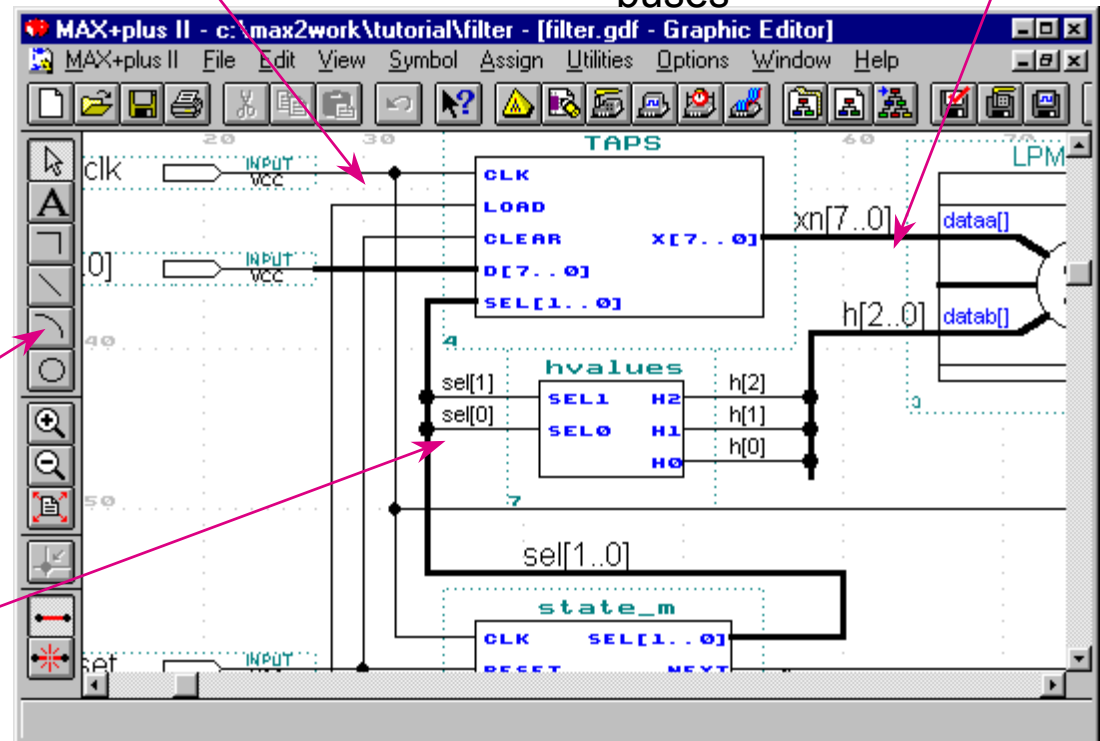
- Multi-bit line

## ◆ Signal name

- Matching name
- Attached to wire

Bus - Bus signal names required for LPM module buses

Drawing tool shortcuts  
Wire to Bus Connection





# Graphic Editor Options

## ◆ Font, Text Size

- Text Control

## ◆ Line Style

- Select Wire or Bus

## ◆ Display Assignments

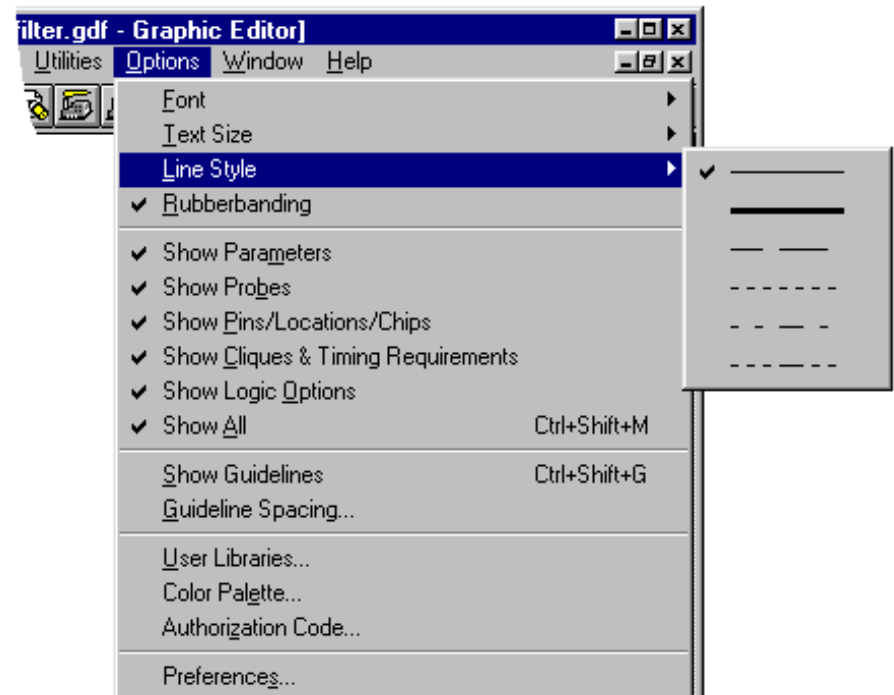
- Turns display on or off

## ◆ Guideline Control

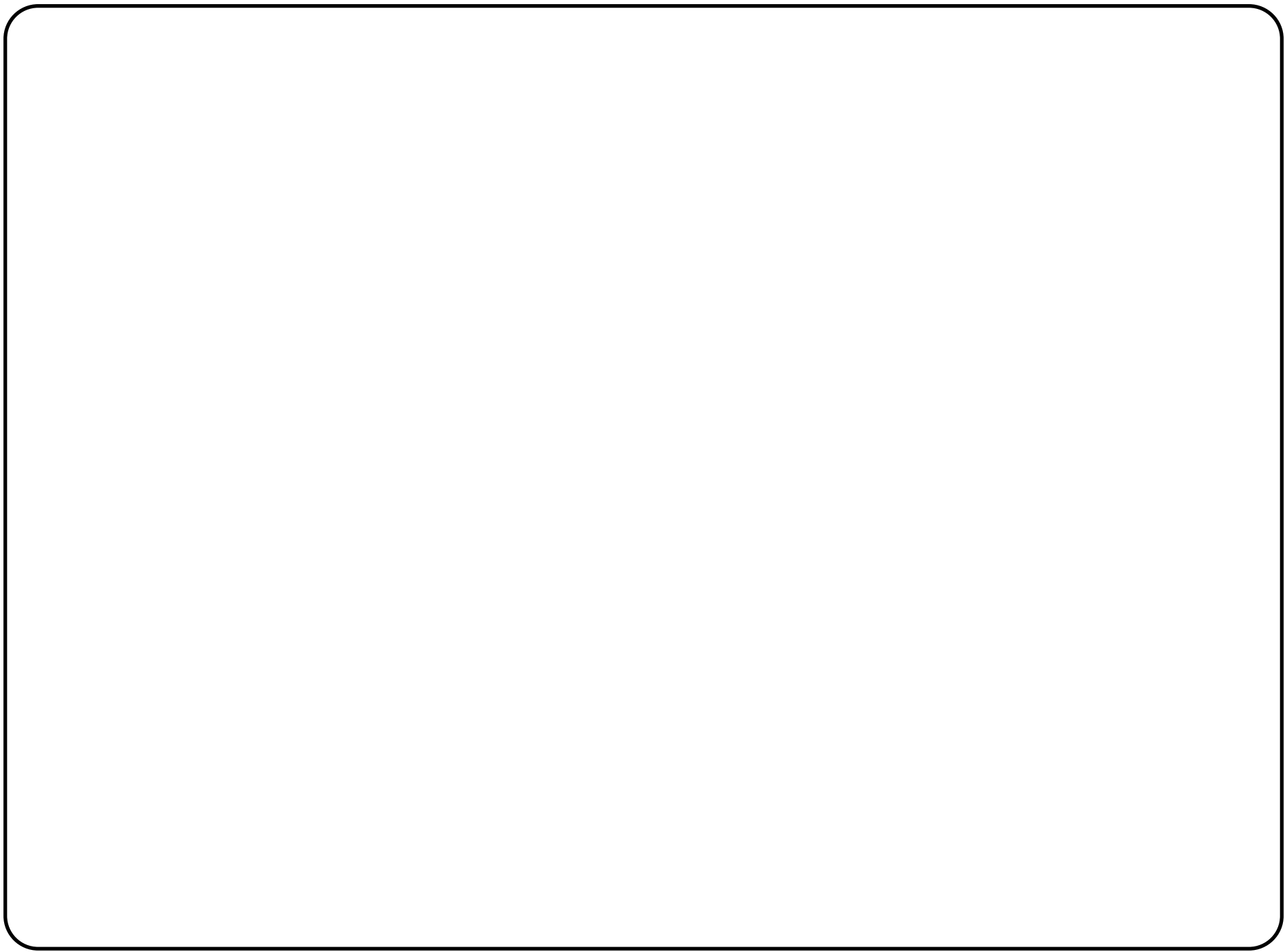
- Controls grid lines

## ◆ Rubber-banding

- Wires move with symbols



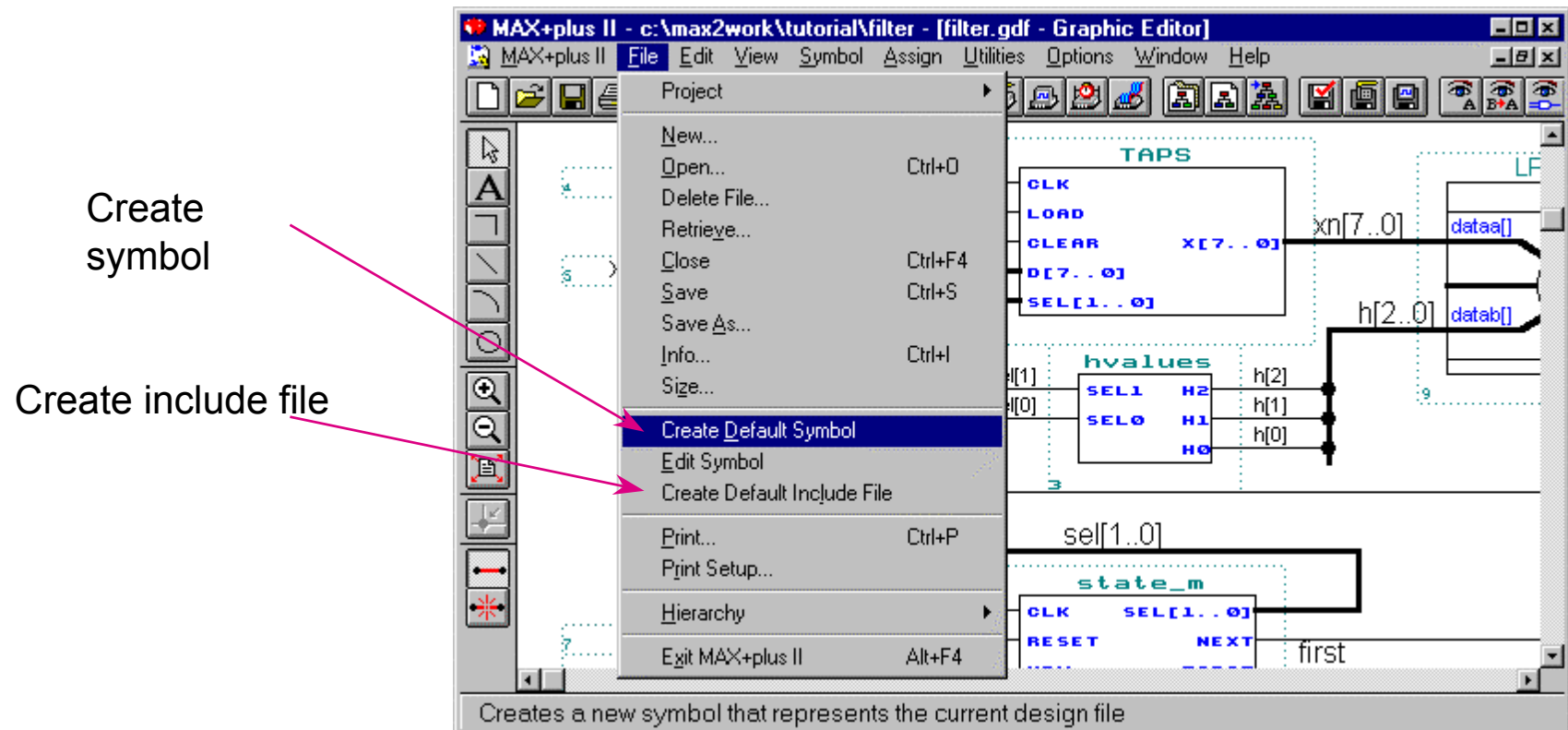






# Generate Symbols and Include Files

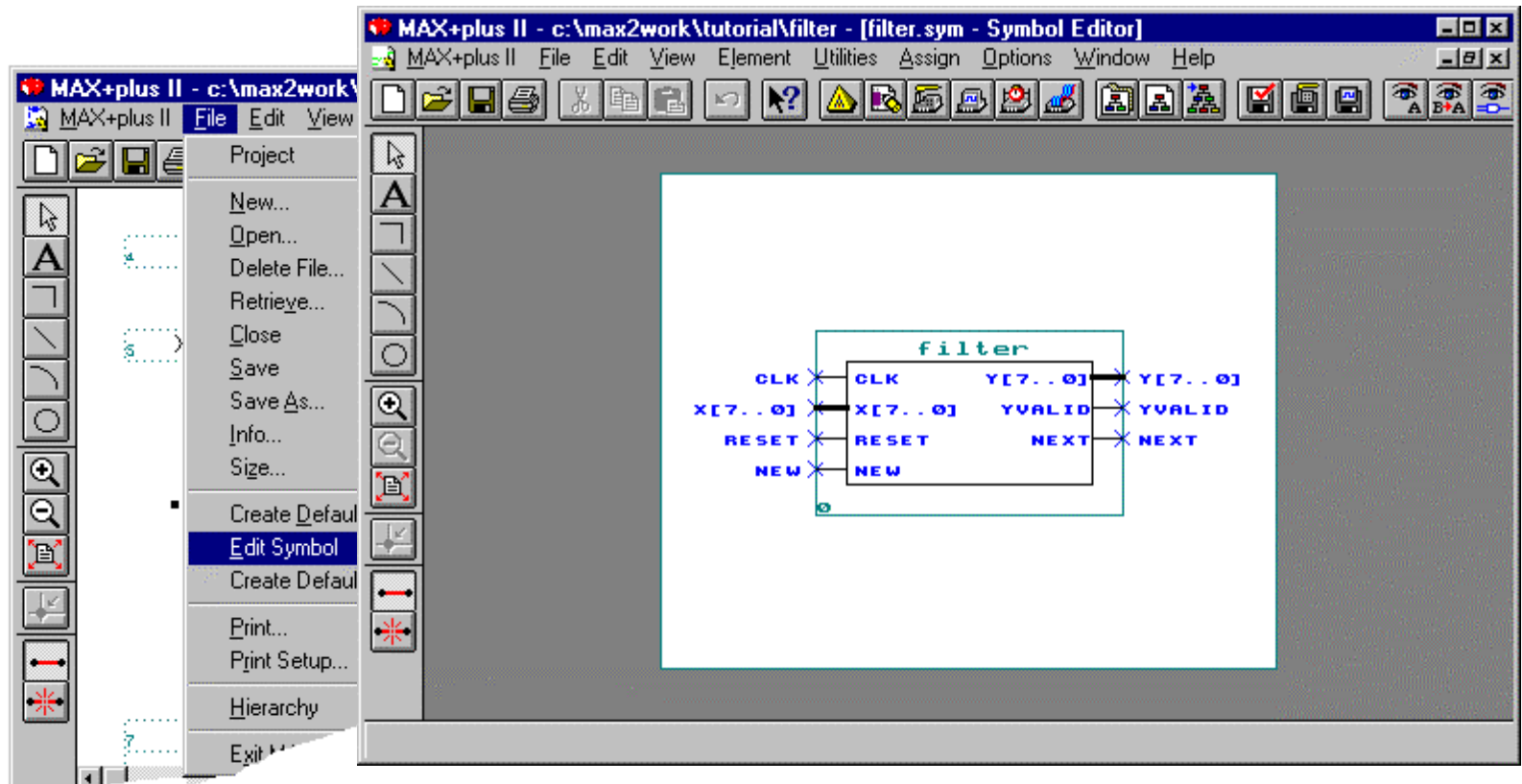
- ◆ Create symbol for higher-level schematic capture
- ◆ Create include file for AHDL or Verilog function prototype





# Symbol Editor

- ◆ Symbols can be modified with the Symbol Editor





# Pin/Node Naming

## ◆ Pin/node name

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

## ◆ Pin/node naming rules

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



# Bus Naming

## ◆ Single-range bus name

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

## ◆ Dual-range bus name

- Example:  $D[3..0][1..0] = D3\_1, D3\_0, D2\_1, D2\_0, D1\_1, D1\_0$
- A dual-range bus name uses two bracket-enclosed ranges  $[ ]$ : the bus width and the bus size. Bus widths and sizes can together define a maximum of 256 bits.

## ◆ Sequential bus name

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



# Using Buffer Primitives - (1)

## ◆ Buffer primitives

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

## ◆ GLOBAL primitive

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

## ◆ TRI primitive

- A active-high tri-state buffer

## ◆ OPNDRN primitive

- An open-drain buffer, equivalent to a `TRI` primitive whose output enable input is fed by an signal, but whose primary input is fed by a `GND` primitive
- Only supported for the FLEX 10K and MAX 7000S device families



# Using Buffer Primitives - (2)

## ◆ LCELL primitive

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

## ◆ SOFT primitive

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



# More on LPM Libraries

## ◆ Library of Parameterized Modules

- Standard Library of basic and functional elements
- Based on EDIF standard

## ◆ Advantage of LPMs

- Portability of design
- Architecture independence

## ◆ MAX+PLUS II and LPMs

- LPM can be used in graphical design and HDL designs
- LPM can be customized via the Megawizard feature



# Standard LPM without Megawizard

The screenshot shows the MAX+plus II software interface. The main window displays a circuit diagram with an LPM\_MULT component. Two inputs, 'a[4..1]' and 'b[4..1]', are connected to the 'dataa[]' and 'datab[]' ports of the multiplier. The output 'result[]' is connected to an output bus. A red box highlights the LPM\_MULT component and its associated parameter list on the right.

The parameter list on the right is as follows:

- INPUT\_A\_IS\_CONSTANT=
- INPUT\_B\_IS\_CONSTANT=
- LPM\_PIPELINE=
- LPM\_REPRESENTATION=
- LPM\_WIDTHA=4
- LPM\_WIDTHB=4
- LPM\_WIDTHP=(LPM\_WIDTHA+LPM\_WIDTHB)
- LPM\_WIDTHM=MAXIMIZE
- USE\_EAB=

The 'Edit Ports/Parameters' dialog box is open, showing the configuration for the LPM\_MULT function. The 'Function Name' is 'LPM\_MULT'. The 'Port Name' is 'aclr'. The 'Port Status' is 'Unused'. The 'Inversion' is 'None'. The 'Name' is 'aclr'. The 'Status' is 'Unused'. The 'Inversion' is 'None'. The 'Parameters' section shows the following table:

Parameter Name	Value
INPUT_A_IS_CONSTANT	<none>
INPUT_B_IS_CONSTANT	<none>
LPM_PIPELINE	<none>
LPM_REPRESENTATION	<none>

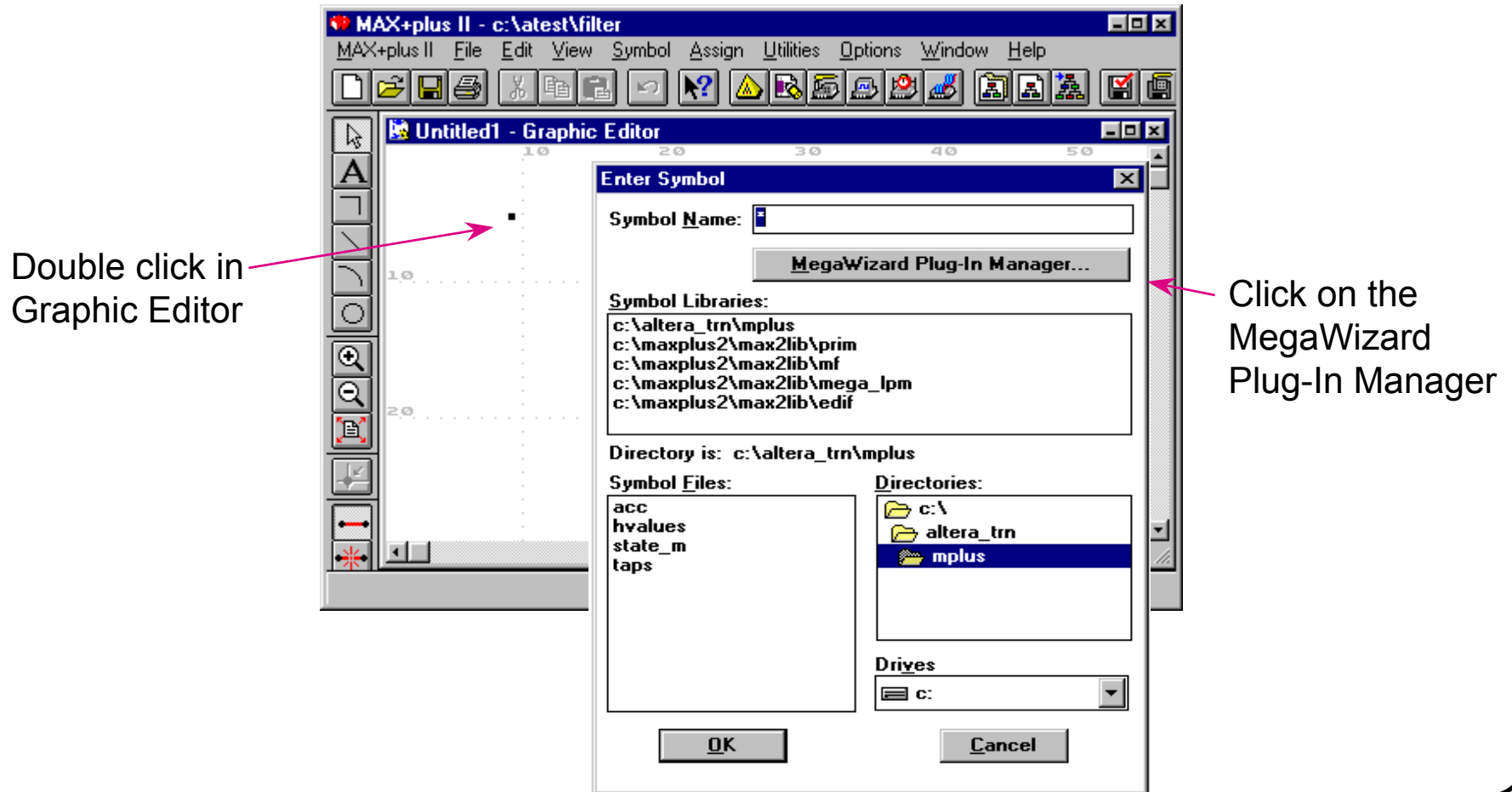
At the bottom of the dialog box, there are 'OK' and 'Cancel' buttons.

For Help on this dialog box, press F1



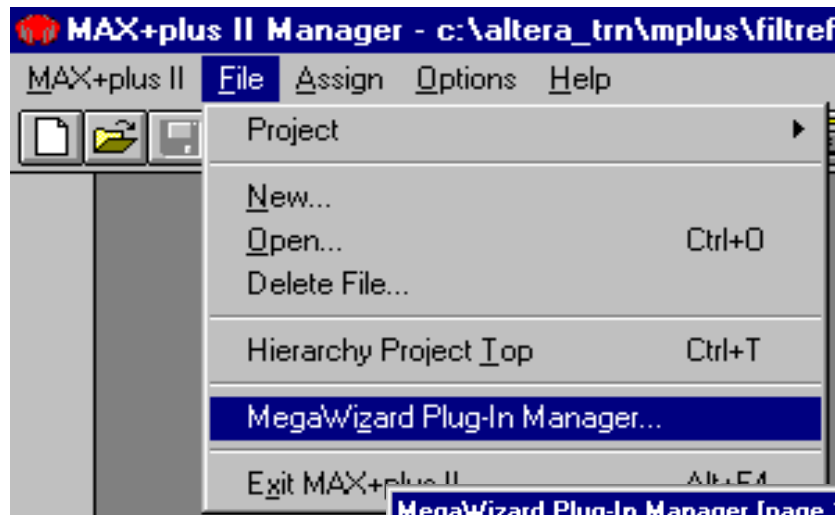
# Using MegaWizard Plug-In Manager

## ◆ Click on the MegaWizard Plug-In Manager Button





# Accessing the MegaWizard



Select MegaWizard Plug-In Manager



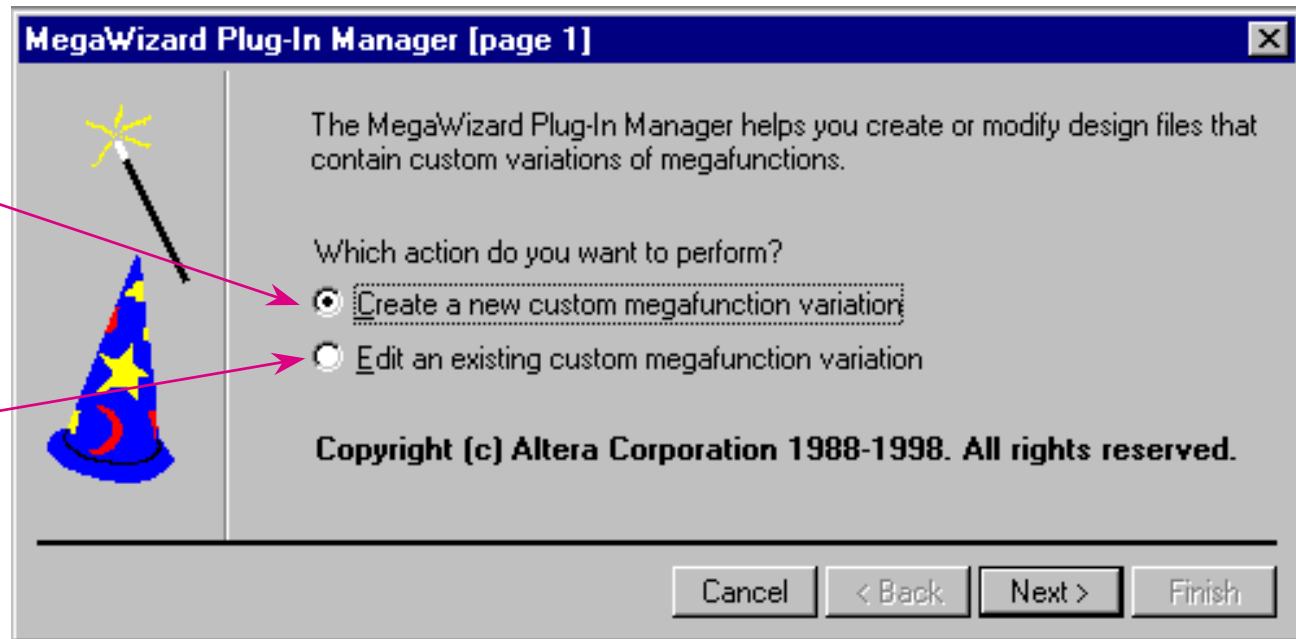


# New vs Existing Megafunction

- ◆ Choose between a new custom megafunction variation or an existing megafunction variation

New Custom  
Megafunction

Edit Existing  
Custom  
Megafunction





# Available Megafunctions & Output File

Select a function from the available megafunction

MegaWizard Plug-In Manager [page 2a]

Available Megafunctions:

- + LPM arithmetic
- LPM gates
  - LPM\_AND
  - LPM\_BUSTRI
  - LPM\_CLSHIFT
  - LPM\_CONSTANT
  - LPM\_DECODE
  - LPM\_INV
  - LPM\_MUX**
  - LPM\_OR
  - LPM\_XOR
- + LPM storage

Which megafunction would you like to customize? Select a megafunction from the list at left.

Which type of output file do you want to create?

☒ AHDL

☐ VHDL

☐ Verilog HDL

What name do you want for the output file?

c:\altera\_tn\mplus\tap\_mux

Note: To compile a project successfully in the MAX+PLUS II software, your design files must be in the project directory or a user library that you specify with the User Libraries command (Options menu).

Your current user library directories are:

Select a type of output file

Select a directory and a output file name



# Customizing the Megafunction

MegaWizard Plug-In Manager - LPM\_MUX [page 3 of 4]

How many 'data' inputs do you want?

How wide should the 'data' input and the 'result' output buses be?  bits

Do you want to pipeline the multiplexer?

☒ No

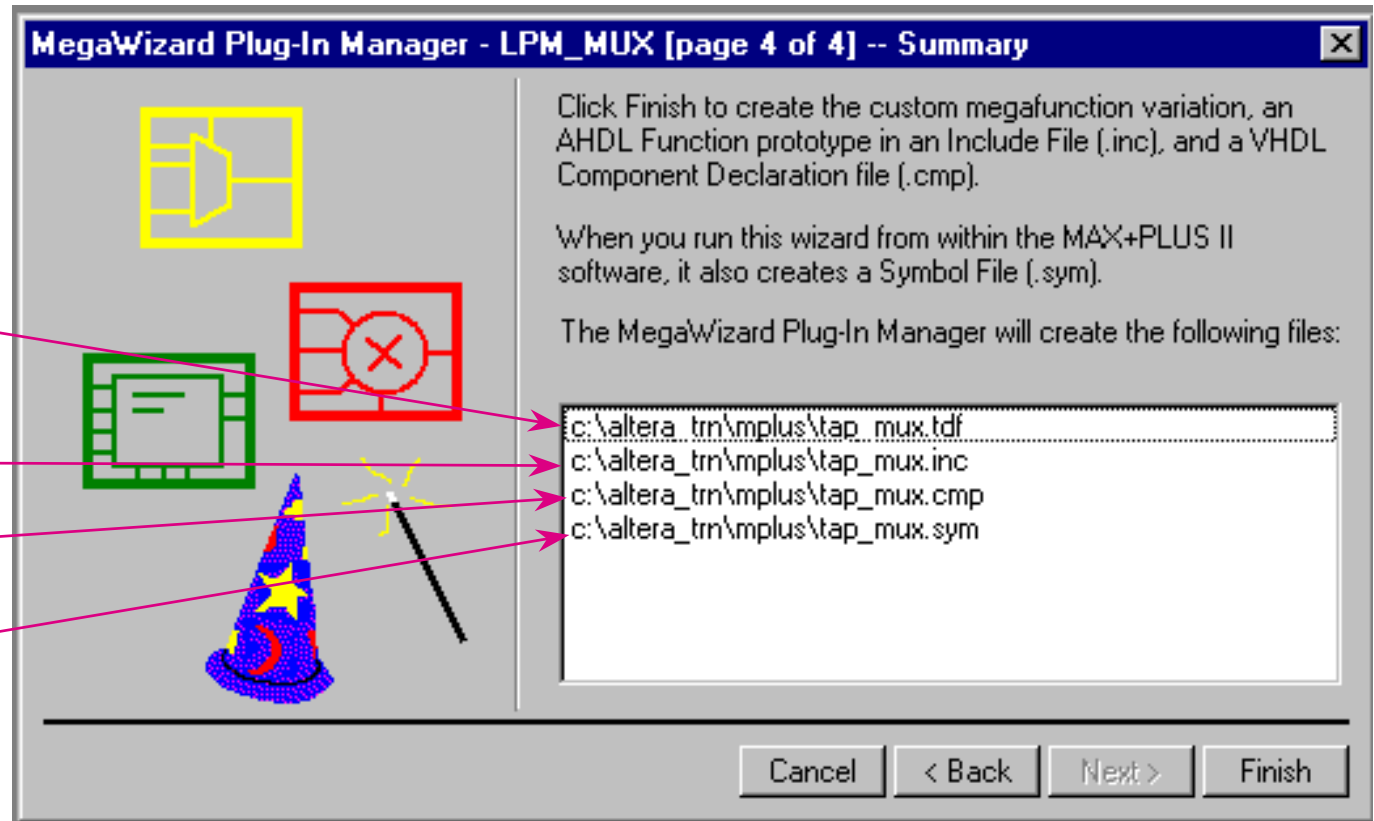
☐ Yes, I want an output latency of  Clock cycles

☐ Create an asynchronous Clear input

Cancel < Back Next > Finish



# Files generated by the MegaWizard



Design file implemented  
in the language you selected  
(.tdf, .vhd, or .v)

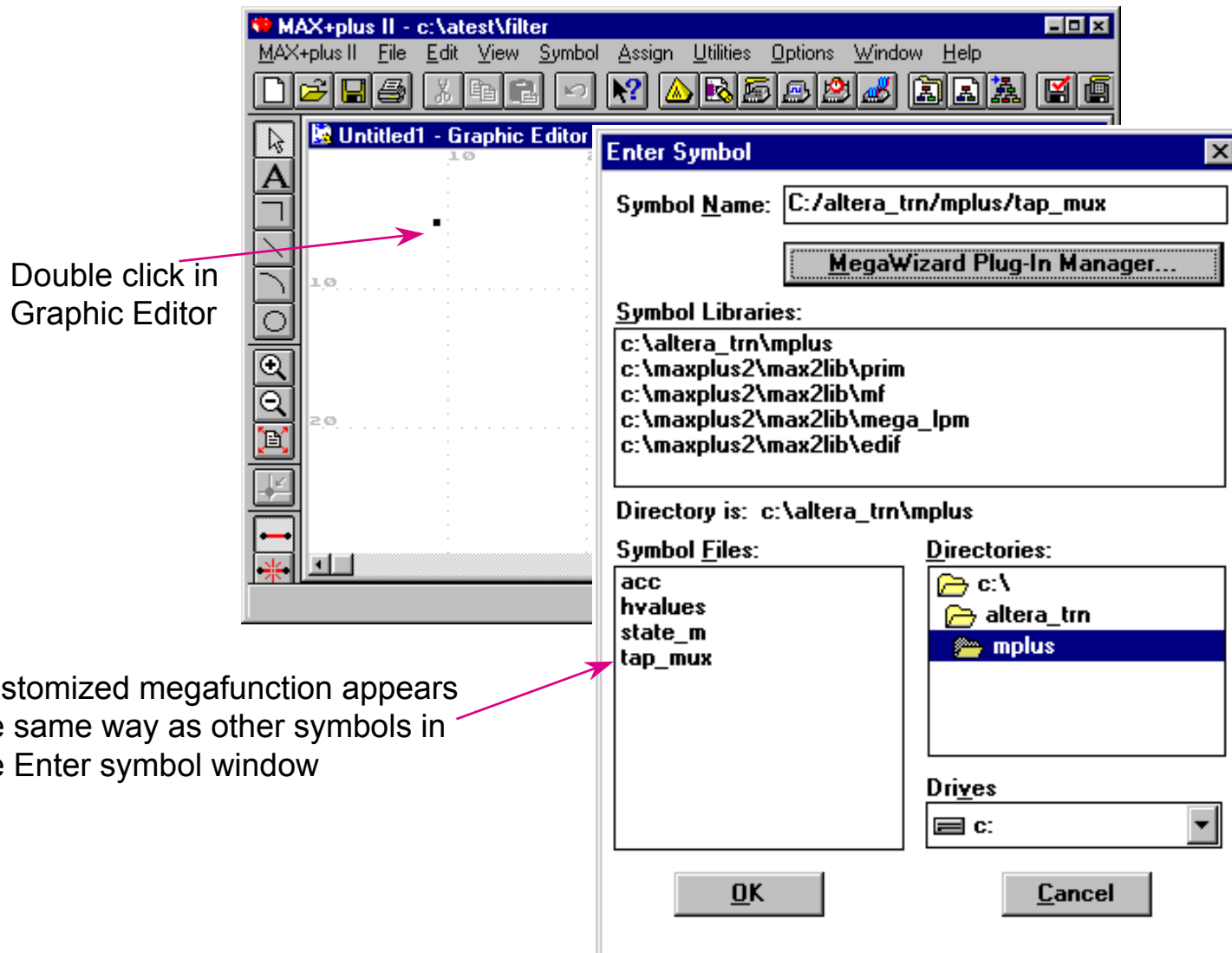
INC an AHDL include file

CMP a VHDL component  
declaration file

SYM a Graphic design  
symbol file



# Entering Customized Megafunction



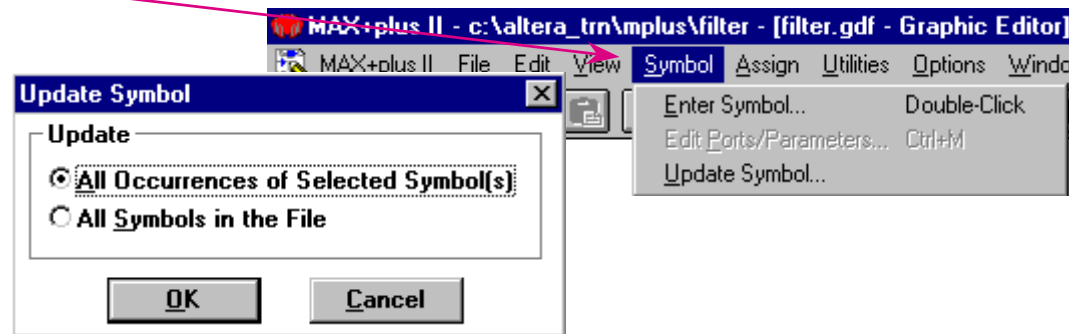
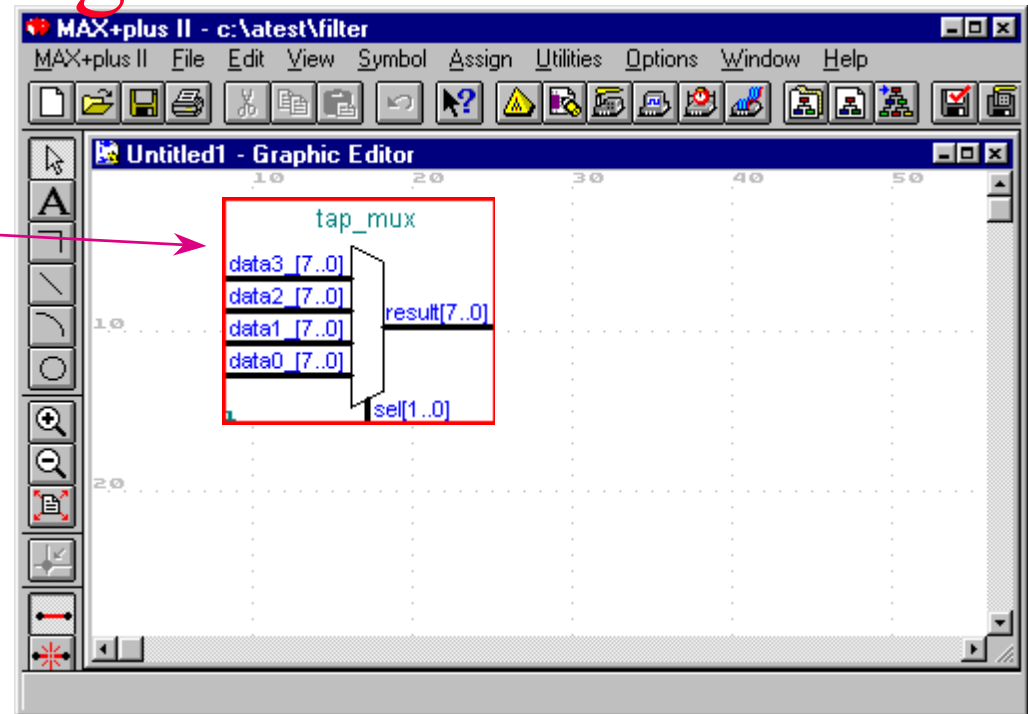


# Make Changes to Customized Megafunction

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

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

Remember to update the symbol in your graphic editor

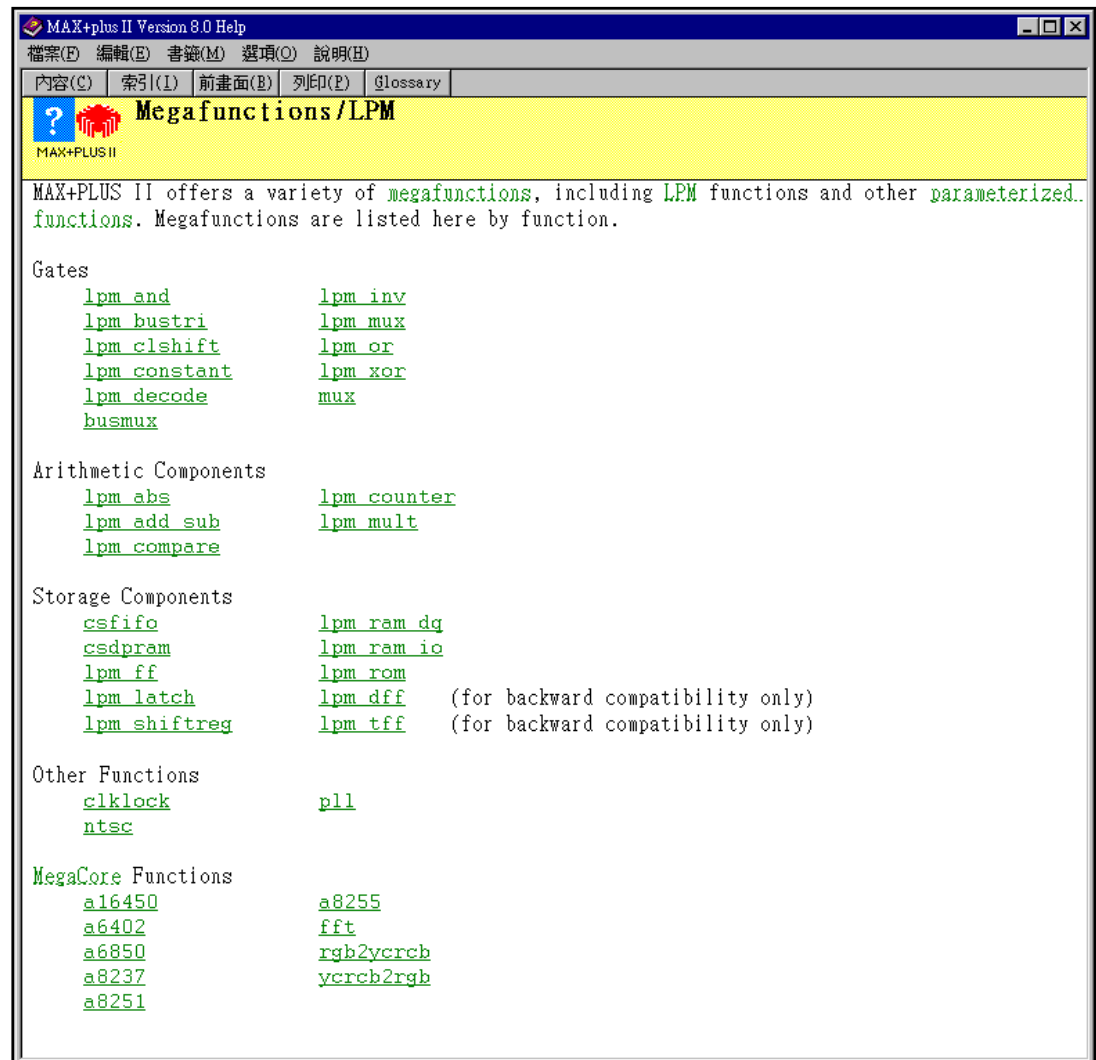




# How to Use System Functions?

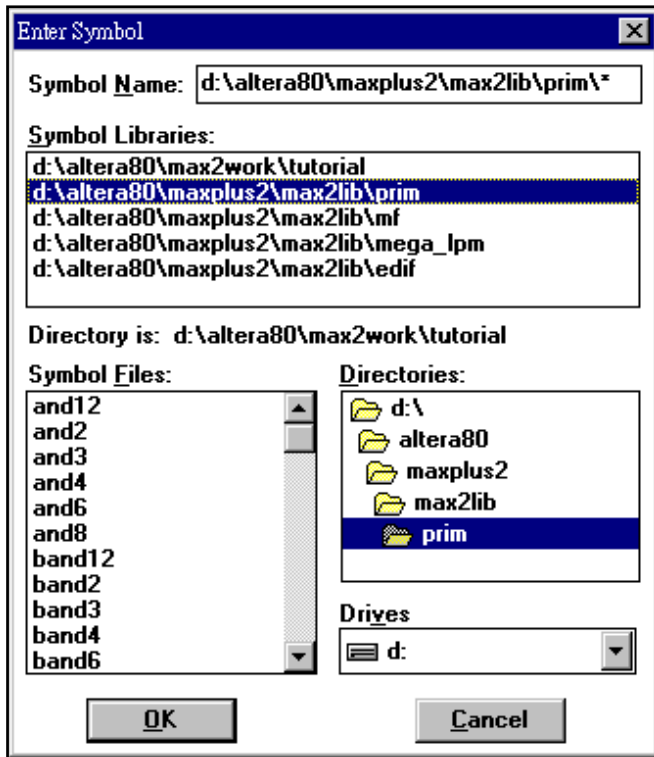
## ◆ To get help...

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





# Entering Symbols



## ◆ Enter a symbol

Menu: *Symbol -> Enter Symbol...*

*(or by double clicking on the empty workspace)*

## ◆ Move/cut/copy/paste symbols

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

## ◆ Commands regarding the symbol

- Just click the right mouse button on the symbol



# Entering I/O Symbol

## ◆ I/O symbols

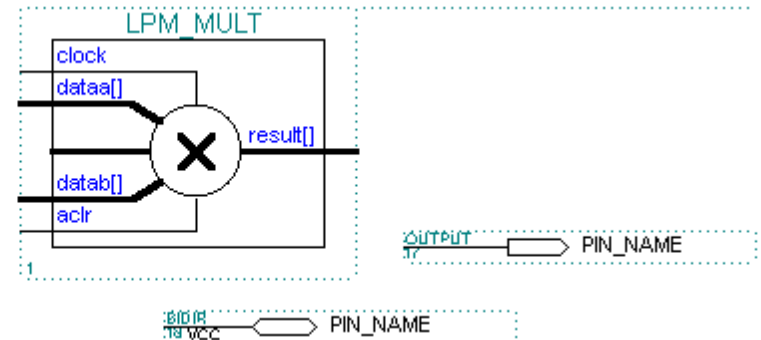
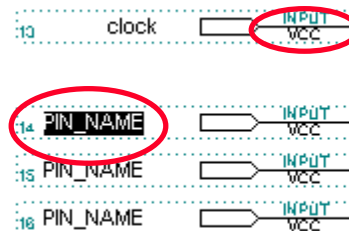
- Input pin/port: enter a INPUT symbol
- Output pin/port: enter a OUTPUT symbol
- Bidirectional pin/port: enter a BIDIR symbol

## ◆ Name the I/O pins/ports

- Double click on the “PIN\_NAME” field of the I/O symbol

## ◆ Pin default value

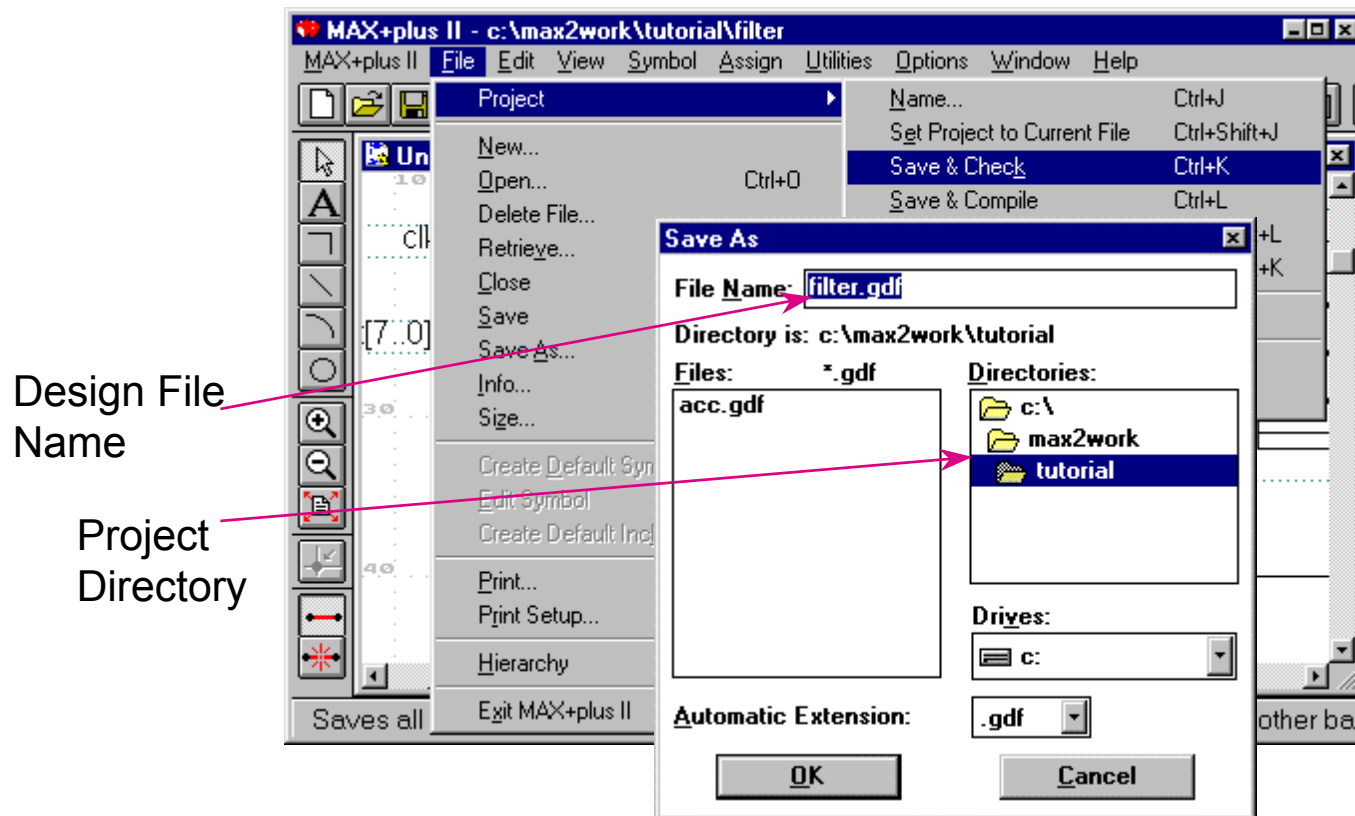
- The values assigned to unconnected INPUT and BIDIR primitives when the symbol that represents the current GDF file is used in a higher-level design file
- Default is VCC
- Double click on the “VCC” field to set the default value





# Save & Check the Design

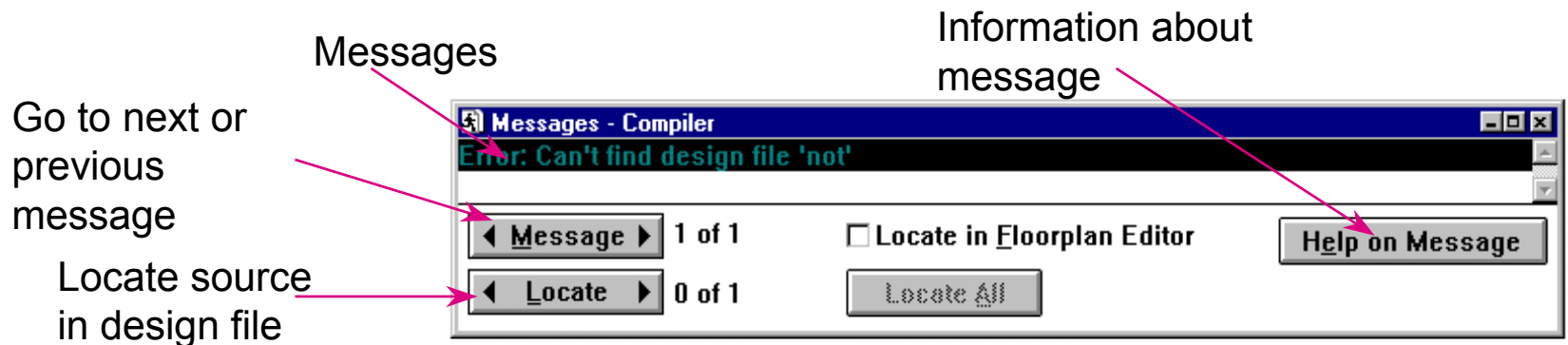
- Save & check the design file with .gdf extension
- Correct any errors with the aid of Message Processor





# Message Processor

- Lists all Info, Warning and Error messages
  - Info messages are general information
  - Warning messages are possible problems
  - Error messages indicate Compiler is unable to complete compilation process
- Provides help on the messages
- Locates source of message in design file

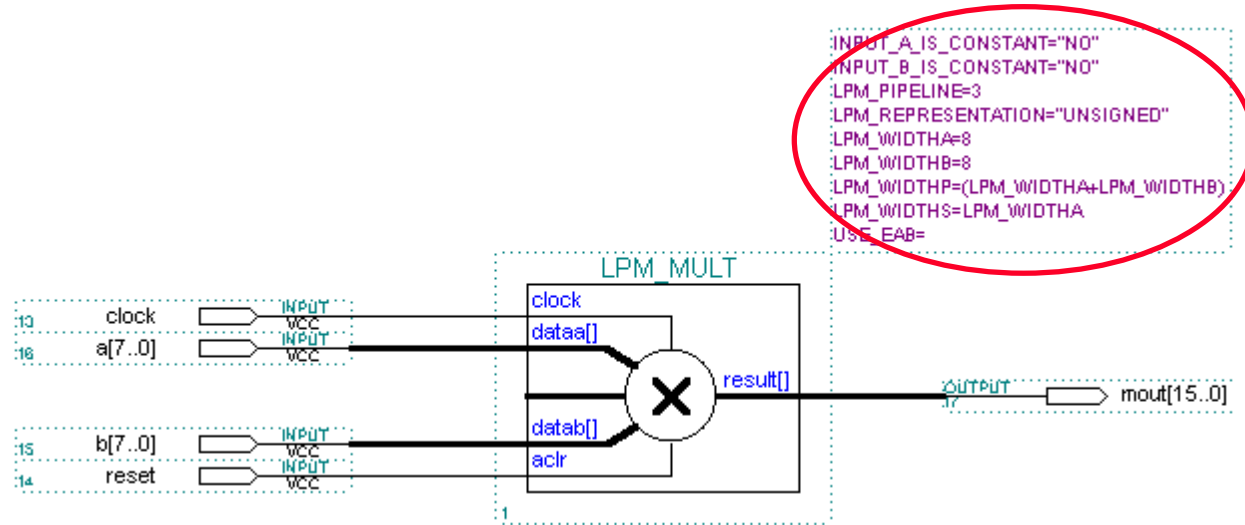




# Example: Multiplier

## ◆ Design a multiplier with `LPM_MULT`

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

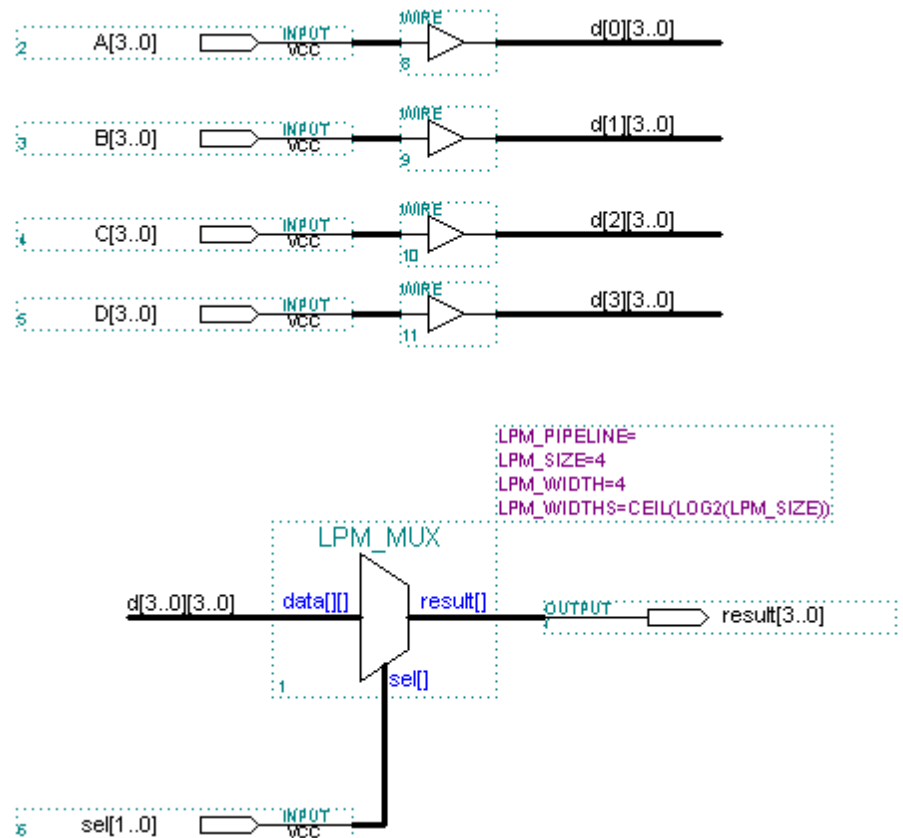




# Example: Multiplexer

## ◆ Design a multiplexer with LPM\_MUX

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

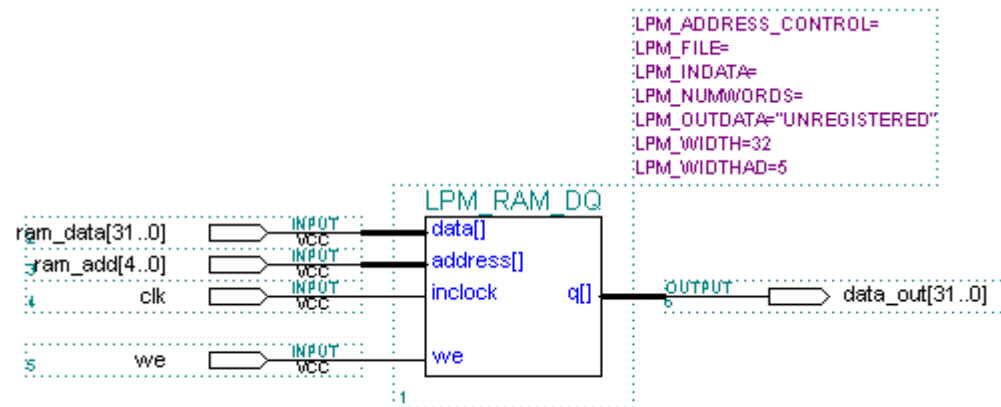




# Example: RAM

## ◆ Design RAM circuit with LPM

- Use LPM\_RAM\_IO to design RAM with a single input & output port
- Use LPM\_RAM\_DQ to design RAM with separate input & output ports

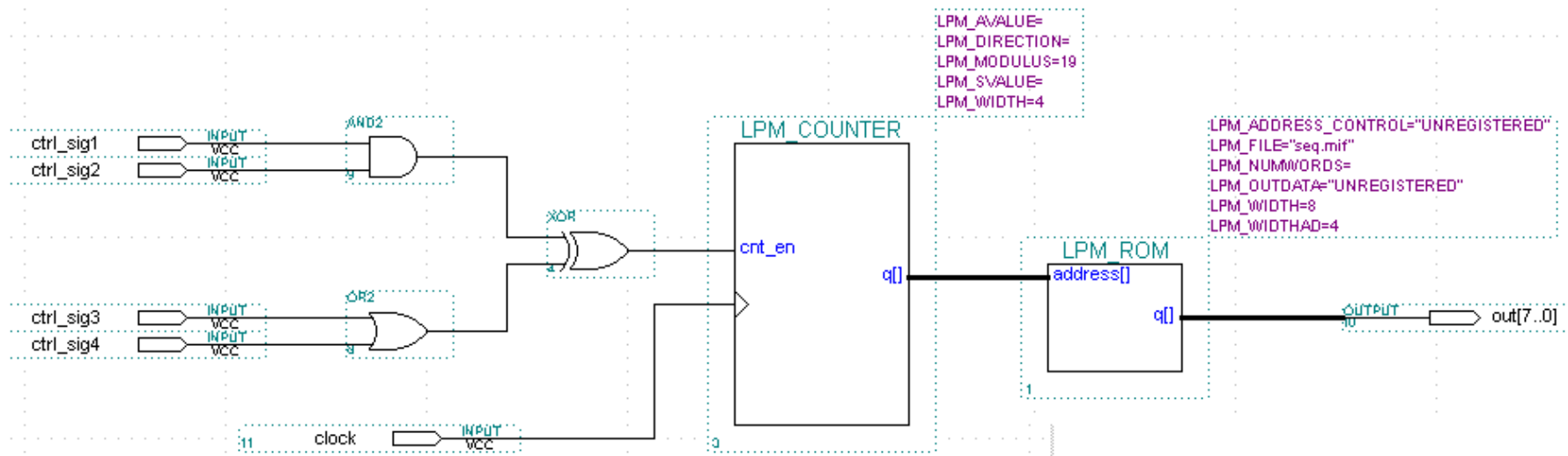




# Example: Sequencer

## ◆ Design a sequencer with 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)

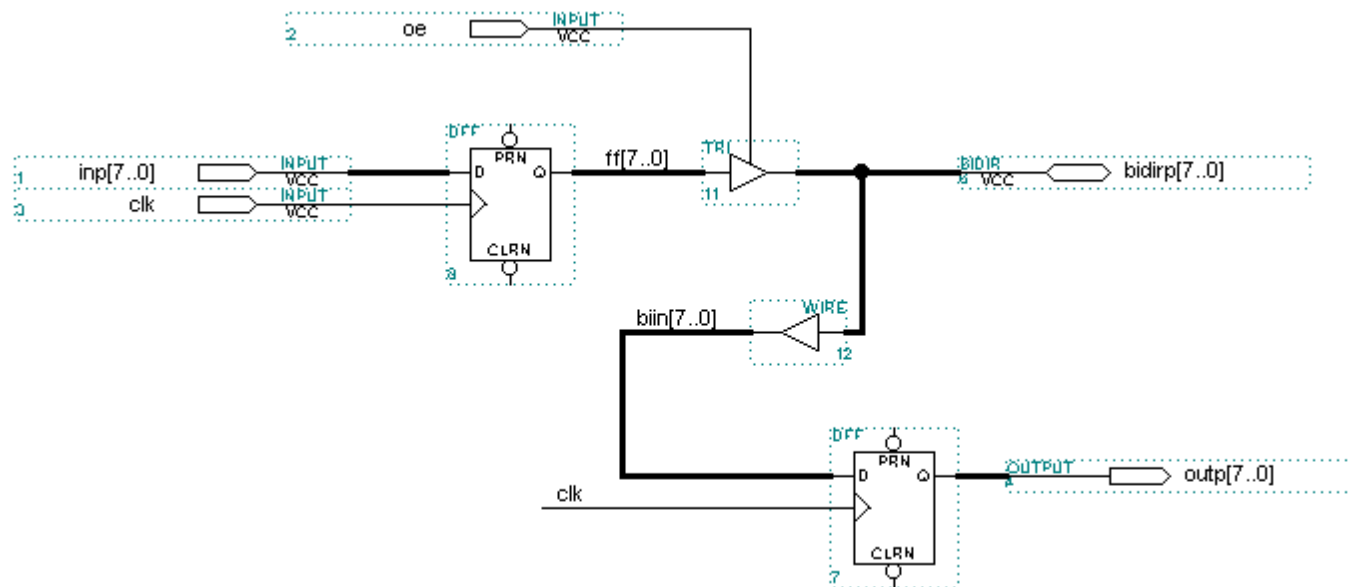




# Example: Bidirectional Pin

## ◆ 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

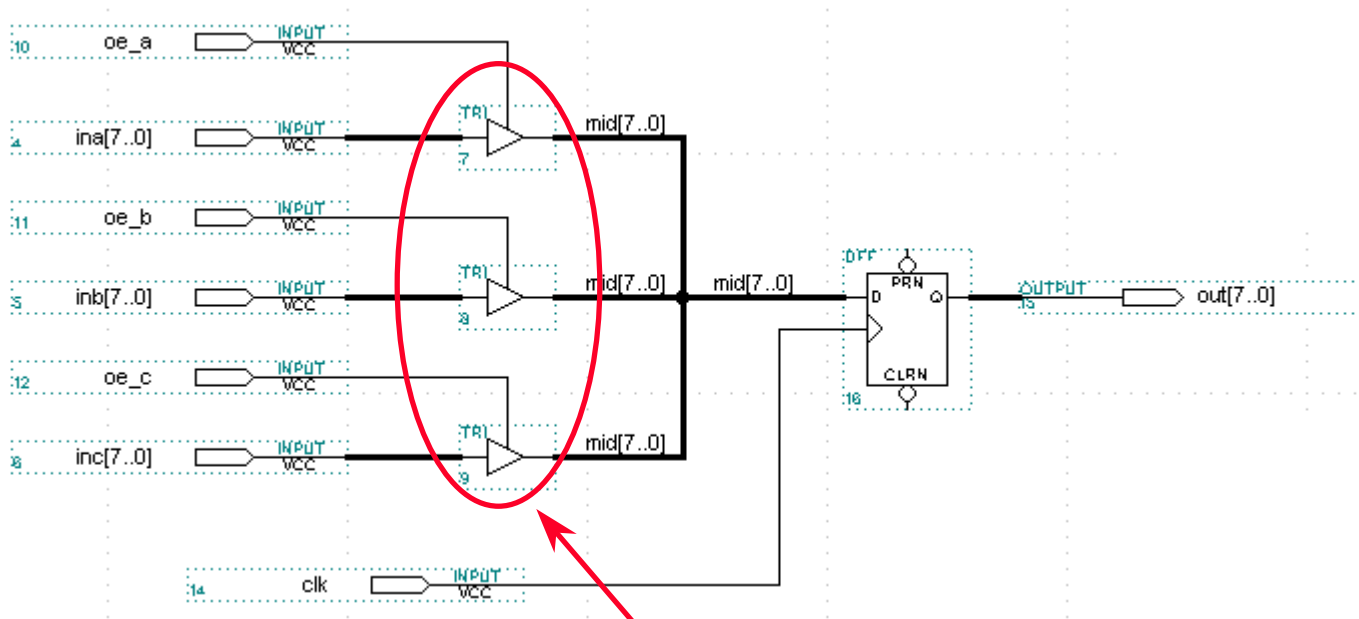




# Example: Tri-State Buses - (1)

## ◆ 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



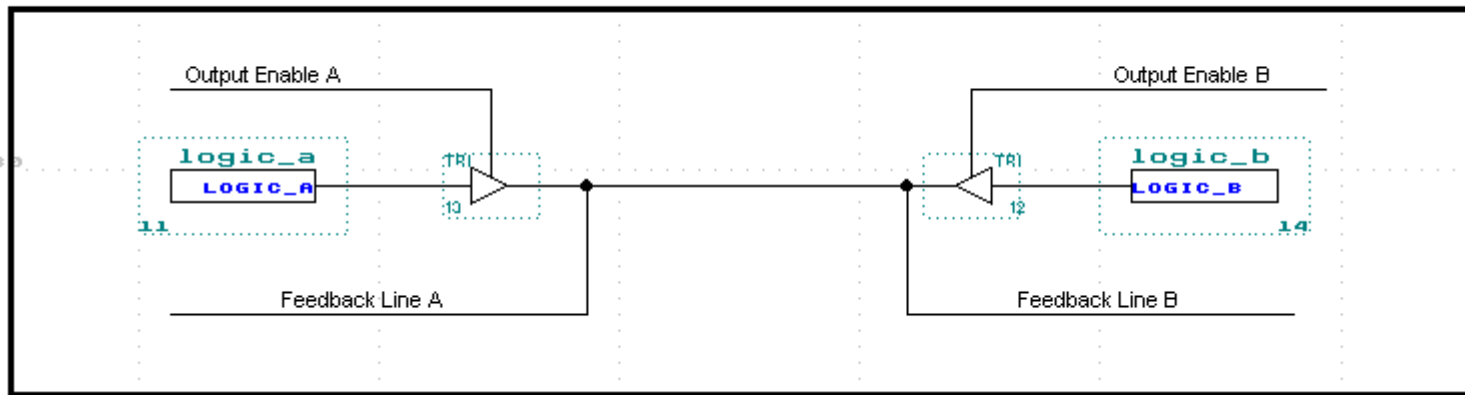
*MAX+PLUS II will automatically convert it into a multiplexer.  
If the tri-state buffers feed a pin, a tri-state buffer will be available  
after the multiplexer.*



# Example: Tri-State Buses - (2)

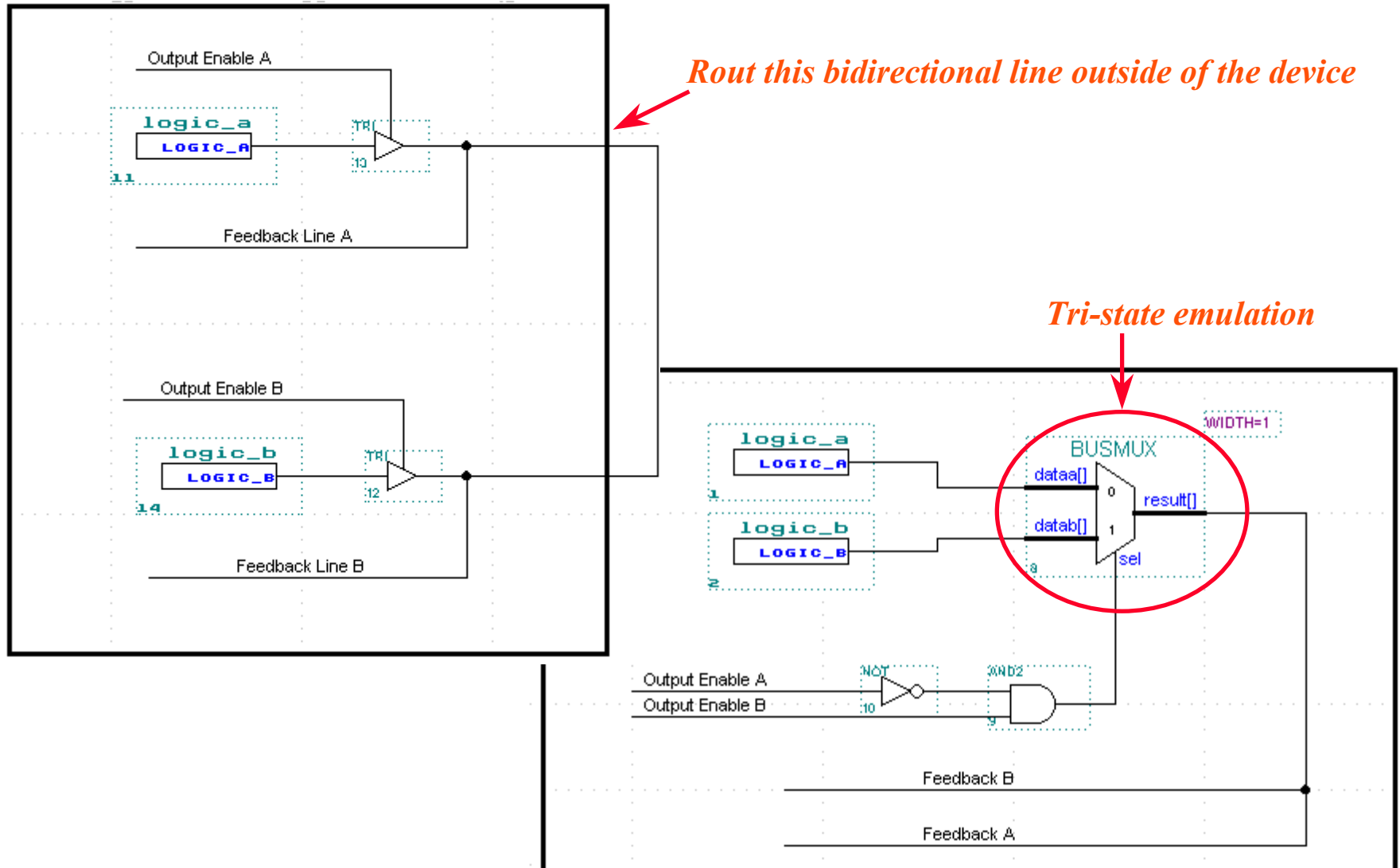
## ◆ Tri-state buses for bidirectional communication

- 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 rout this bidirectional line outside of the device, which uses the tri-states present at the I/O pins, or you can convert the tri-state bus into a multiplexer





# Example: Tri-State Buses - (3)





# Text Design Entry

## ◆ Set up a new project

- Same as Graphic Design Entry

## ◆ Enter text description

- AHDL
- VHDL
- Verilog

## ◆ Save & check the design

- Similar to Graphic Design Entry
- The file extension is .tdf, .vhd, .v



# AHDL

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



# VHDL

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



# Verilog

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



# AHDL Design Entry

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



# What's AHDL?

## ◆ AHDL: Altera Hardware Description Language

- To create MAX+PLUS II text design file (\*.tdf)
- High-level, modular hardware description language
- Completely integrated into the MAX+PLUS II system
- Especially well suited for...
  - Complex combinational logic
  - Group operations
  - State machines
  - Truth tables
  - Parameterized logic
- Easy to learn & debug under MAX+PLUS II system



# AHDL Example - (1)

```

SUBDESIGN 7segment
( i[3..0]                : INPUT;
  a, b, c, d, e, f, g : OUTPUT;
)
%   -a-                0 1 2 3                %
% f|    |b            4 5 6 7                %
%   -g-                8 9 A b                %
% e|    |c            C d E F                %
%   -d-                %

BEGIN
TABLE
    i[3..0] => a, b, c, d, e, f, g;
    H"0"    => 1, 1, 1, 1, 1, 1, 0;
    H"1"    => 0, 1, 1, 0, 0, 0, 0;
    H"2"    => 1, 1, 0, 1, 1, 0, 1;
    H"3"    => 1, 1, 1, 1, 0, 0, 1;
    H"4"    => 0, 1, 1, 0, 0, 1, 1;
    H"5"    => 1, 0, 1, 1, 0, 1, 1;
    H"6"    => 1, 0, 1, 1, 1, 1, 1;
    H"7"    => 1, 1, 1, 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;  
  out   : OUTPUT;  
  c     : BIDIR;  
)
```

```
VARIABLE
```

```
  b : NODE;  
  t : TRI_STATE_NODE;
```

```
BEGIN
```

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

```
END;
```



# 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

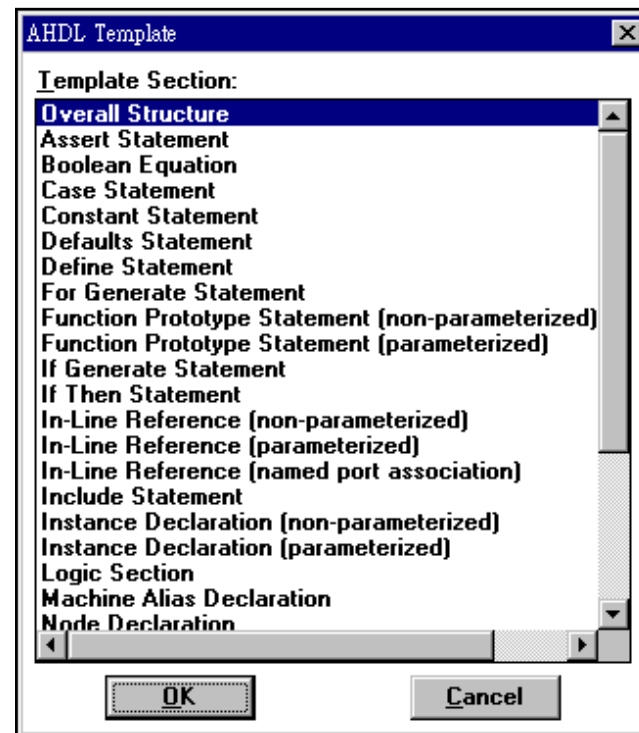


# AHDL Templates

## ◆ AHDL templates make design easier

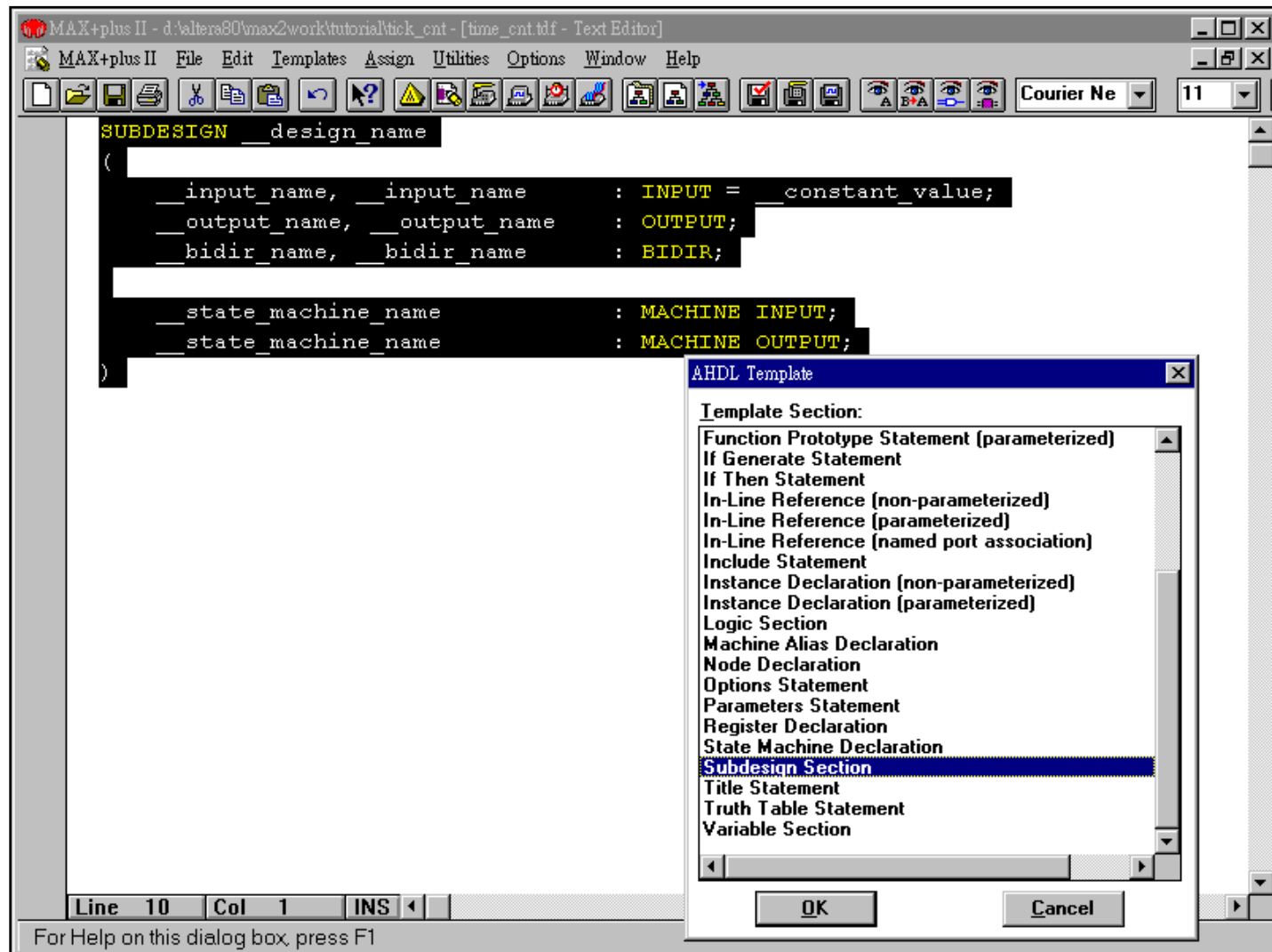
- You can insert AHDL template into your TDF, then replace placeholder variables in the templates with your own identifiers and expressions

Menu: *Templates -> AHDL Template...*





# Inserting AHDL Template





# Using Syntax Coloring

## ◆ Syntax Coloring command

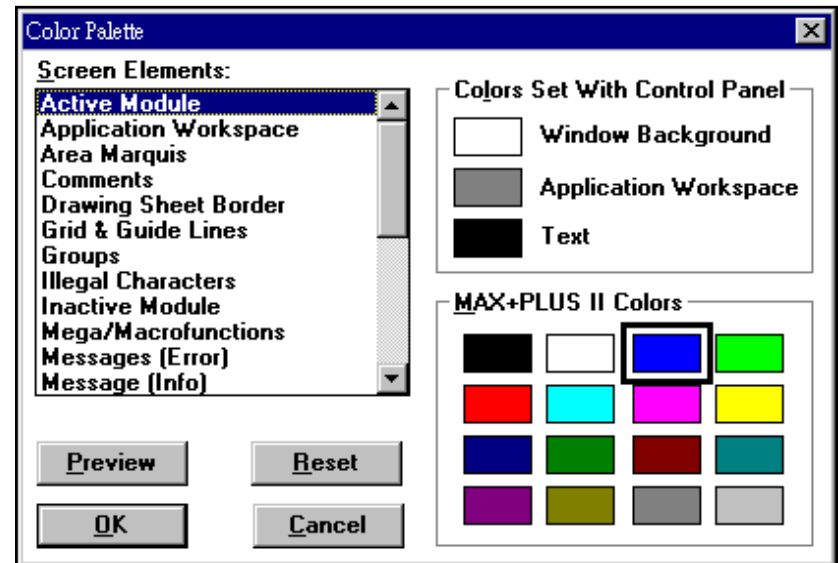
- To improve TDF readability & accuracy

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





The screenshot shows the MAX+plus II Text Editor window. The title bar indicates the file is 'auto\_max.tdf'. The menu bar includes File, Edit, Templates, Assign, Utilities, Options, Window, and Help. The toolbar contains various icons for file operations and editing. The text area displays the following Verilog code:

```

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 16 states
                                OF BITS (q2,q1,q0)       % 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
                                    ...                  % ...
                                )

```

The status bar at the bottom shows Line 4, Col 24, and the cursor is in the Insert (INS) mode.



# 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_WIDTHHP=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_WIDTHS-1..0])  
  WITH (LPM_WIDTH, LPM_SIZE, LPM_WIDTHS, CASCADE_CHAIN)  
  RETURNS (result[LPM_WIDTH-1..0]);  
  
SUBDESIGN mux  
(  
  a[3..0], b[3..0], c[3..0], d[3..0] : INPUT;  
  select[1..0]                       : INPUT;  
  result[3..0]                       : OUTPUT;  
)  
  
BEGIN  
  result[3..0] = lpm_mux (a[3..0], b[3..0], c[3..0], d[3..0], select[1..0])  
    WITH (LPM_WIDTH=4, LPM_SIZE=4, LPM_WIDTHS=2);  
END;
```



# 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
- ◆ Design Entry Summary



# MAX+PLUS II Waveform Editor

## ◆ Features of MAX+PLUS II Waveform Editor

- To serve 2 roles:
  - As a design entry tool: to create Altera waveform design files (\*.wdf)
  - As a tool for entering test vectors & viewing simulation results: simulation channel files (\*.scf)

## ◆ For design entry

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

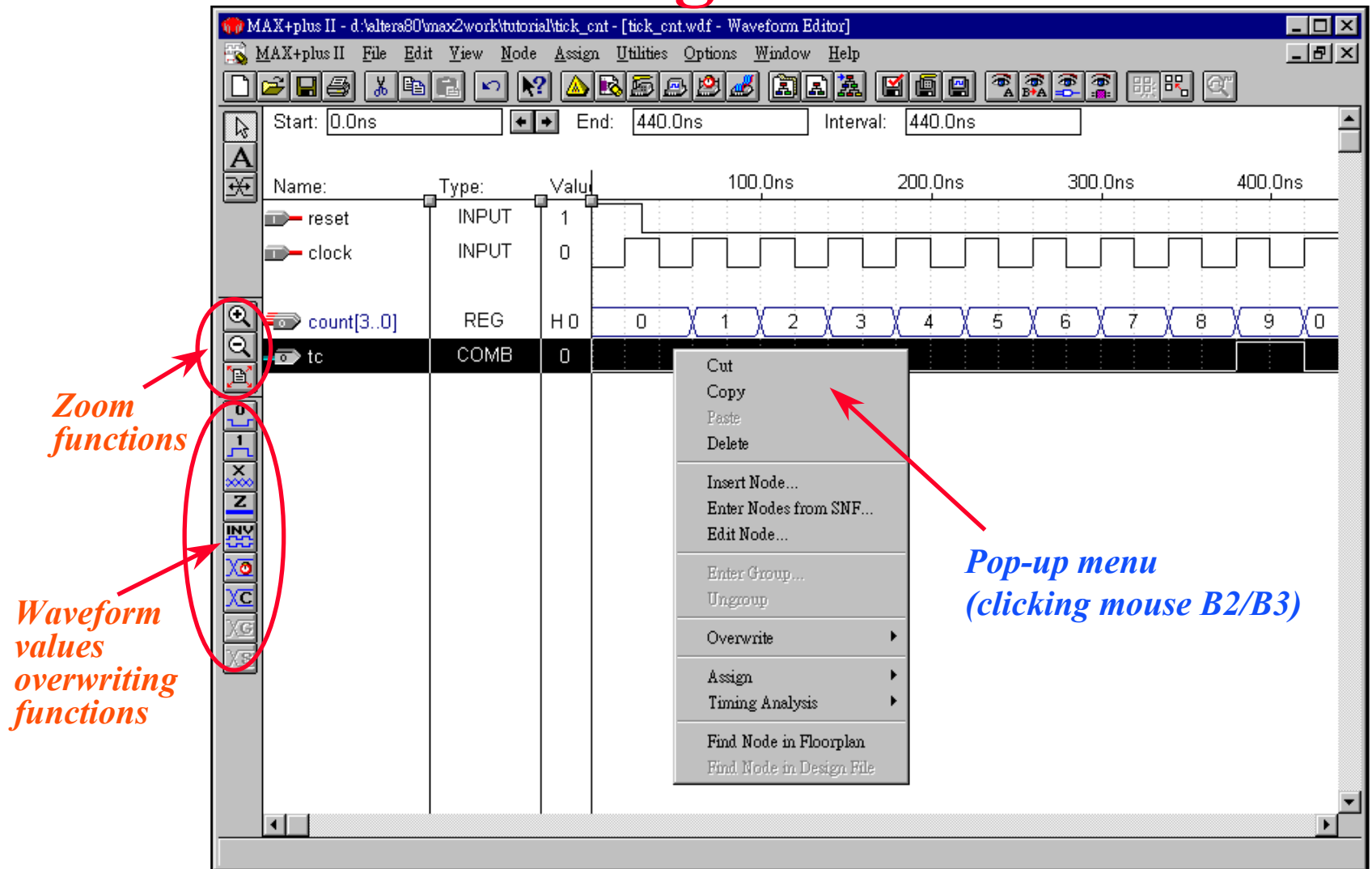
## ◆ For design verification

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



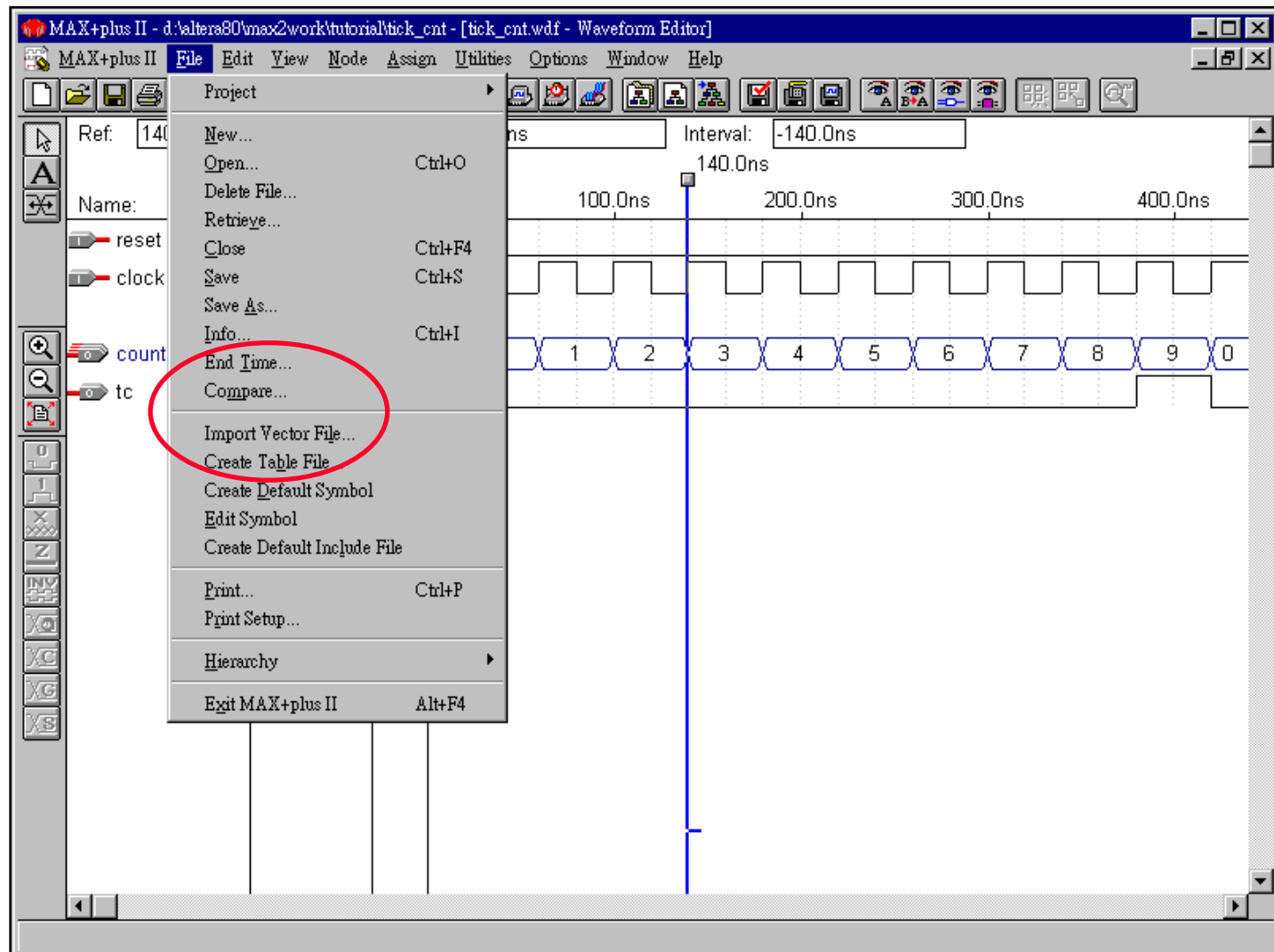
# MAX+PLUS II

## Waveform Design Environment



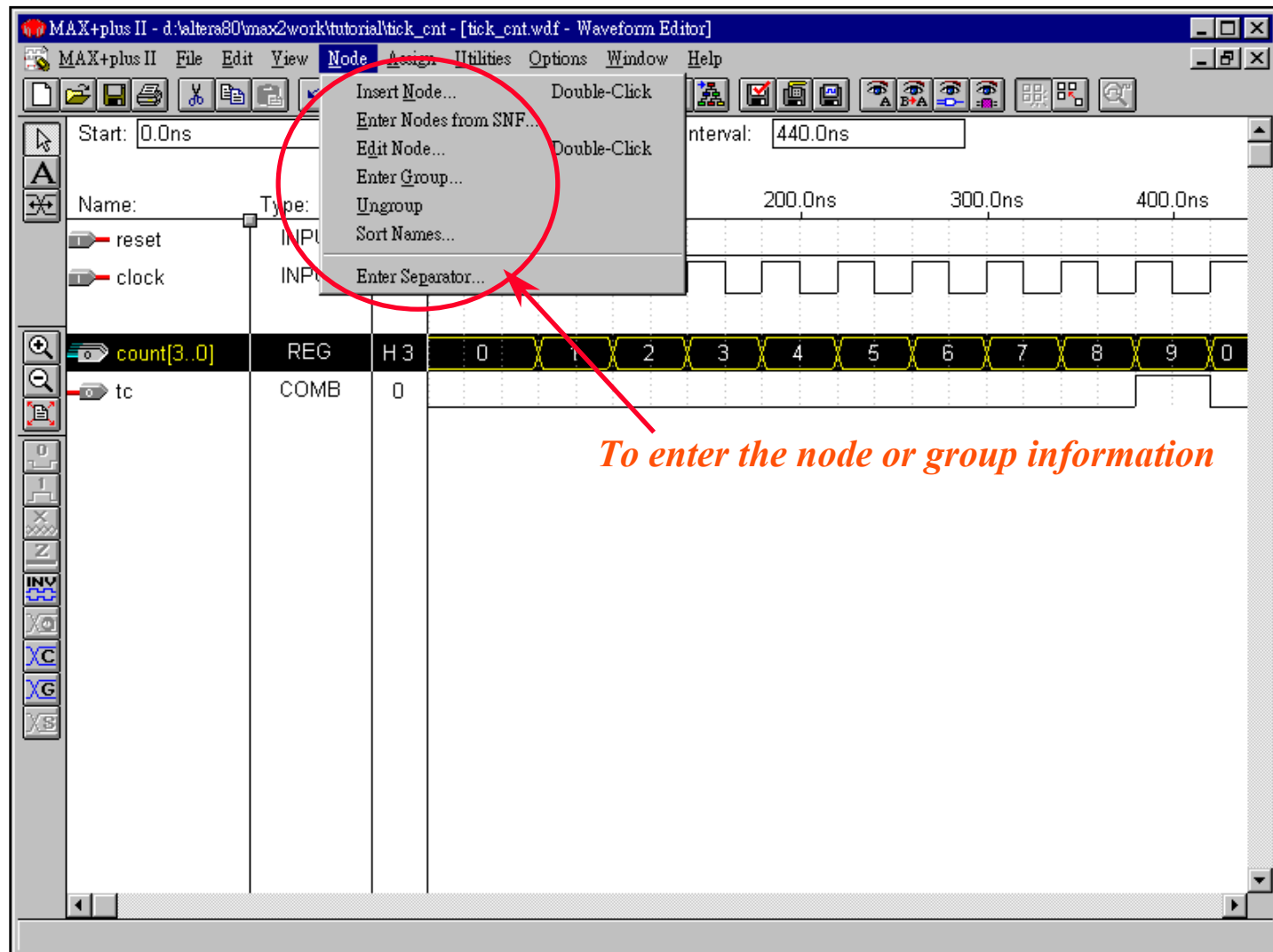


# File Menu



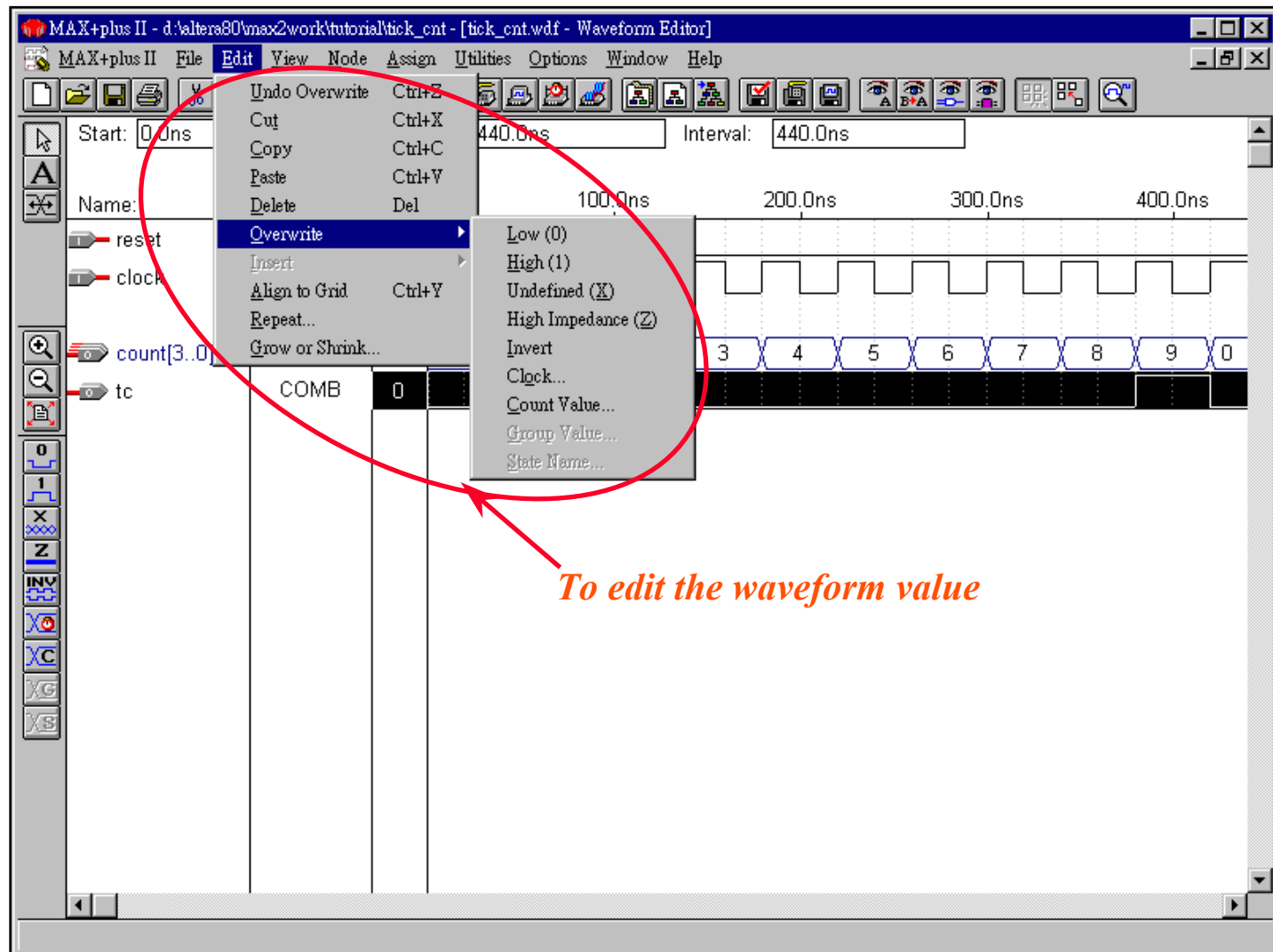


# Node Menu





# 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*



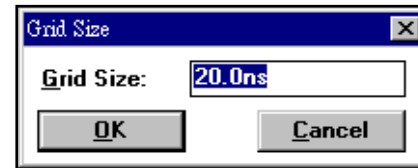
# Setting Waveform Editor Options

## ◆ Set the grid size & show the grid

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

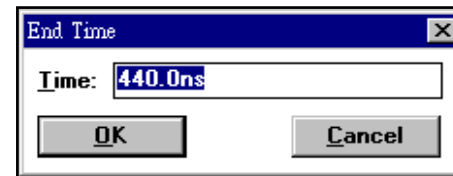
Menu: *Options -> Show Grid*

- 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

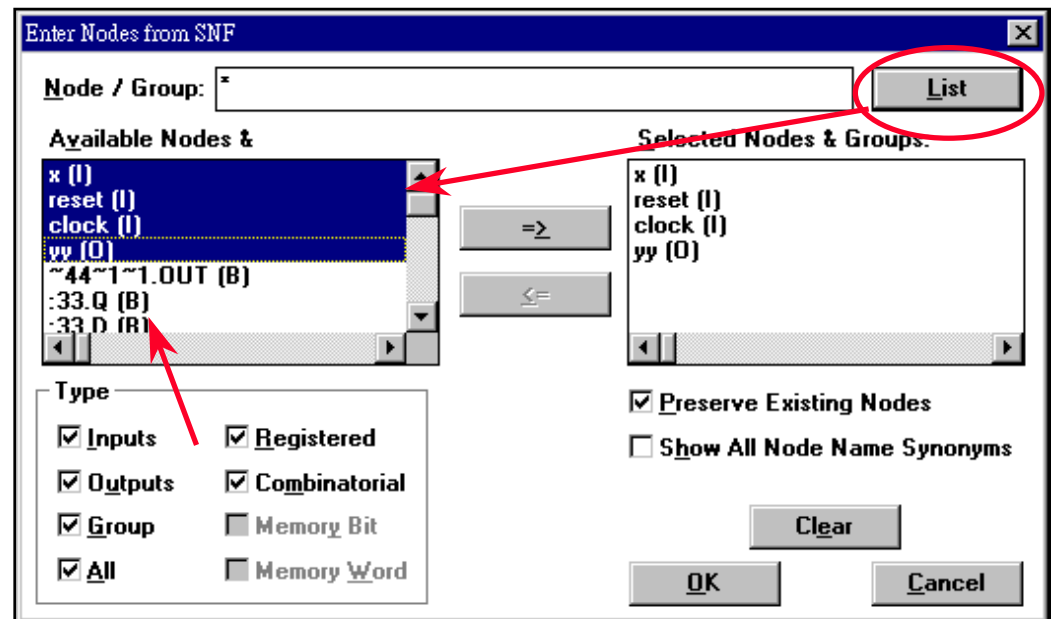


# Entering Nodes from SNF

## ◆ 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)

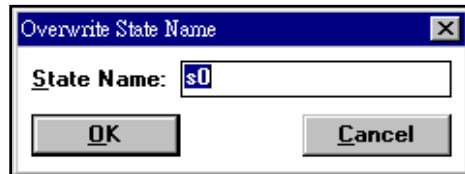
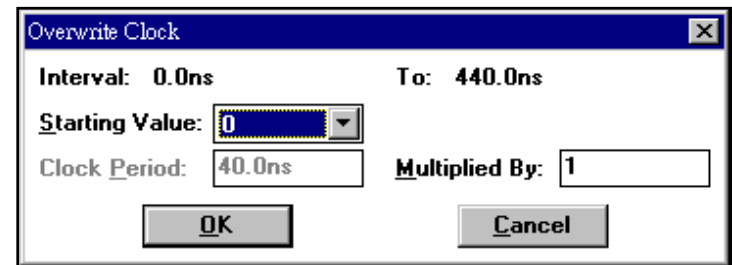
## ◆ Edit the waveforms

- 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)

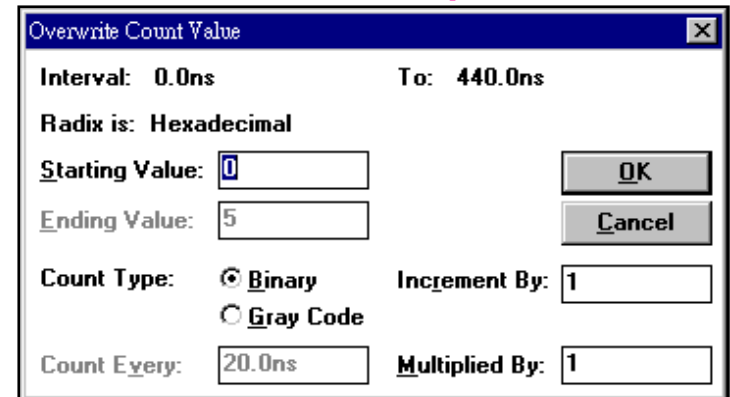
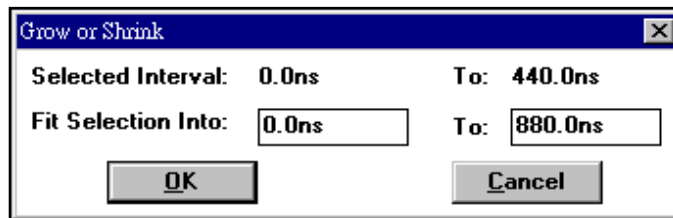
## ◆ Edit the waveforms

- 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...

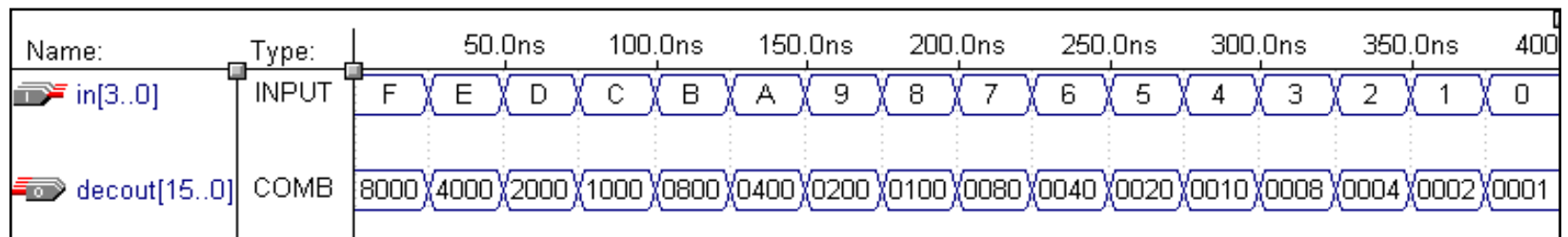
- WDFs cannot be at intermediate levels of a hierarchy
- Include all possible combinations of input values
- Align all logic level and state name transition
- Assume a 0ns propagation delay for all logic
- Assume a 0.1ns setup time and 0ns hold time for state machine node
- For clarity, Altera recommends 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

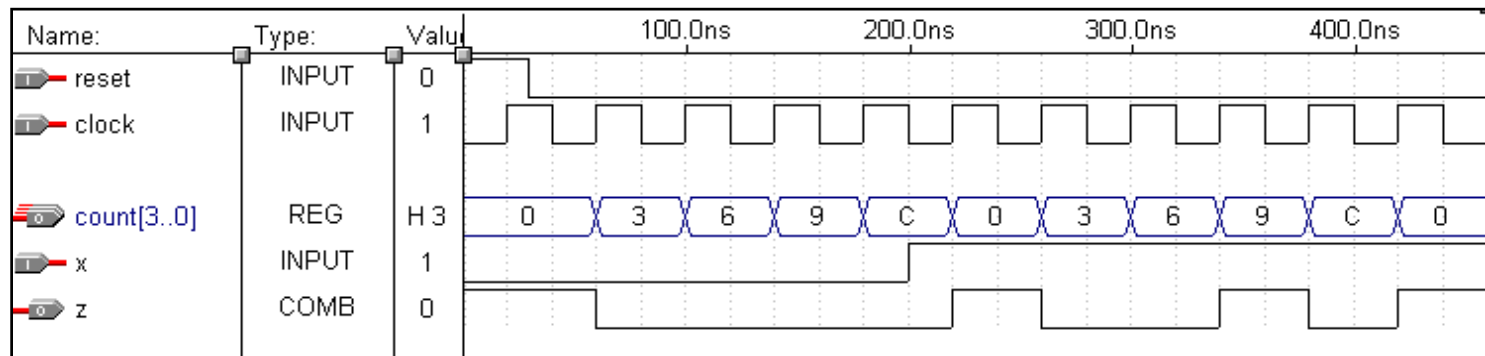
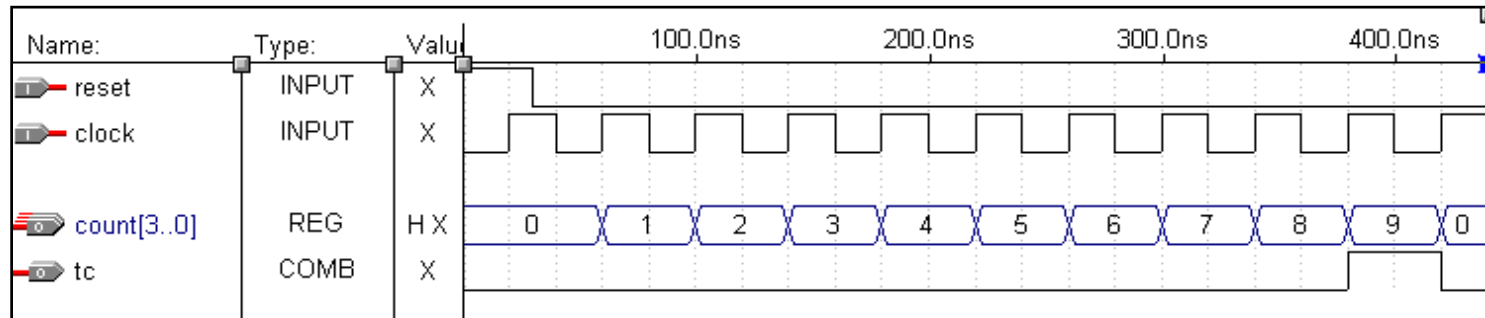




# Example: Counter

## ◆ When design a counter

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

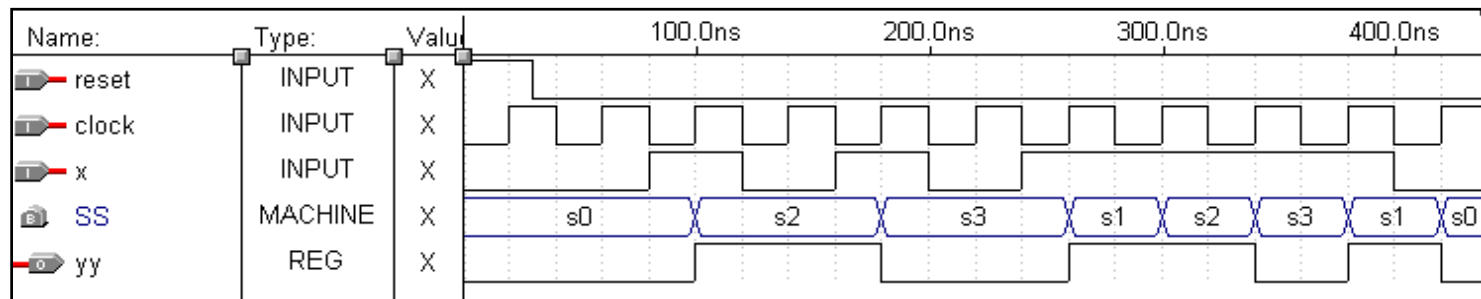




# Example: State Machine

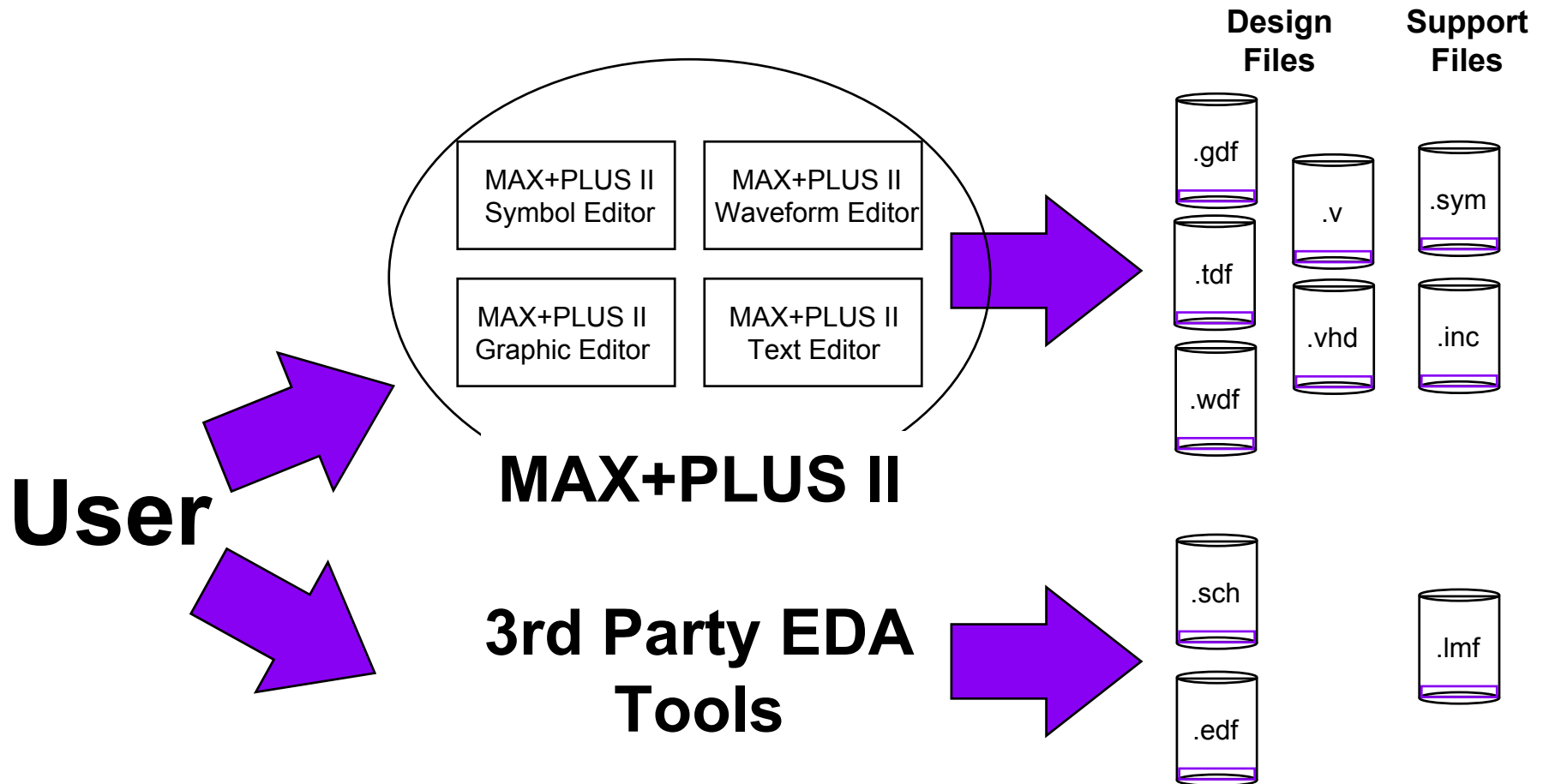
## ◆ When design a state machine

- Use “Overwrite State Name” to help create a state machine output
  - You can specify machine values with the state names instead of logic values
- Make sure all possible combinations of inputs and states are included





# Design Entry Summary

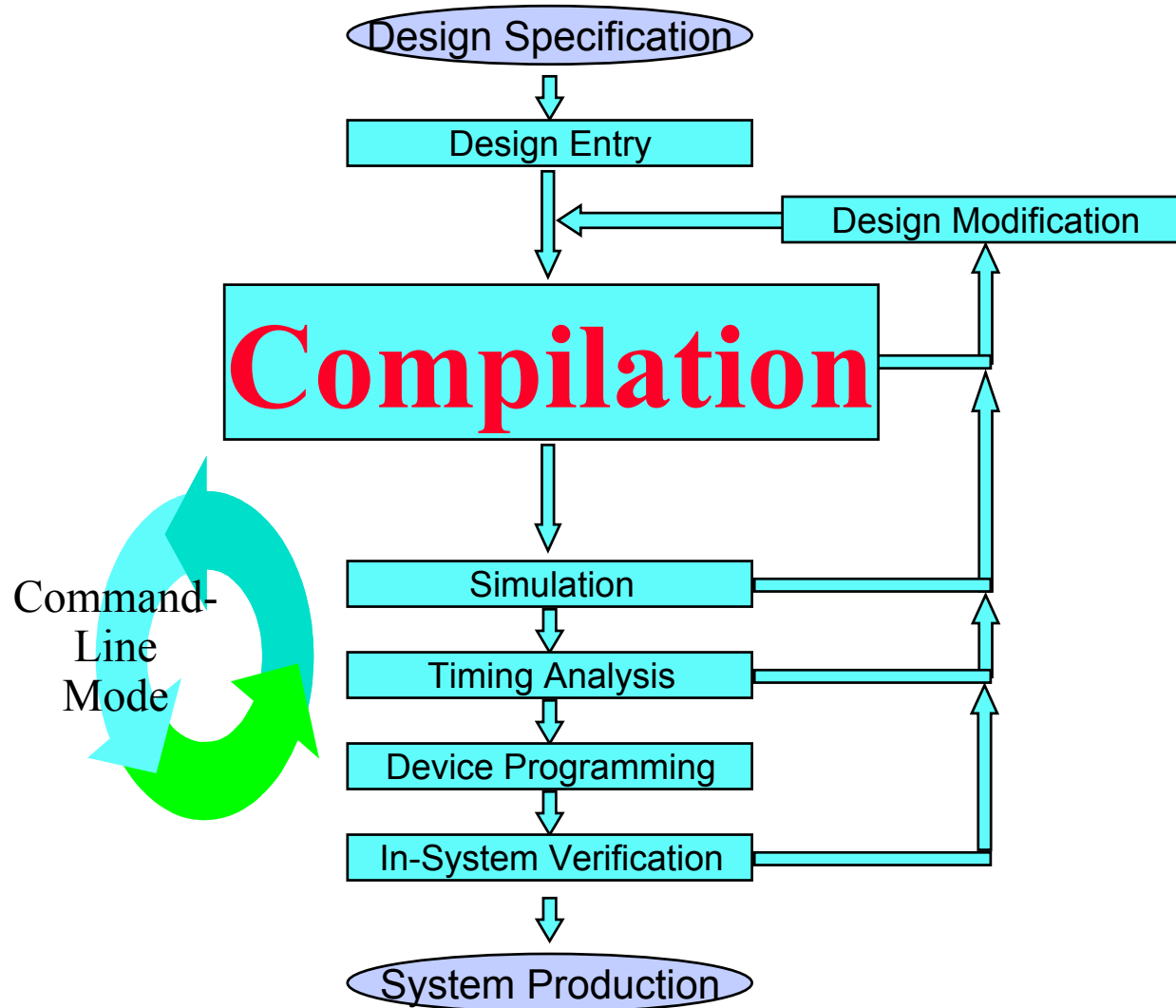




# Design Implementation

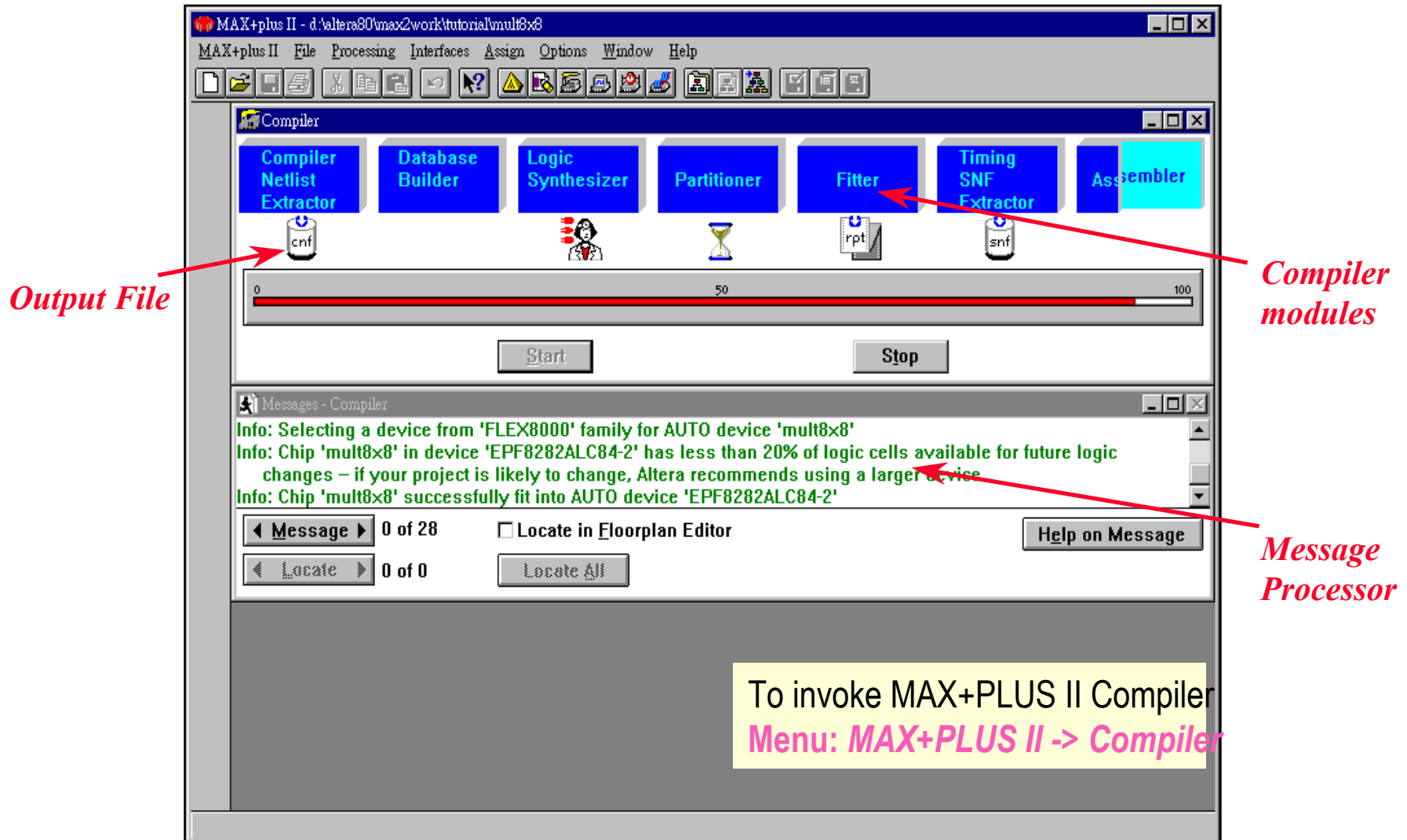
- ◆ MAX+PLUS II Compiler
- ◆ Preparing for Compilation
- ◆ Compiling the Project
- ◆ Analyzing the Compilation Results
- ◆ Floorplan Editor
- ◆ Appendix: Interfacing with 3rd-Party Tools







# MAX+PLUS II Compiler Window





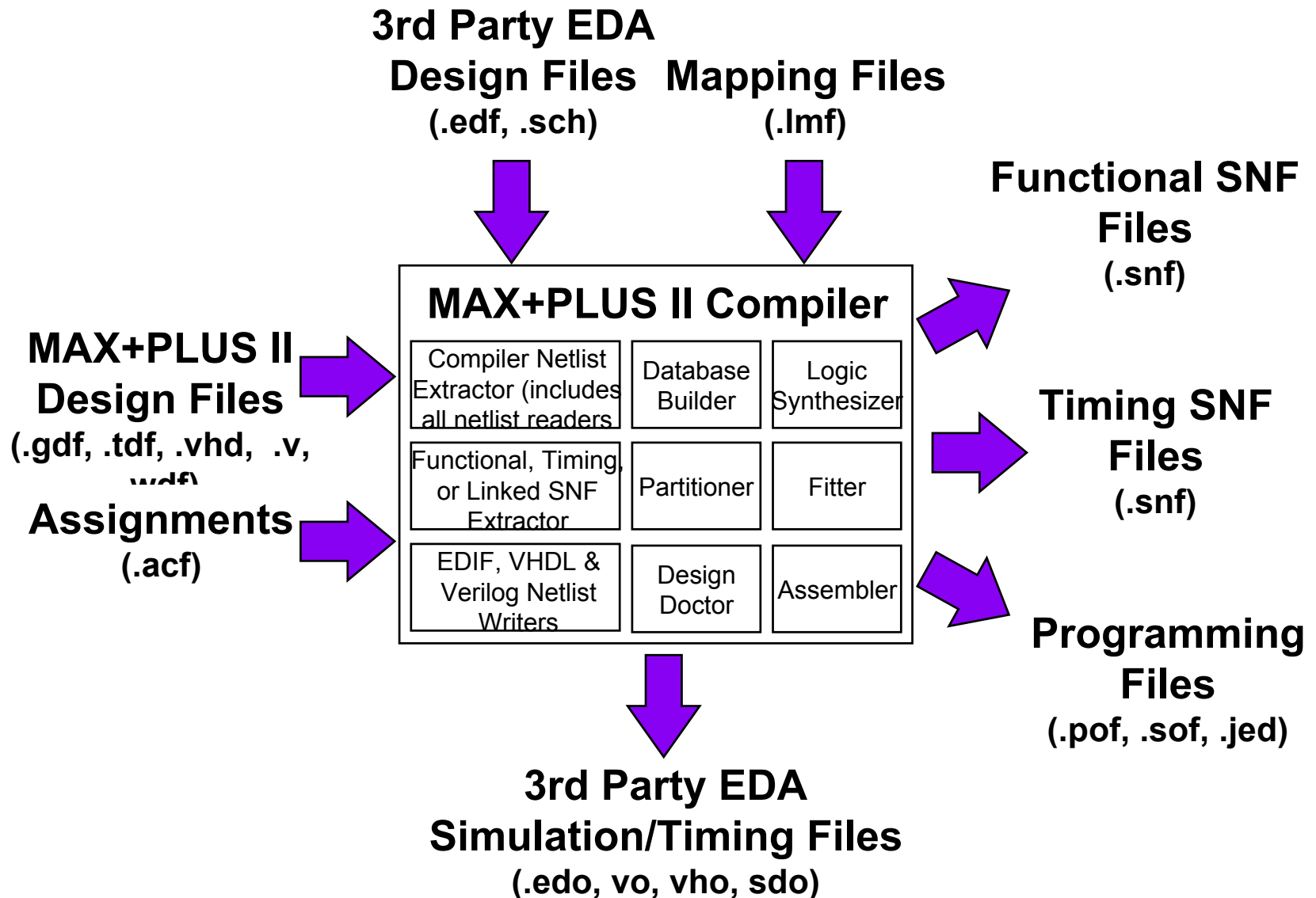
# MAX+PLUS II Compiler

- ◆ **Process all design files associated with the project**
  - Files can be created with MAX+PLUS II or 3rd party EDA Tools
- ◆ **Checks for syntax errors and common design pitfalls**
- ◆ **Performs logic synthesis and place & route**
  - According to assignments in .acf file
- ◆ **Generates files for simulation and timing analysis**
  - Files can be used by MAX+PLUS II or 3rd party EDA Tools
- ◆ **Generates files for programming targeted devices**





# Compiler Input and Output Files





# Compiler Input Files

## ◆ Design files

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

## ◆ Assignment and Configuration File (.acf)

- Controls the Compiler's synthesis and place & route operations
- Automatically generated when user enter assignments
- Automatically updated when user changes assignments or back-annotates project



# Compiler Output Files

## ◆ Design verification files

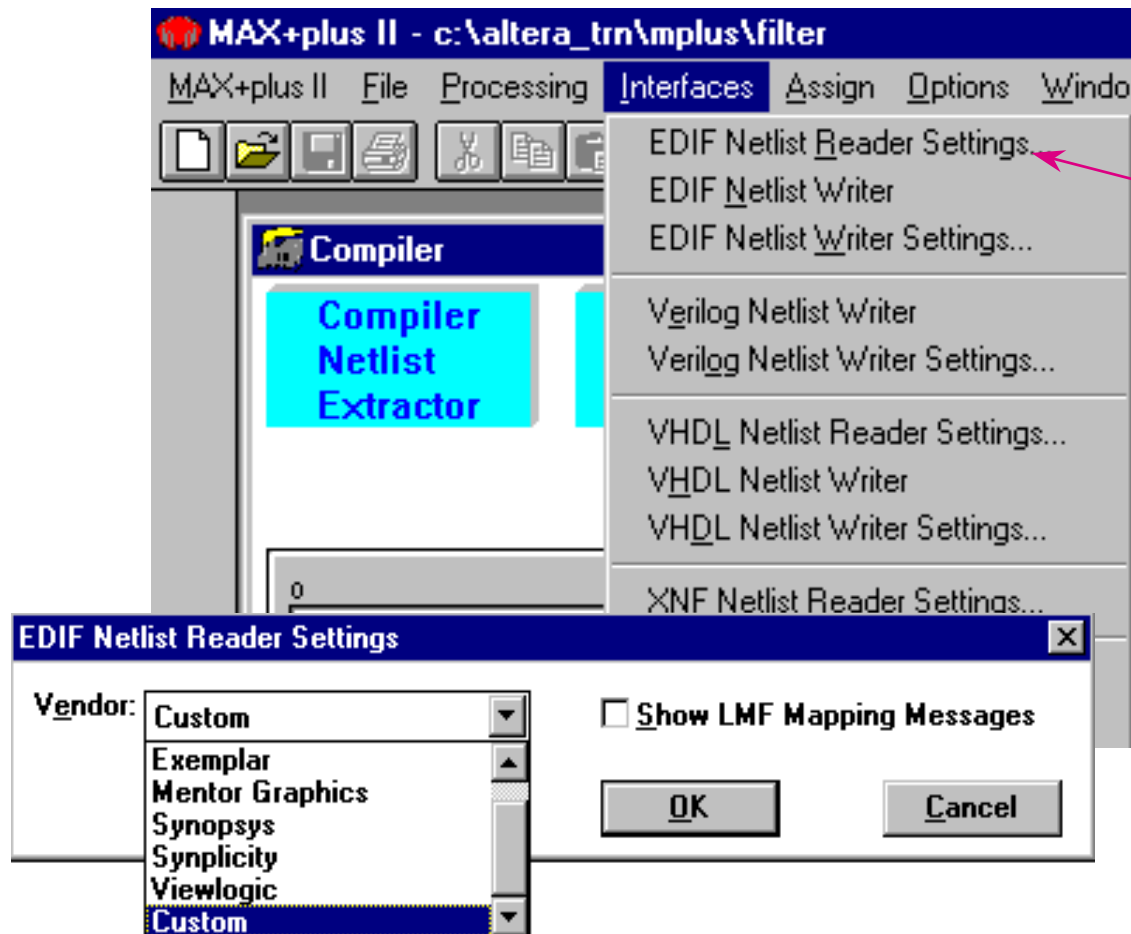
- MAX+PLUS II
  - Simulation Netlist File (.snf)
- 3rd Party EDA Tools
  - VHDL netlist file (.vho)
  - EDIF netlist file (.edo)
  - Verilog netlist file (.vo)
  - Standard Delay Format SDF file (.sdo)

## ◆ Programming files

- Programmer Object file (.pof)
- SRAM Object file (.sof)
- JEDEC file (.jed)



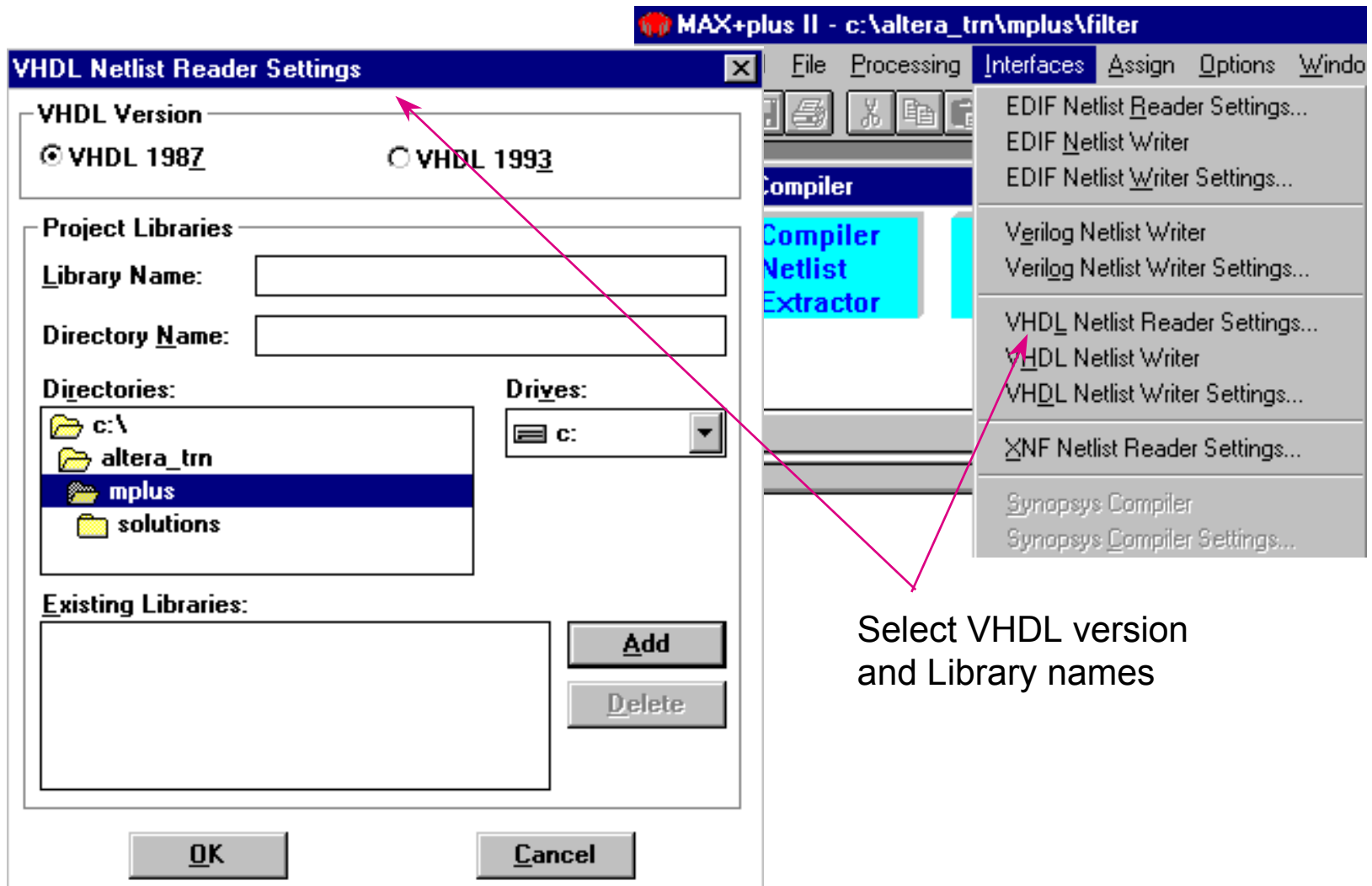
# For EDIF Netlist Input



For EDIF input, the  
EDIF Reader Settings  
need to be selected

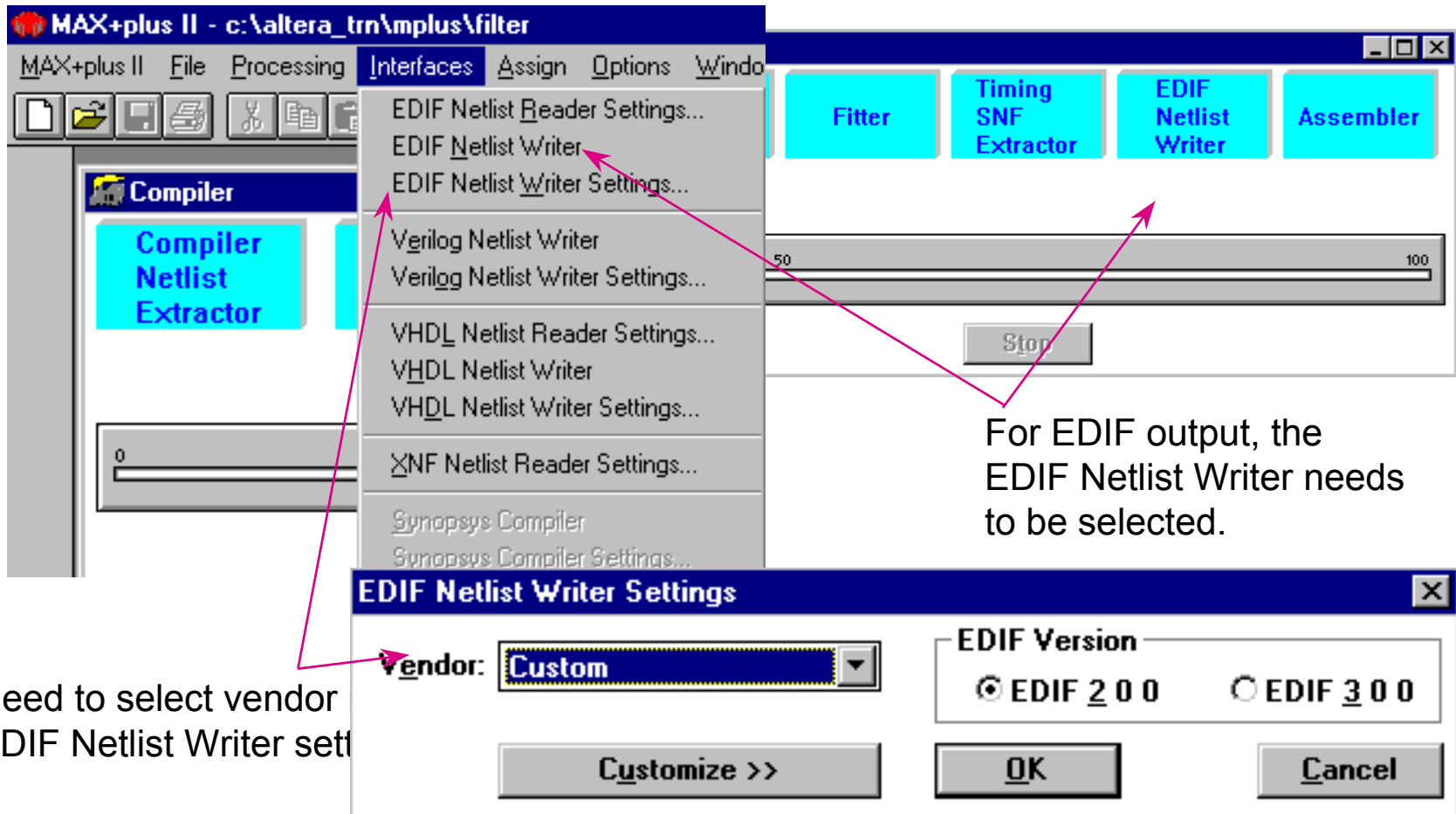


# VHDL Netlist Reader Settings



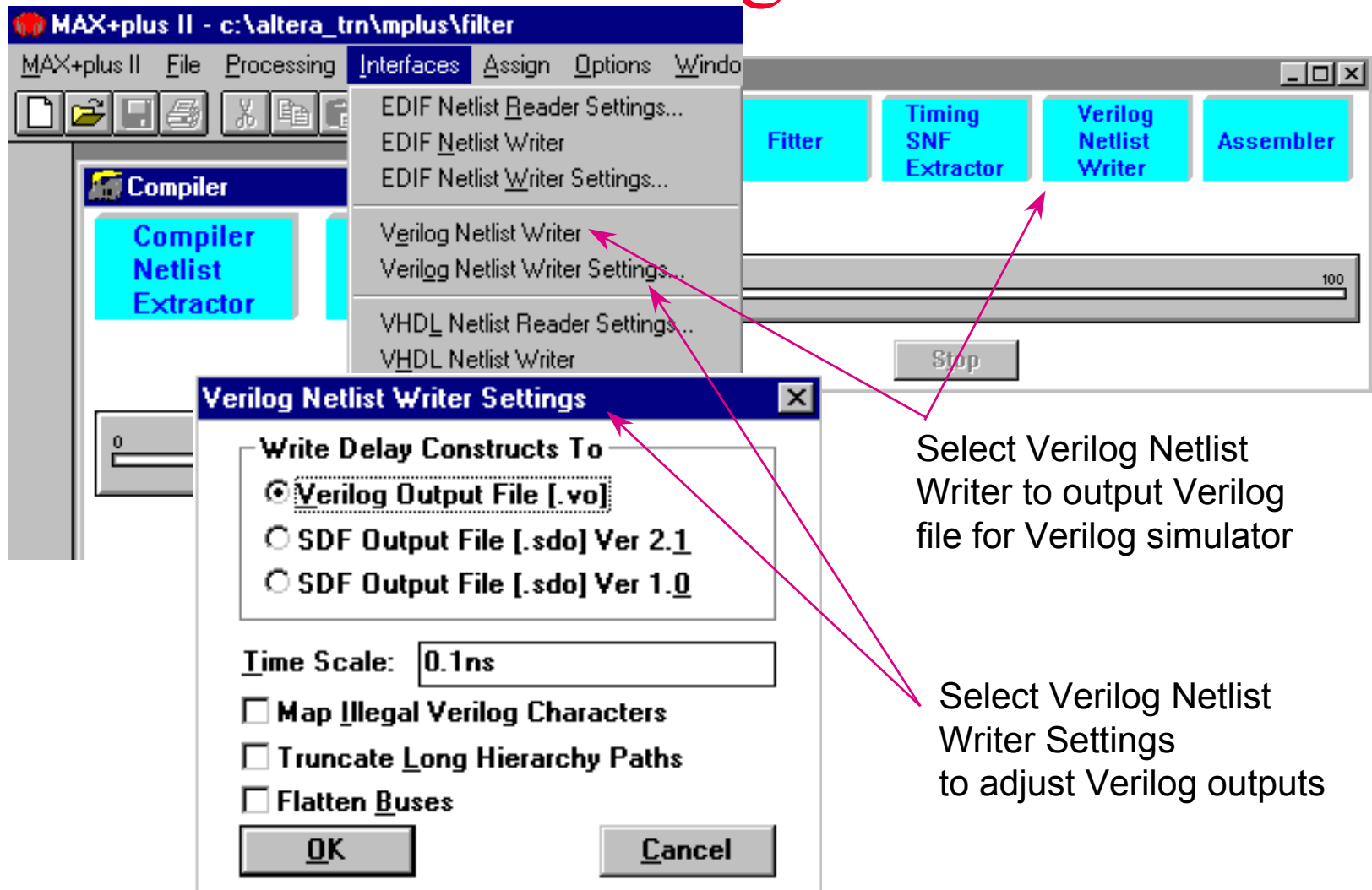


# For EDIF Netlist Output





# Verilog Netlist Writer & Writer Settings





# VHDL Netlist Writer & Writer Settings

The screenshot shows the MAX+plus II interface. The 'Interfaces' menu is open, showing options like 'EDIF Netlist Reader Settings...', 'EDIF Netlist Writer', 'EDIF Netlist Writer Settings...', 'Verilog Netlist Writer', 'Verilog Netlist Writer Settings...', 'VHDL Netlist Reader Settings...', 'VHDL Netlist Writer', and 'VHDL Netlist Writer Settings...'. A pink arrow points from the 'VHDL Netlist Writer Settings...' option to the 'VHDL Netlist Writer Settings' dialog box. Another pink arrow points from the 'VHDL Netlist Writer' option in the menu to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A third pink arrow points from the 'VHDL Netlist Writer' button to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A fourth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A fifth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A sixth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A seventh pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. An eighth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A ninth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab. A tenth pink arrow points from the 'VHDL Netlist Writer' button in the 'Compiler' tab to the 'VHDL Netlist Writer' button in the 'Compiler' tab.

Select VHDL Netlist Writer Settings to adjust the VHDL output

Select VHDL Netlist Writer to output VHDL file for VHDL simulator

**VHDL Netlist Writer Settings**

**VHDL Version**

☒ VHDL 1987 ☐ VHDL 1993

**Write Delay Constructs To**

☒ VHDL Output File [.vho]  
☐ SDF Output File [.sdo] Ver 2.1 (VITAL)  
☐ SDF Output File [.sdo] Ver 1.0

☐ Generate Configuration Declaration  
☐ Truncate Long Hierarchy Paths  
☐ Flatten Buses

**OK** **Cancel**



# Imported Design

## ◆ Top-level Design: can be read in directly

- EDIF Netlist files
- OrCAD schematics
- Refer to MAX+PLUS II Read Me file for the version of 3rd Parties tools it interface with

## ◆ Lower-level modules

- EDIF, OrCAD schematics files
  - Create symbols or files to instantiate component
- Other proprietary files
  - JEDEC, ABEL, PALASM
  - Conversion utilities exist in Altera ftp site



# Compiler Modules - (1)

## ◆ Compiler Netlist Extractor

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



# Compiler Modules - (2)

## ◆ Database Builder

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



# Compiler Modules - (3)

## ◆ Logic Synthesizer

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

## ◆ Design Doctor Utility

- The Compiler utility that checks each design file in a project for poor design practices that may cause reliability problems when the project is implemented in one or more devices



# Compiler Modules - (4)

## ◆ Partitioner

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



# Compiler Modules - (5)

## ◆ Fitter

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



# Compiler Modules - (6)

## ◆ SNF(Simulation Netlist File) Extractor

- Functional SNF Extractor
  - The Compiler module that creates a functional SNF containing the logic information required for functional simulation.
  - Since the functional SNF is created before logic synthesis, partitioning, and fitting are performed, it includes all nodes in the original design files for the project
- Timing SNF Extractor
  - The Compiler module that creates a timing SNF containing the logic and timing information required for timing simulation, delay prediction, and timing analysis
  - The timing SNF describes a project as a whole. Neither timing simulation nor functional testing is available for individual devices in a multi-device project.
- Linked SNF Extractor
  - The Compiler module that creates a linked SNF containing timing and/or functional information for several projects
  - A linked SNF of a super-project combines the timing and/or functional information for each project, allowing you to perform a board-level simulation



# 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)

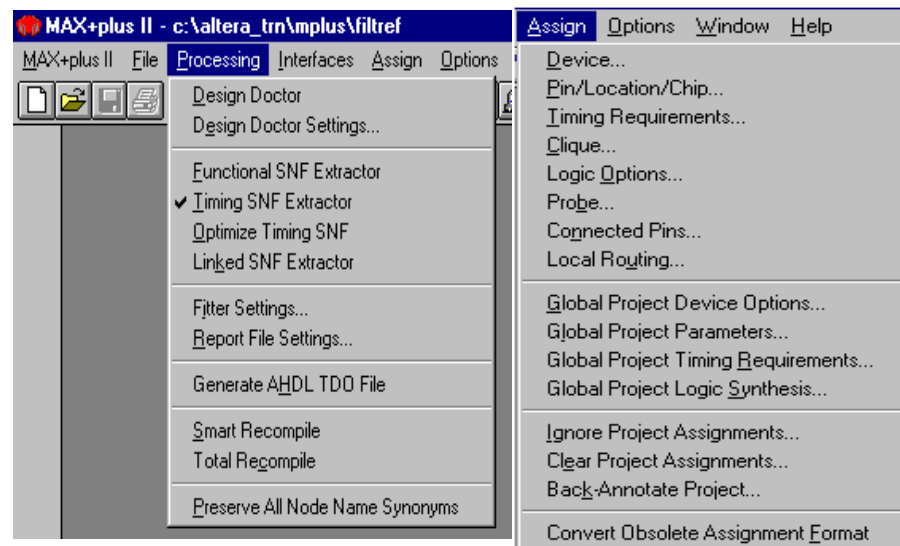
## ◆ Assembler

- 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



# Compiling a Project

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





# The Functional Compilation Process

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





# The Timing Compilation Process

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

■ Ass



device



# Compiler Processing Options

## ◆ Functional

- Compilation generates file for Functional Simulation
  - Functional SNF file (.snf)

## ◆ Timing

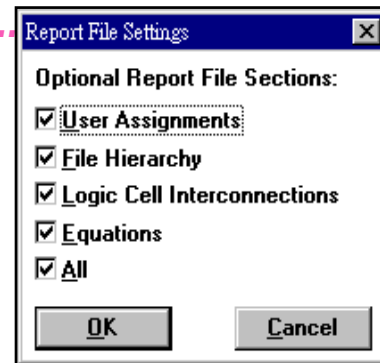
- Compilation generates user selectable files for
  - Timing Simulation and Timing Analysis
    - Timing SNF file (.snf)
  - 3rd party EDA Simulation
    - Verilog file (.vo)
    - VHDL file (.vho)
    - SDF file (.sdo)
  - Device Programming
    - Altera Programmer file (e.g. .pof, .sof)



# Compilation Process Settings - (6)

## ◆ Customize the report file settings

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





# Compilation Process Settings - (7)

## ◆ “Smart Recompile” & “Total Recompile”

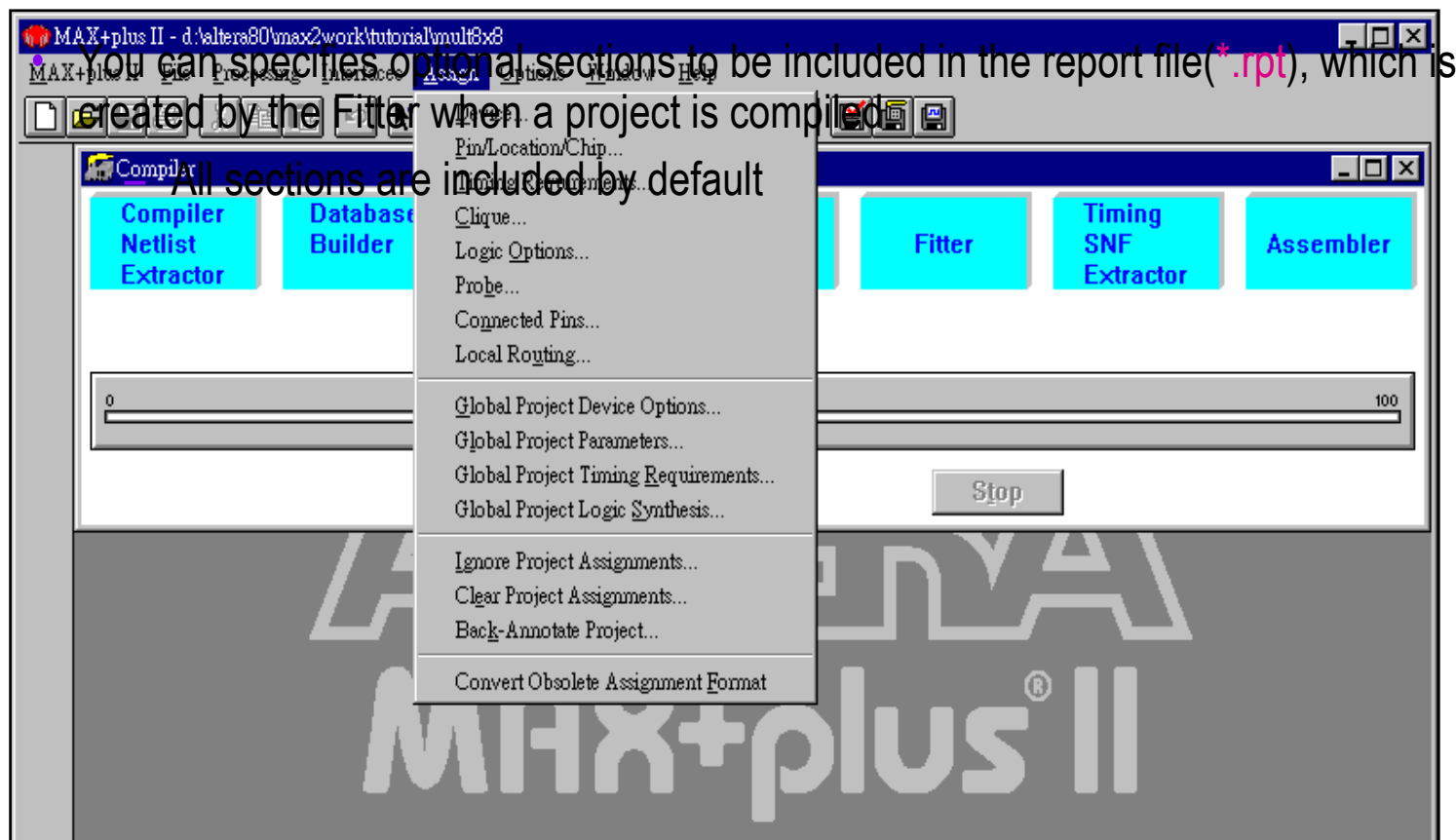
- The first time the Compiler processes a project, all design files of that project are compiled
- Use “Smart Recompile” feature to create an expanded project database that helps to accelerate subsequent compilations
  - Allow you to change physical device resource assignments without rebuilding the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

*Menu: Processing -> Smart Recompile*

*Menu: Processing -> Total Recompile*



# Assign Menu





# Assignments Control

## ◆ Device FIT

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

## ◆ Device Utilization

- Circuit design
- Logic assignment

## ◆ Performance

- Circuit design
- Logic assignments
- Logic placements



# Assignments

## ◆ Most common Assignments

- Device assignments
- Pin assignments

## ◆ Other assignments

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

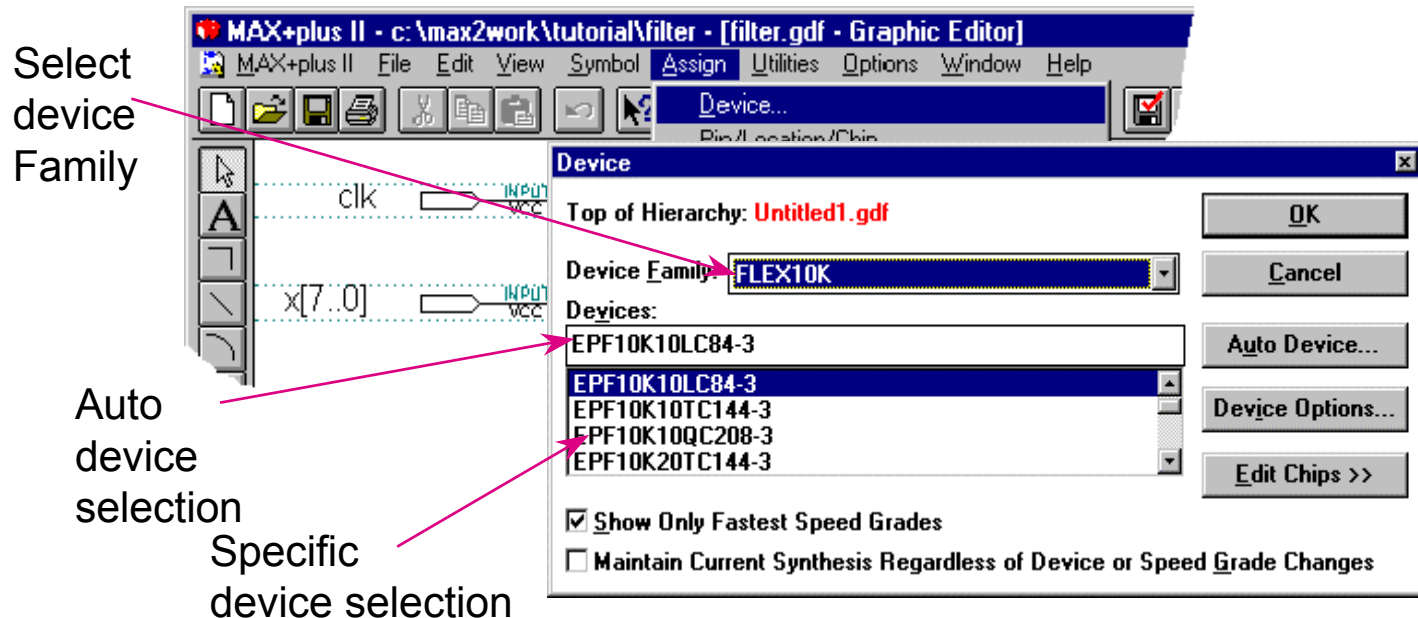




# Making Device Assignment

## ◆ Select Device

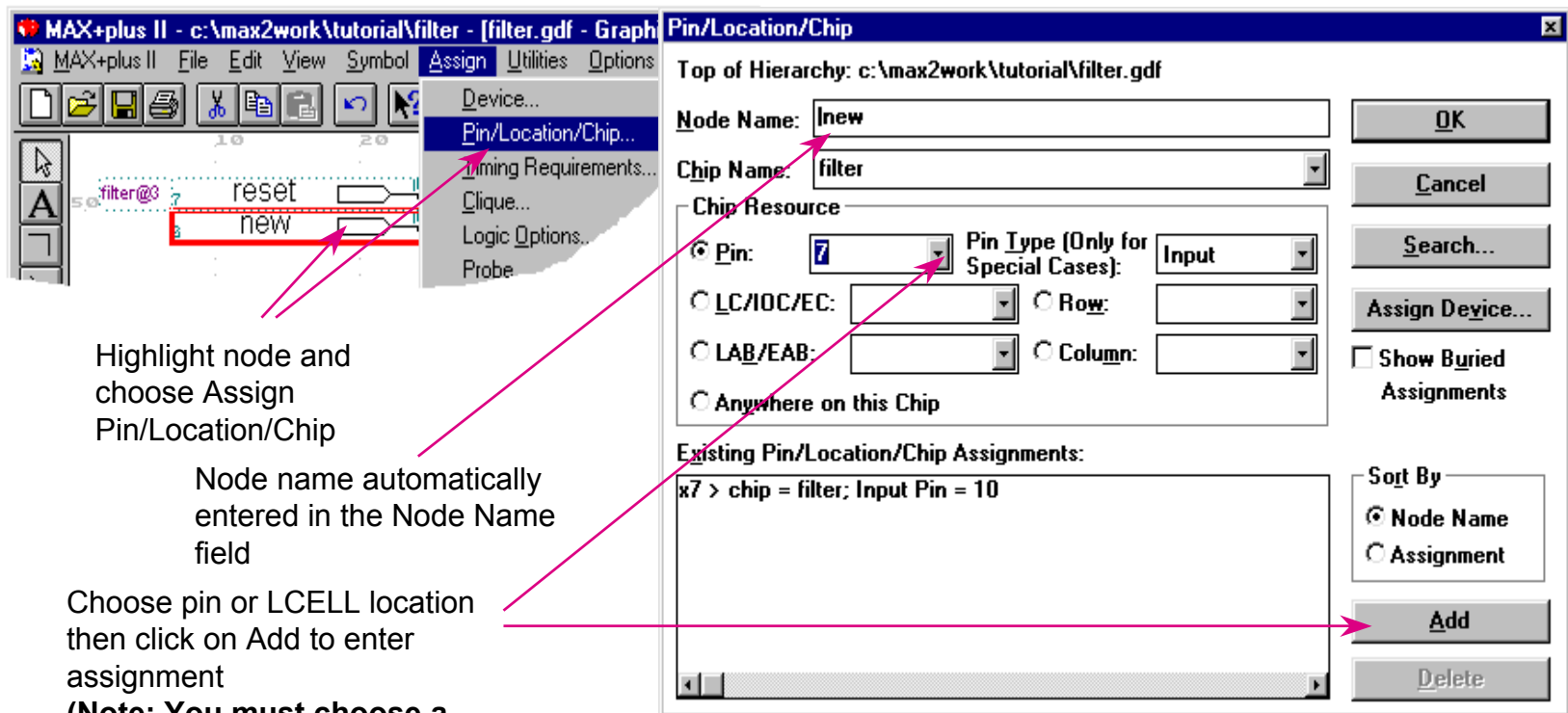
- Specific device
- Auto
  - MAX+PLUS II chooses smallest and fastest device the design fits into





# Making Pin Assignment

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





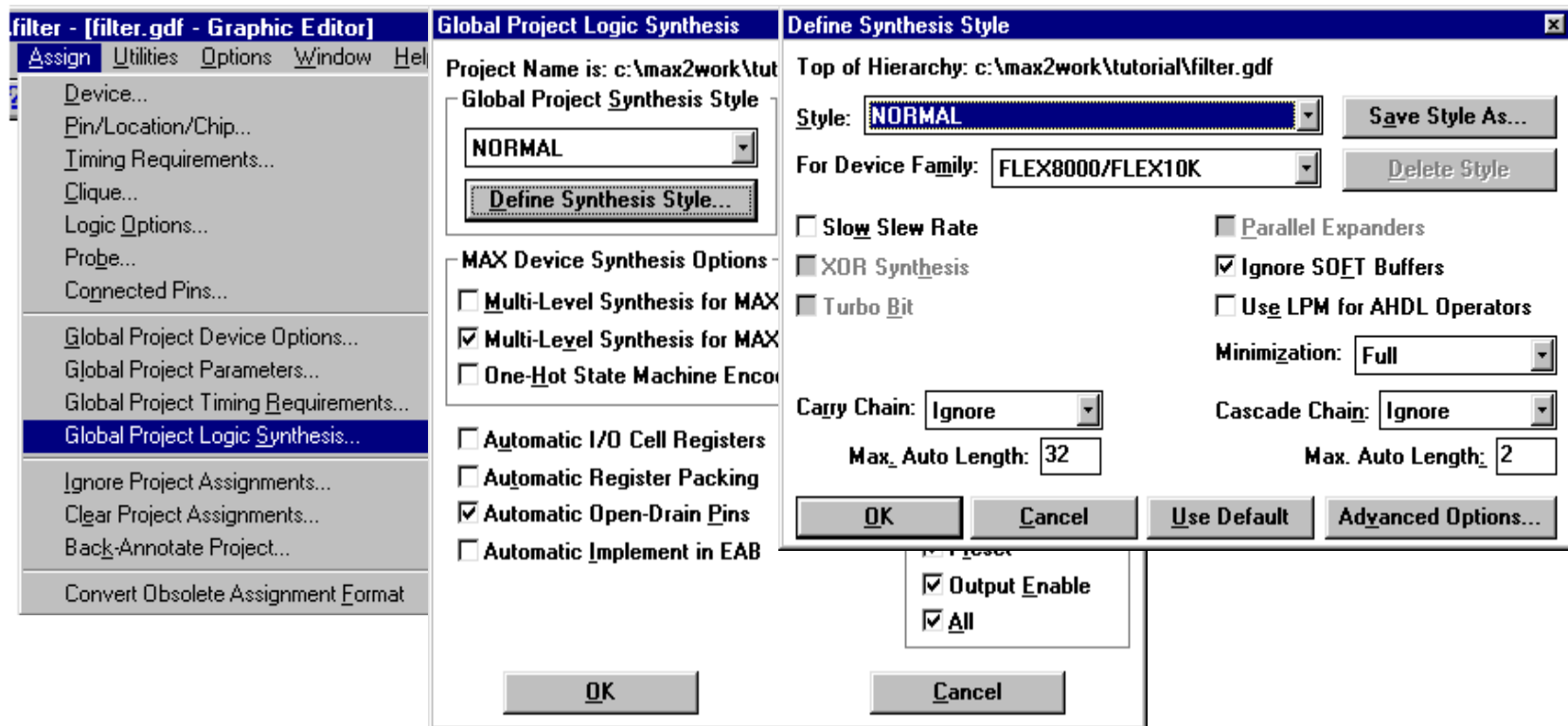
# Logic Synthesis Style

- ◆ The most common way toward adjusting these assignments is to apply the predefined Logic Synthesis Style toward the different portion of your design:
  - Normal
  - Fast
  - WYSIWYG
  
- ◆ Each of the Logic Synthesis Styles is a collection of both logic synthesis options and individual architectural settings



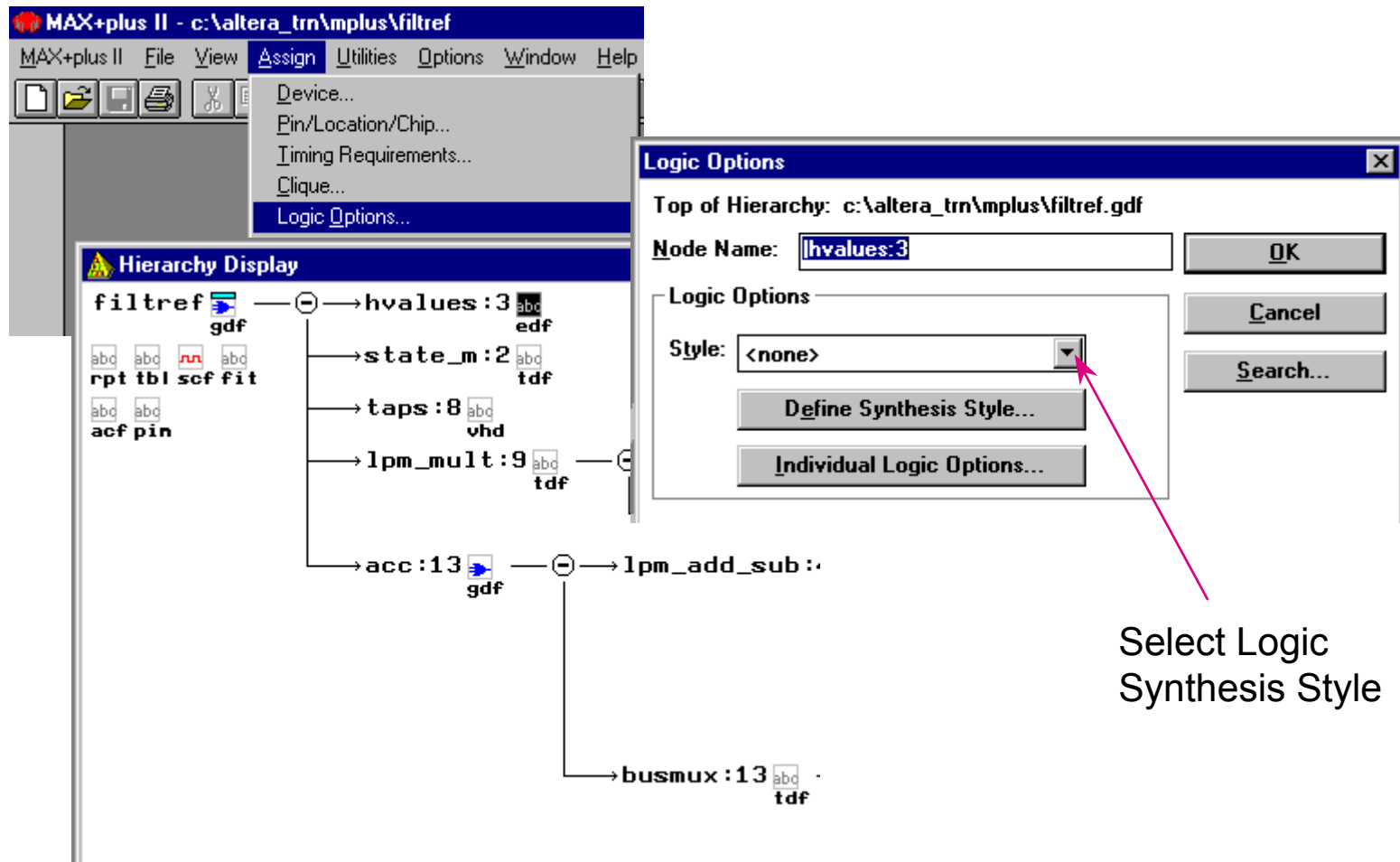
# Global Project Logic Synthesis Style

- ◆ Choose Assign then Global Project Logic Synthesis
- ◆ Select from predefined synthesis style
  - NORMAL (default), FAST or WYSIWYG
- ◆ Or create user tailored settings





# Assign Logic Synthesis Style Locally





# Individual Logic Option Assignment

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

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

Individual Logic Options

Top of Hierarchy: c:\altera\_trn\mplus\filtref.gdf

<input checked="" type="checkbox"/> Slow Slew Rate	<input checked="" type="checkbox"/> Parallel Expanders
<input checked="" type="checkbox"/> XOR Synthesis	<input checked="" type="checkbox"/> Ignore SOFT Buffers
<input checked="" type="checkbox"/> Turbo Bit	<input checked="" type="checkbox"/> Use LPM for AHDL Operators
<input checked="" type="checkbox"/> Hierarchical Synthesis	<input checked="" type="checkbox"/> Implement in EAB
<input checked="" type="checkbox"/> Implement as Output of Logic Cell	<input checked="" type="checkbox"/> Fast I/O
<input checked="" type="checkbox"/> Insert Additional Logic Cell	<input checked="" type="checkbox"/> Global Signal
<input checked="" type="checkbox"/> Increase Input Delay	<input checked="" type="checkbox"/> Disable Fast Feedback Path

CLKLOCKx1 Input Freq (MHz):

Carry Chain:

Minimization:

Cascade Chain:

Max. Auto Length:

Max. Auto Length:

OK Cancel Use Default Advanced Options...

Advanced Options

Top of Hierarchy: c:\max2work\tutorial\filter.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



# Location assignments

The screenshot illustrates the process of assigning locations in the Altera Manager. It features three main components:

- Manager - c:\altera\_trn\implus\filter**: The main application window with a menu bar (Assign, Options, Help) and a list of actions: Device..., Pin/Location/Chip..., Timing Requirements..., Clique..., and Logic Options....
- Hierarchy Display**: A tree view showing the project hierarchy. The selected node is `hvalues:3` under `filter`. Other nodes include `state_m:2`, `taps:8`, `lpm_mult:9`, and `acc:13`.
- Pin/Location/Chip**: A dialog box for assigning resources. It shows the top of hierarchy as `c:\altera_trn\implus\solutions\filter.gdf` and the node name as `hvalues:3`. The chip name is `filter`.

Annotations with pink arrows indicate the workflow:

- An arrow points from the **Pin/Location/Chip...** menu item to the **Pin/Location/Chip** dialog box, labeled "Select Pin/Location/Chip ...".
- An arrow points from the `hvalues:3` node in the Hierarchy Display to the **Pin/Location/Chip** dialog box, labeled "Select Location".

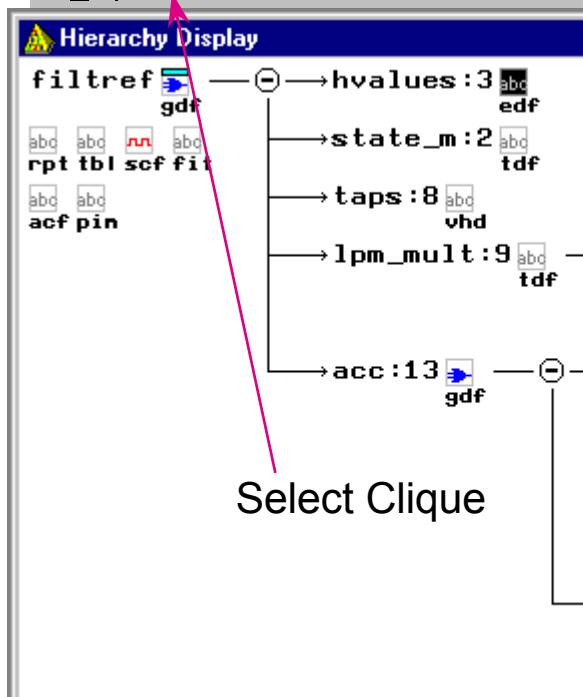
The **Pin/Location/Chip** dialog box includes the following fields and options:

- Top of Hierarchy**: `c:\altera_trn\implus\solutions\filter.gdf`
- Node Name**: `hvalues:3`
- Chip Name**: `filter`
- Chip Resource**:
  - ☐ Pin: [dropdown] Pin Type (Only for Special Cases): [dropdown]
  - ☐ LC/IOC/EC: [dropdown] Row: [dropdown]
  - ☐ LAB/EAB: [dropdown] Column: [dropdown]
  - ☒ Anywhere on this Chip
- Existing Pin/Location/Chip Assignments**: A list box (currently empty).
- Buttons**: OK, Cancel, Search..., Assign Device..., Show Buried Assignments (checkbox), Sort By (Node Name, Assignment), Add, Delete.



# Clique Assignments

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



Clique

Top of Hierarchy: c:\altera\_trn\mplus\solutions\filter.gdf

Node Name: hvalues:3

Clique Name:

Existing Clique Assignments:

Enter clique name

Click **Add** to add assignment, click **OK** to close window

OK

Cancel

Search...

Sort By

- ☒ Node Name
- ☐ Clique Name

Add

Delete



# Timing Requirements Assignments

*FLEX devices only*

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

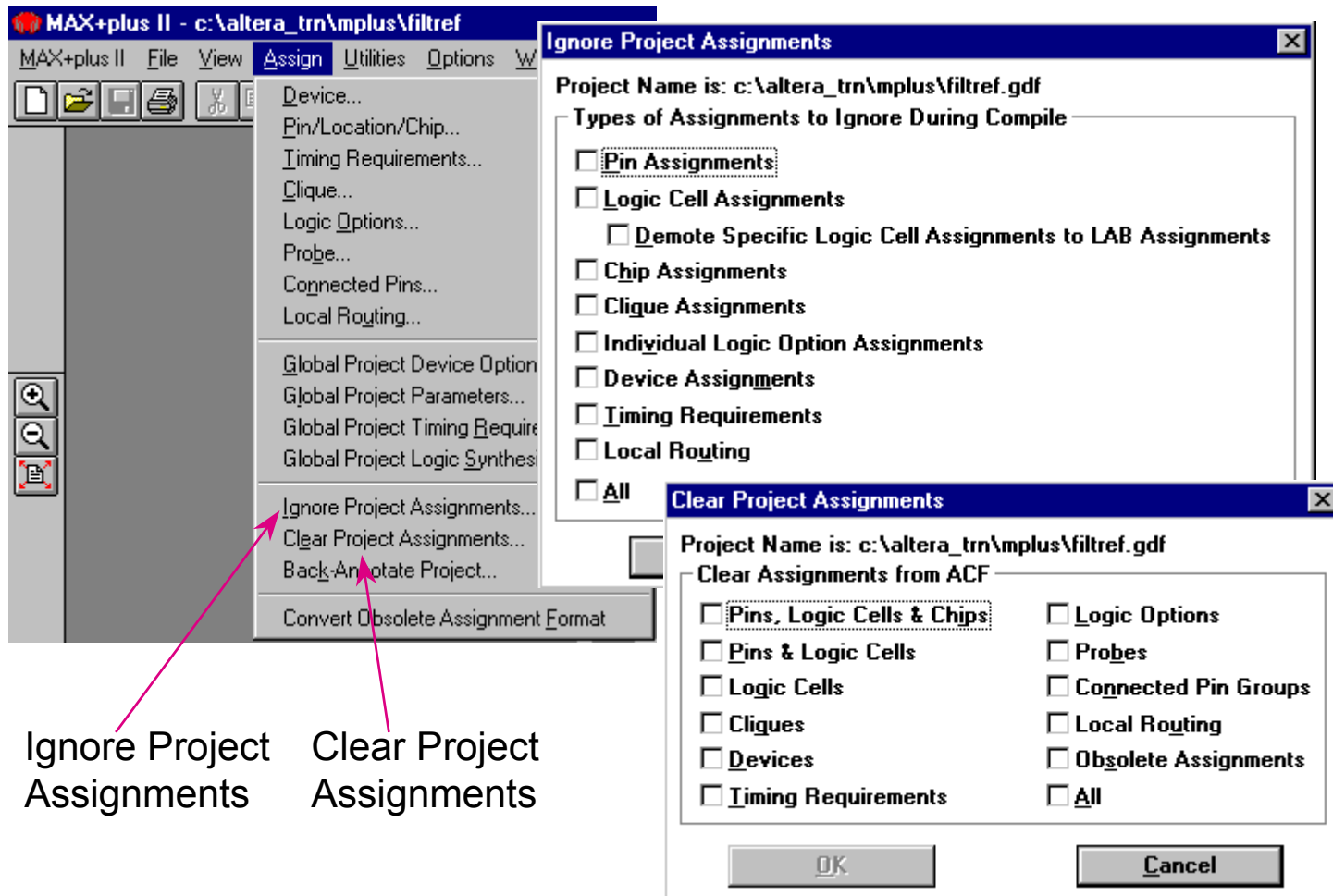


# Assignment Recommendation

- ◆ Start with device and pin assignments. Beware, your pin assignments might affect performance. Ideally, you should let MAX+PLUS II choose the pin assignments. If you have pin assignments, you might want to compile your design once without your pin assignments to see if they affect your performance.
- ◆ Compile design. Check device utilization and performance.
- ◆ If you need to adjust device utilization or performance try the other assignments. Try the synthesis style assignments first.
- ◆ Assignments can only be made to “hard” nodes or lower-level designs that contains hard nodes. Hard nodes are objects that translate directly into objects in silicon e.g. Flip-flops, LCELLs and I/O pins



# Ignore or Clear Assignments





# Global Project Device Options

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

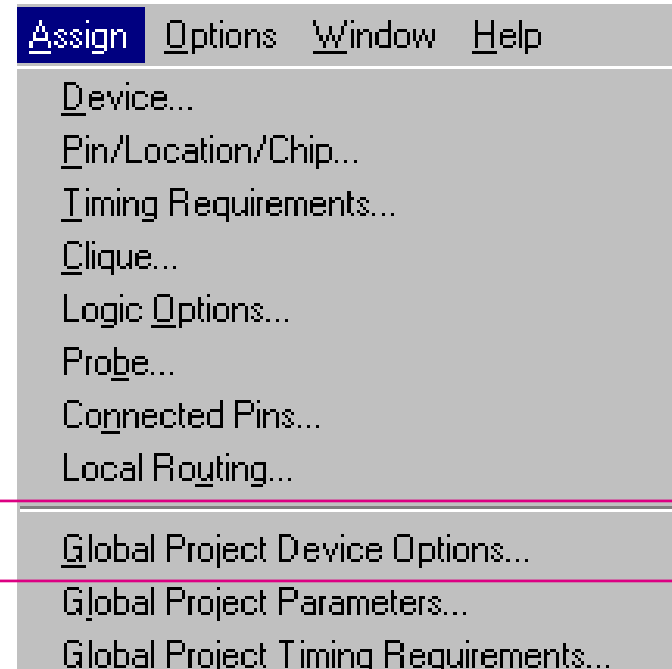
**For example,**

## **FLEX Device**

- configuration scheme
- multi-volt I/O

## **MAX Device**

- Enable JTAG support
- security bit





# More Compiler Processing Options

## ◆ Design Doctor

- Checks for common design errors

## ◆ Fitter Settings

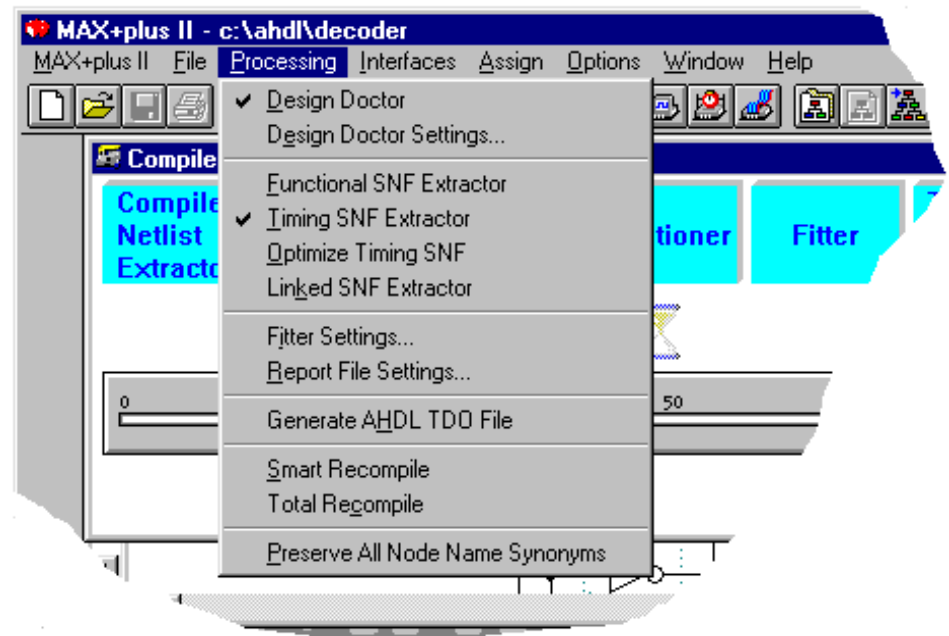
- Set place & route options

## ◆ Smart Recompile

- Faster compilation time

## ◆ Total Recompile

- Recompile every file





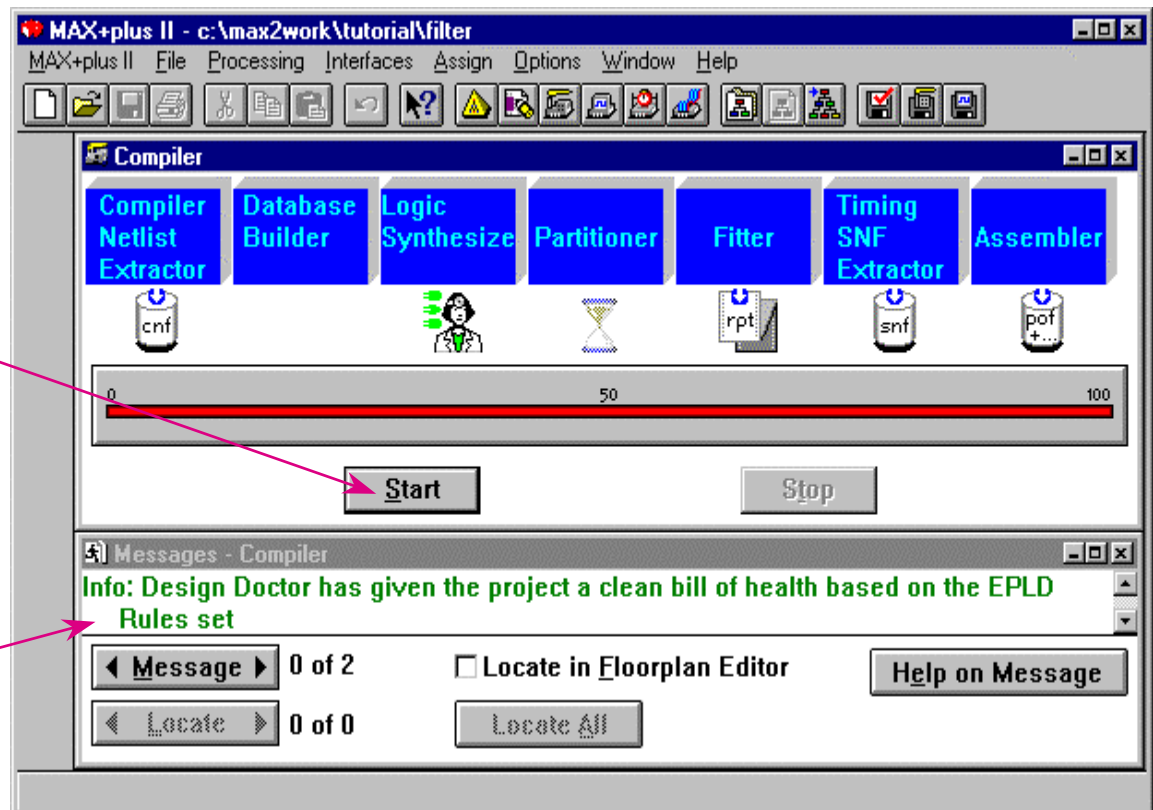
# Compile the Design

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

- Info
- Warning
- Error

Start  
Compilation

Messages





# The Report File

## ■ Project summary

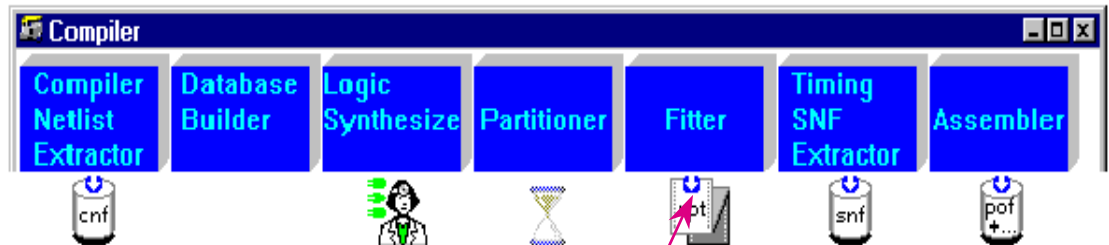
- Device assignments
- Error summary
- Device pin-out diagram (useful for PCB layout)

## ■ Resource utilization

- Pin
- LCELL
- Equations

## ■ Compiler resources

- Compilation time
- Memory usage



Open report file by  
double clicking on the  
rpt icon



# Checking the Messages

## ◆ Check the messages in Message Processor

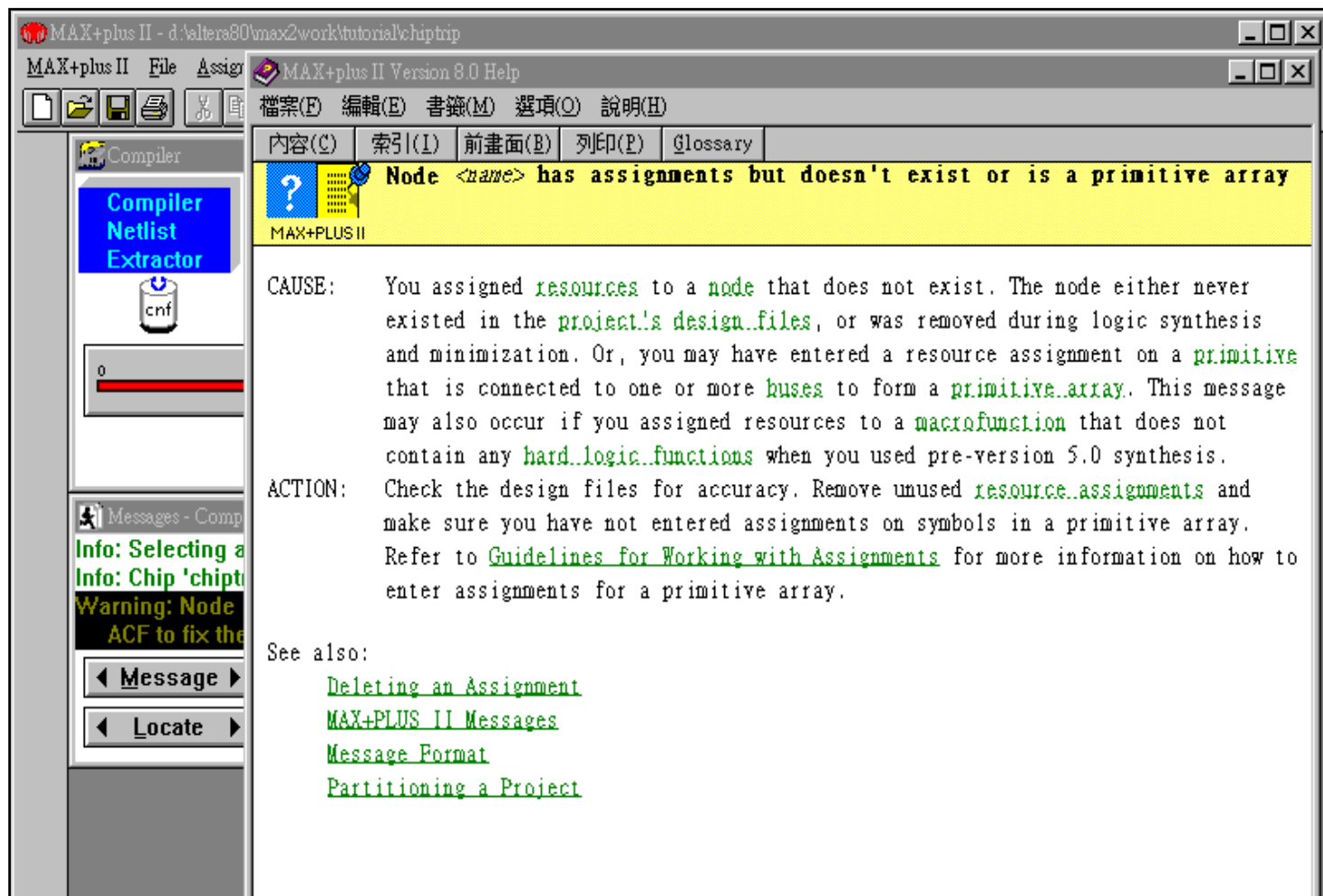
- In Message Processor window, choose the message and click the HELP on Message to understand the meaning of the message, its cause and the possible solutions (suggested actions)

## ◆ Error location

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



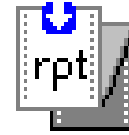
# Help on Message





# 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

The screenshot shows the MAX+plus II software interface. The main window is titled "MAX+plus II - d:\altera80\max2work\tutorial\chiptrip". 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 compilation. Below the toolbar is a "Compiler" window with several buttons: "Compiler Netlist Extractor", "Database Builder", "Logic Synthesizer", "Partitioner", "Filter", "Timing SNF Extractor", and "Assembler". Each button has a corresponding icon below it. The "chiptrip.rpt - Text Editor" window is open, displaying the following text:

```
** DEVICE SUMMARY **
```

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	LCs	% Utilized
chiptrip	EPF8282ALC84-2	6	13	0	80	38 %

User Pins: 6 13 0

\$

Project Information d:\altera80\max2work\tutorial\chiptrip.rpt

```
** PROJECT COMPILATION MESSAGES **
```

Warning: Ignored all pin assignments as requested in the Ignore Project Assignmer

Line 41 Col 49 INS

On the right side of the interface, there is a vertical panel with a progress bar at the top (labeled 100), a button labeled "edit the project's", and a button labeled "Help on Message".



# Pin-out file (.pin)

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

N.C. = Not Connected.

VCCINT = Dedicated power pin, which MUST be connected to VCC (5.0 volts).

VCCIO = Dedicated power pin, which MUST be connected to VCC (5.0 volts).

GNDINT = Dedicated ground pin or unused dedicated input, which MUST be connected to GND.

GNDIO = Dedicated ground pin, which MUST be connected to GND.

RESERVED = Unused I/O pin, which MUST be left unconnected.

-----  
CHIP "filter" ASSIGNED TO AN EPF10K10QC208-3

TCK : 1

CONF\_DONE : 2

nCEO : 3

TDO : 4

VCCIO : 5

VCCINT : 6

N.C. : 7

N.C. : 8

N.C. : 9

x7 : 10



# Floorplan Editor

- ◆ **Graphical user interface for viewing/creating resource assignments**

- Pins
- Logic cells
- Cliques
- Logic options

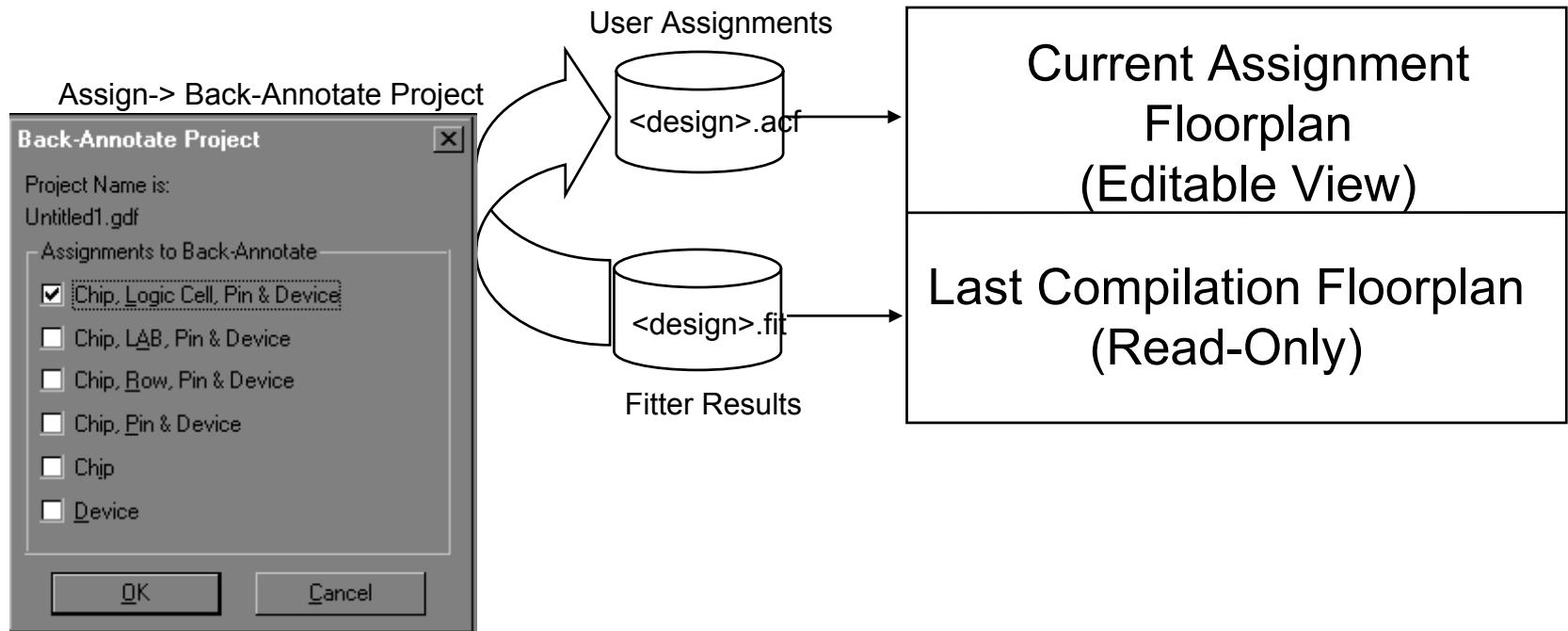
- ◆ **Drag-and-drop capability for assigning pins/logic cells**

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

- ◆ **LAB view or external chip view**



# Floorplan Views





# Floorplan Editor (Read Only)

## ◆ Last Compilation Floorplan Full Screen LAB View with Report File Equation Viewer

The screenshot displays the MAX+plus II Floorplan Editor in Full Screen LAB View. The interface includes a menu bar (File, Edit, View, Layout, Assign, Utilities, Options, Window, Help) and a toolbar. A 'Display control' menu is open, showing options: Full Screen (checked), Report File Equation Viewer (checked), Device View, LAB View (checked), Current Assignments Floorplan, and Last Compilation Floorplan (checked). The main display area shows a grid of LAB cells. A 'Highlighted LCELL' is selected, with its 'Fan-in and Fan-out' connections visible. The 'LCELL equation' is displayed in the bottom panel, showing the logic equation for the selected cell. The equation is: 
$$LC3\_C3 = DFFE(\_EQ022, GLOBAL(\_clk), VCC, VCC, VCC);$$
$$\_EQ022 = \_LC3\_C2 \& \_LC3\_C3 \& \_LC3\_C14 \& \_LC5\_C3 \# \_LC3\_C2 \& \_LC3\_C3 \& \_LC5\_C3 \# \_LC3\_C2 \&$$

Display control

Highlighted LCELL

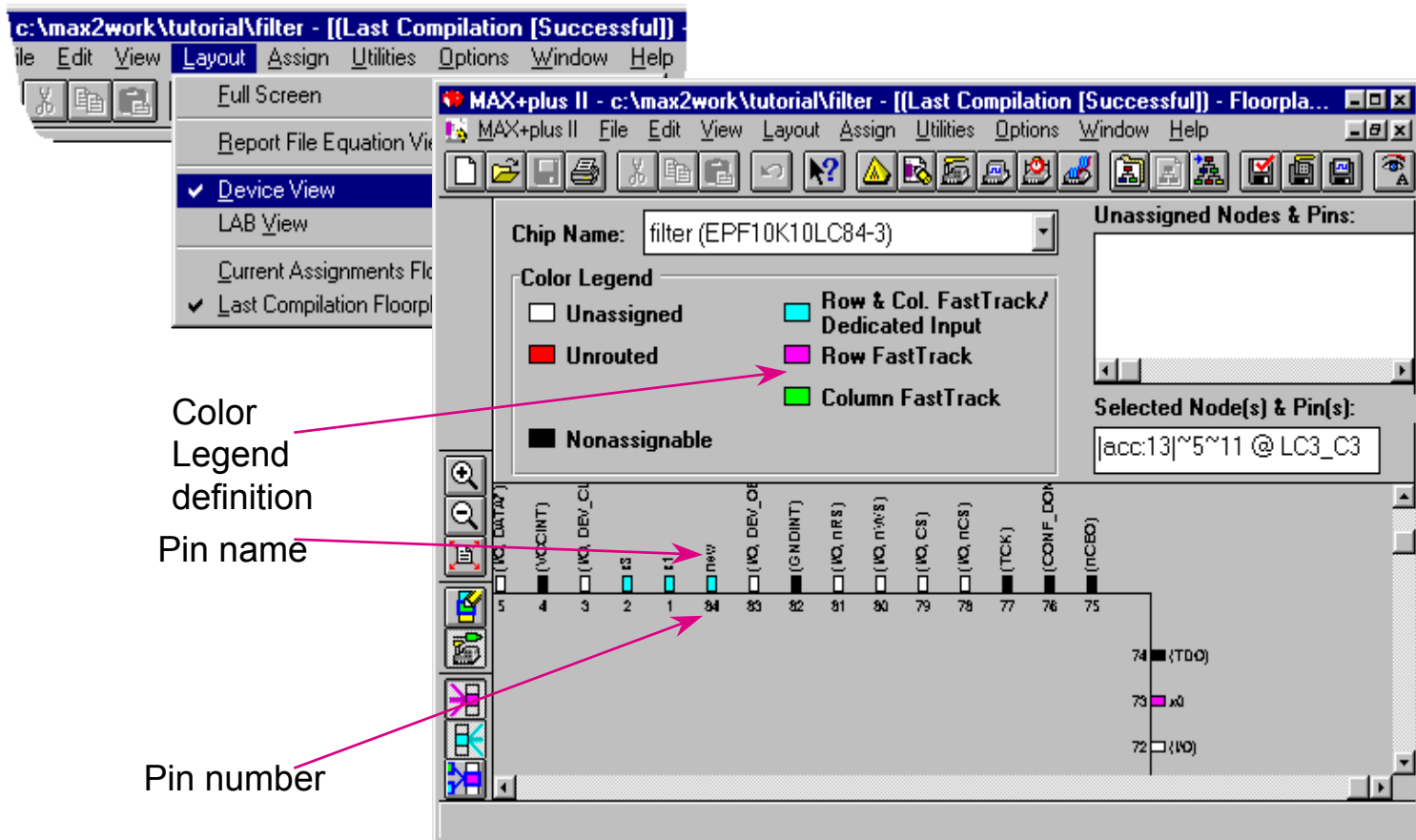
Fan-in and Fan-out

LCELL equation



# Floorplan Editor (Read Only)

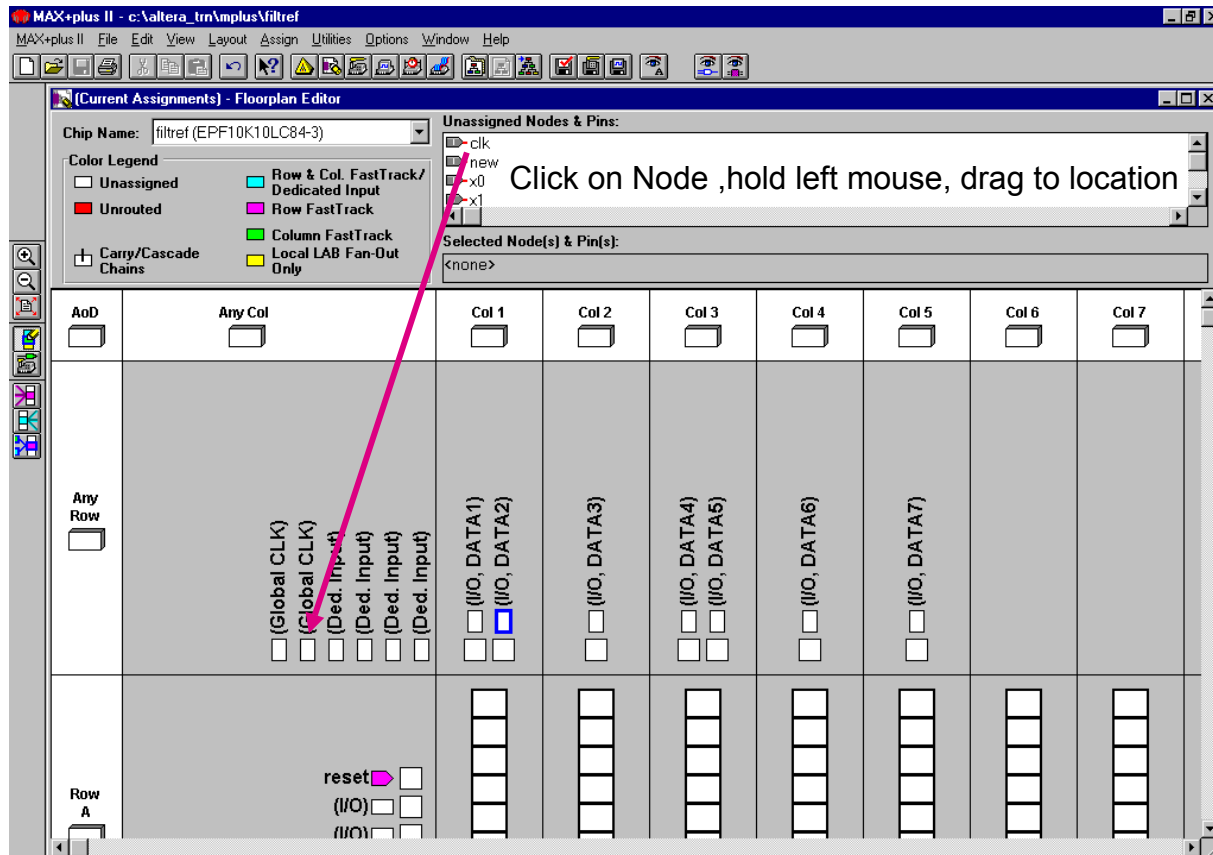
## ◆ Last Compilation Floorplan Device View





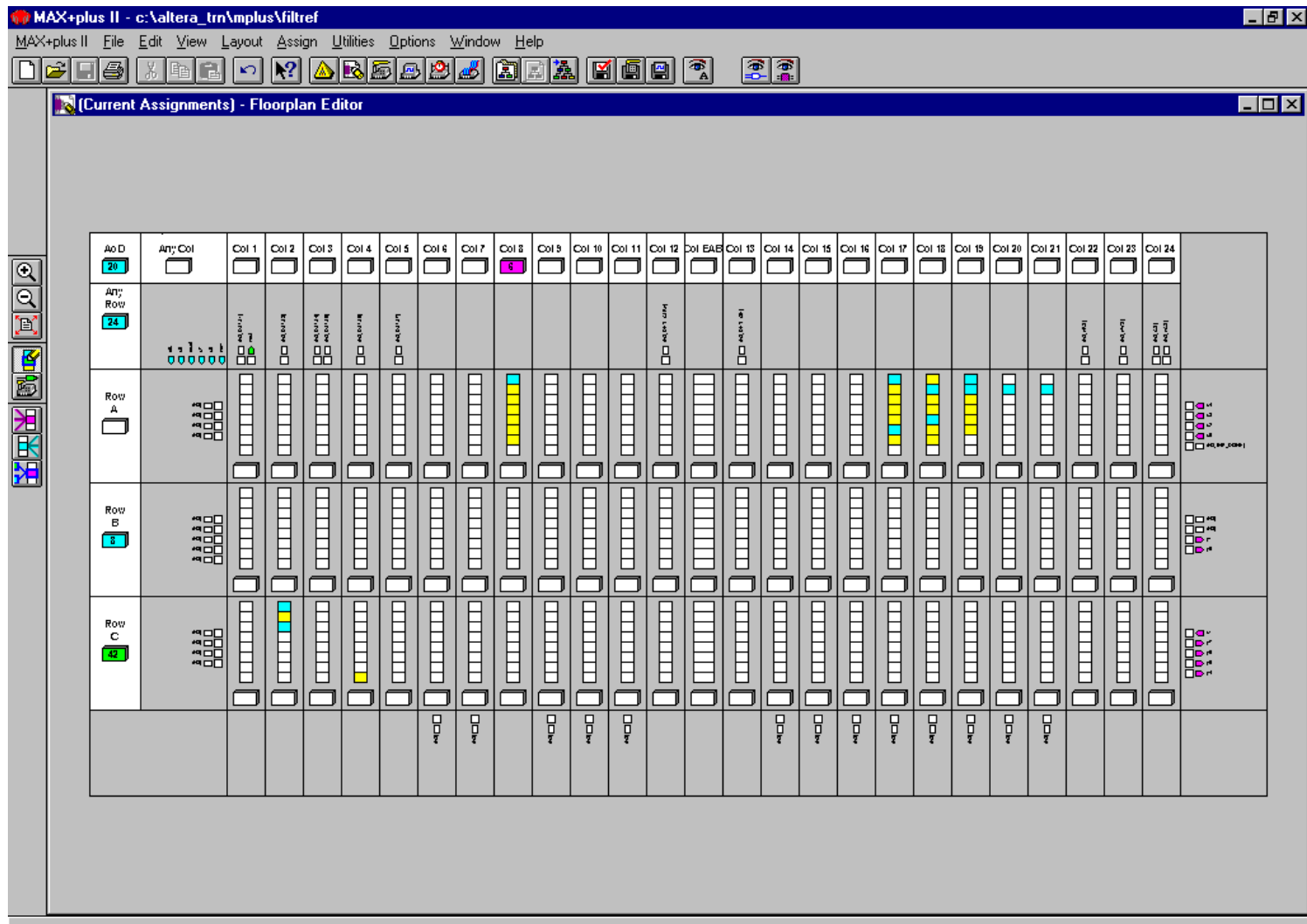
# Floorplan Editor (Editable)

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





# Floorplan Editor (Editable)





# Project Compilation Recommendations

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



# Report File Equation Viewer

The screenshot displays the Report File Equation Viewer interface. The main grid shows equations for various columns (Col 1 to Col 13) and rows (Any Row, Row A, Row B). The equations are represented by colored blocks (yellow, cyan, magenta) and black lines. A red arrow points to the 'Go To' button in the navigation pane, which is also highlighted by a red circle. The navigation pane shows the following structure:

- Fan-In (3)**
  - IN: aclr
  - IN: clk
  - REG: \_LC2\_A18 (fir\_08tp:1|nb8tp:1|nb8tp:2)
- Equations (1)**
  - < Go To
  - \_LC1\_B6 = DFF(\_LC2\_A18, GLOBAL(clk), GLOBAL(laclr), VCC);
  - Go To >
- Fan-Out (1)**
  - REG: \_LC8\_B6 (fir\_08tp:1|nb8tp:1|nb8tp:2)

A red arrow points to the 'Go To' button in the navigation pane, which is also highlighted by a red circle. The navigation pane shows the following structure:



# Routing Statistics

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

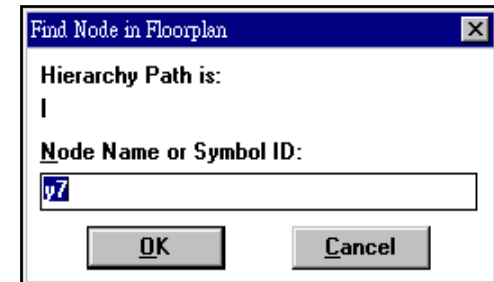
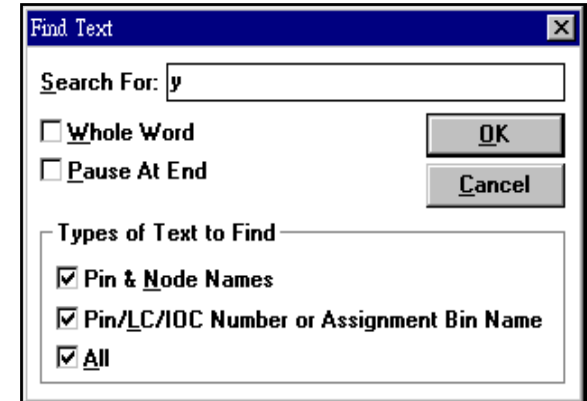
## ◆ To find text, node, ...

- “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 Node in Design File...	Ctrl+B
Find Node in Floorplan...	
Find <u>N</u> ext	Ctrl+N
Find <u>P</u> revious	Ctrl+Shift+N
Find <u>L</u> ast Edit	
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

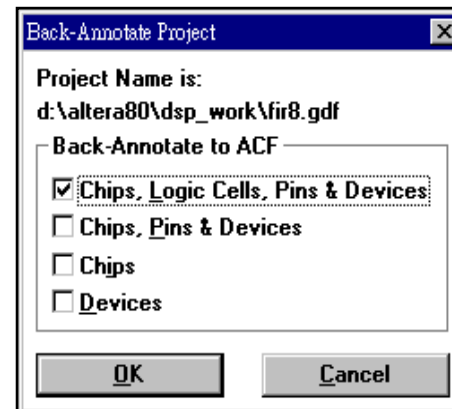
Menu: *Layout -> Current Assignments Floorplan*

Menu: *Layout -> Last Compilation Floorplan*

## ◆ Back-annotate the floorplan for subsequent compilation

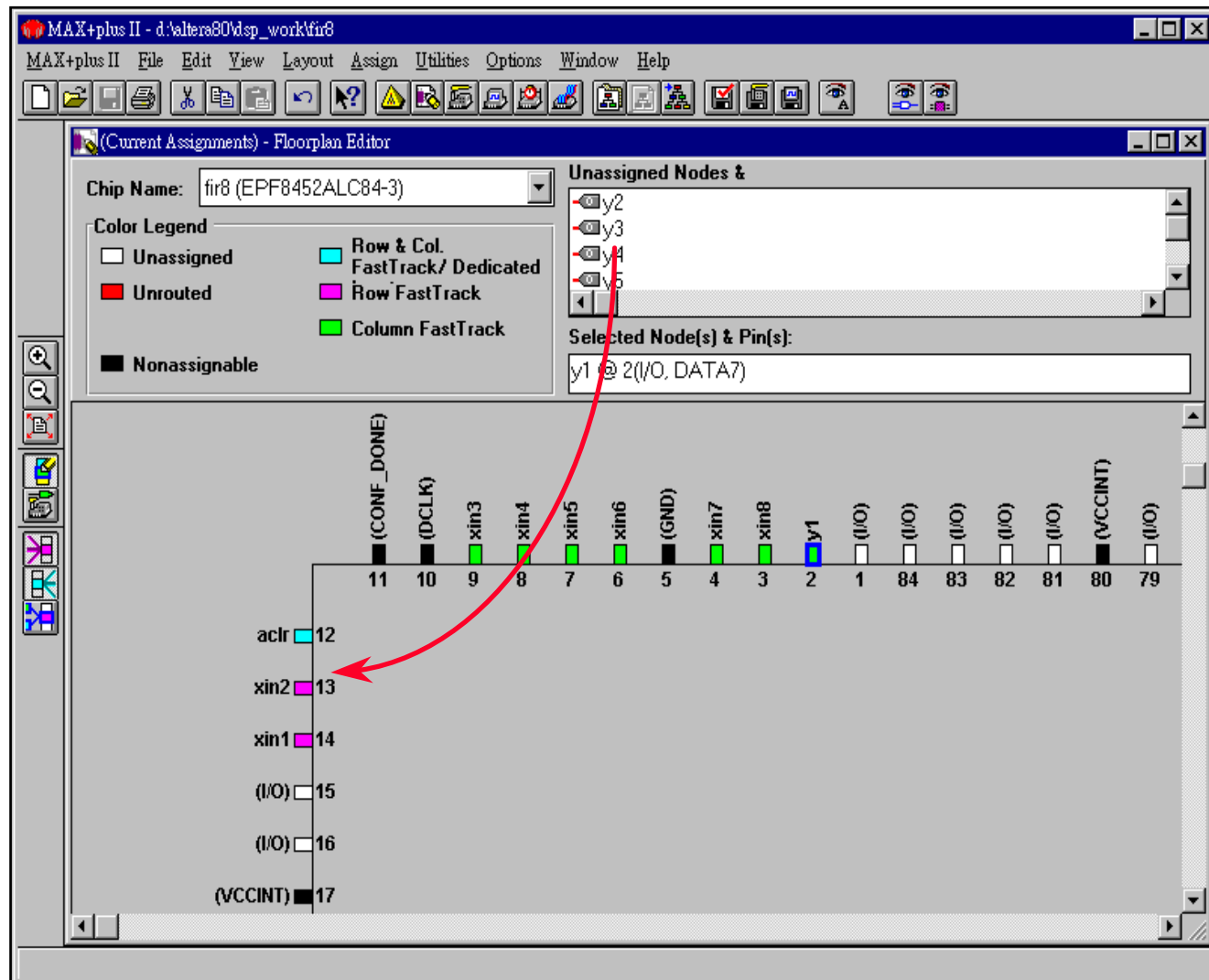
- If necessary, you can back-annotate the floorplan to ACF(Assignment & Configuration File) and it is useful for retaining the current resource and device assignments for future compilations

Menu: *Assign -> Back-Annotate Project...*





# Current Pin Assignment Floorplan



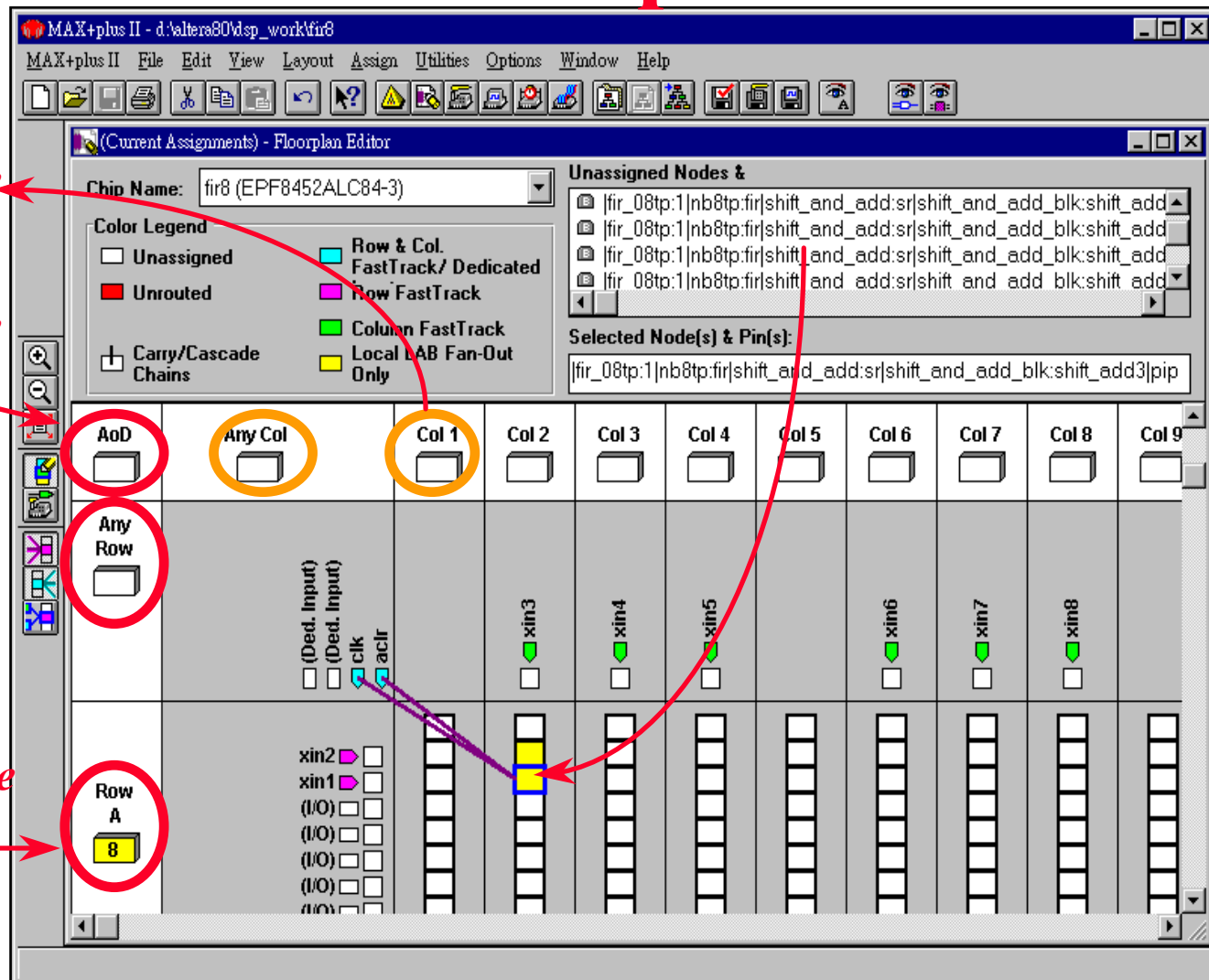


# Current LAB Assignment Floorplan

*Anywhere on this Column*

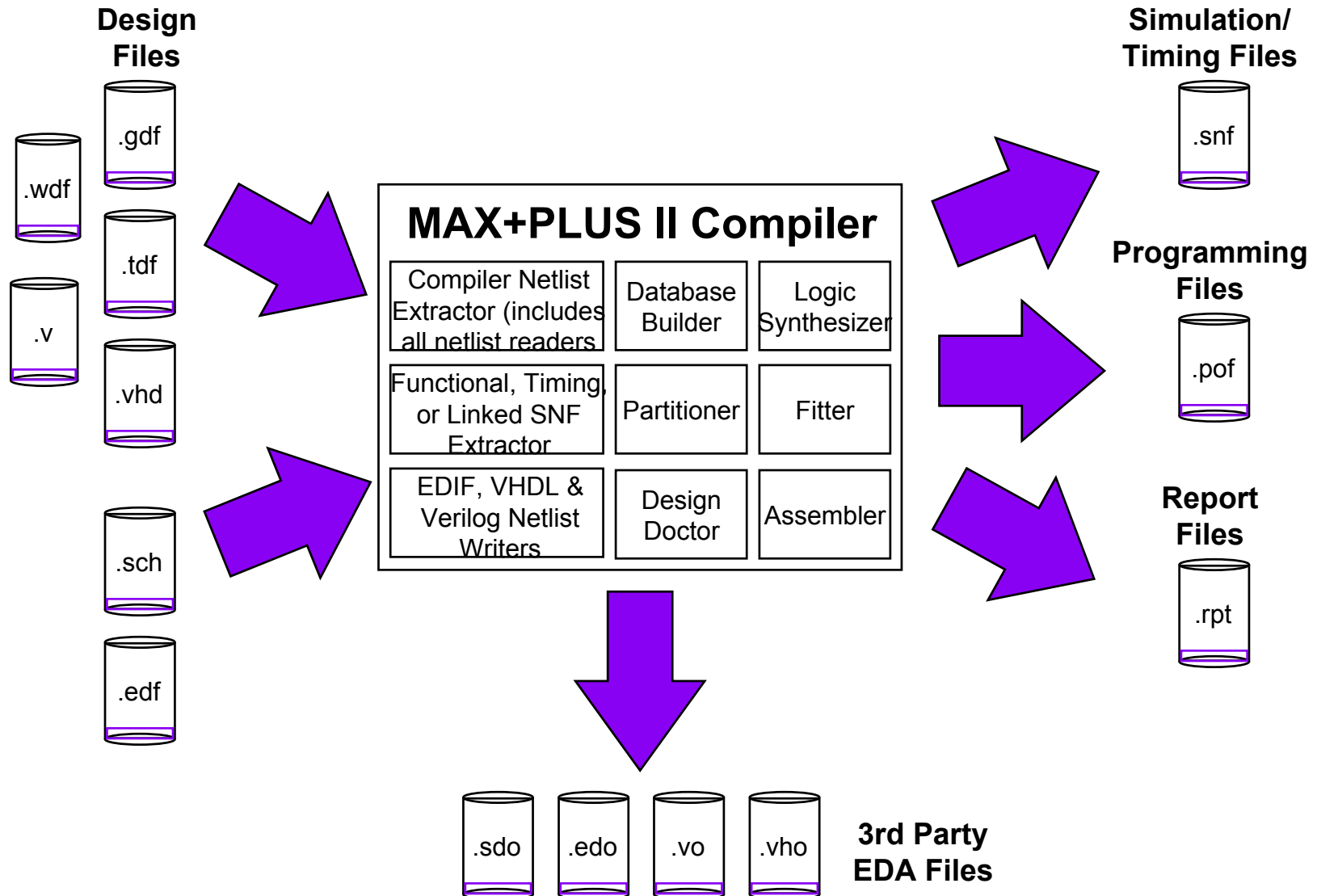
*Anywhere on Device*

*Anywhere on this Row*





# Project Compilation Summary



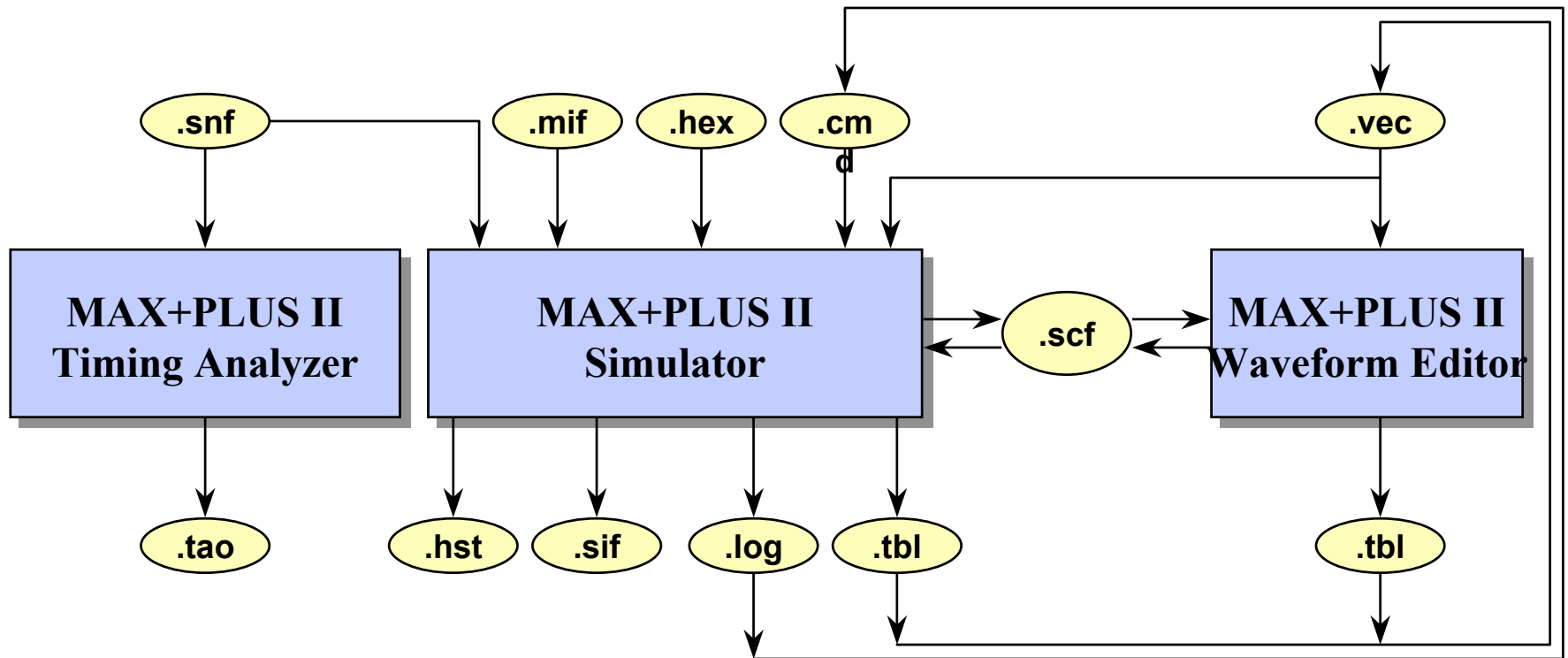


# Project Verification

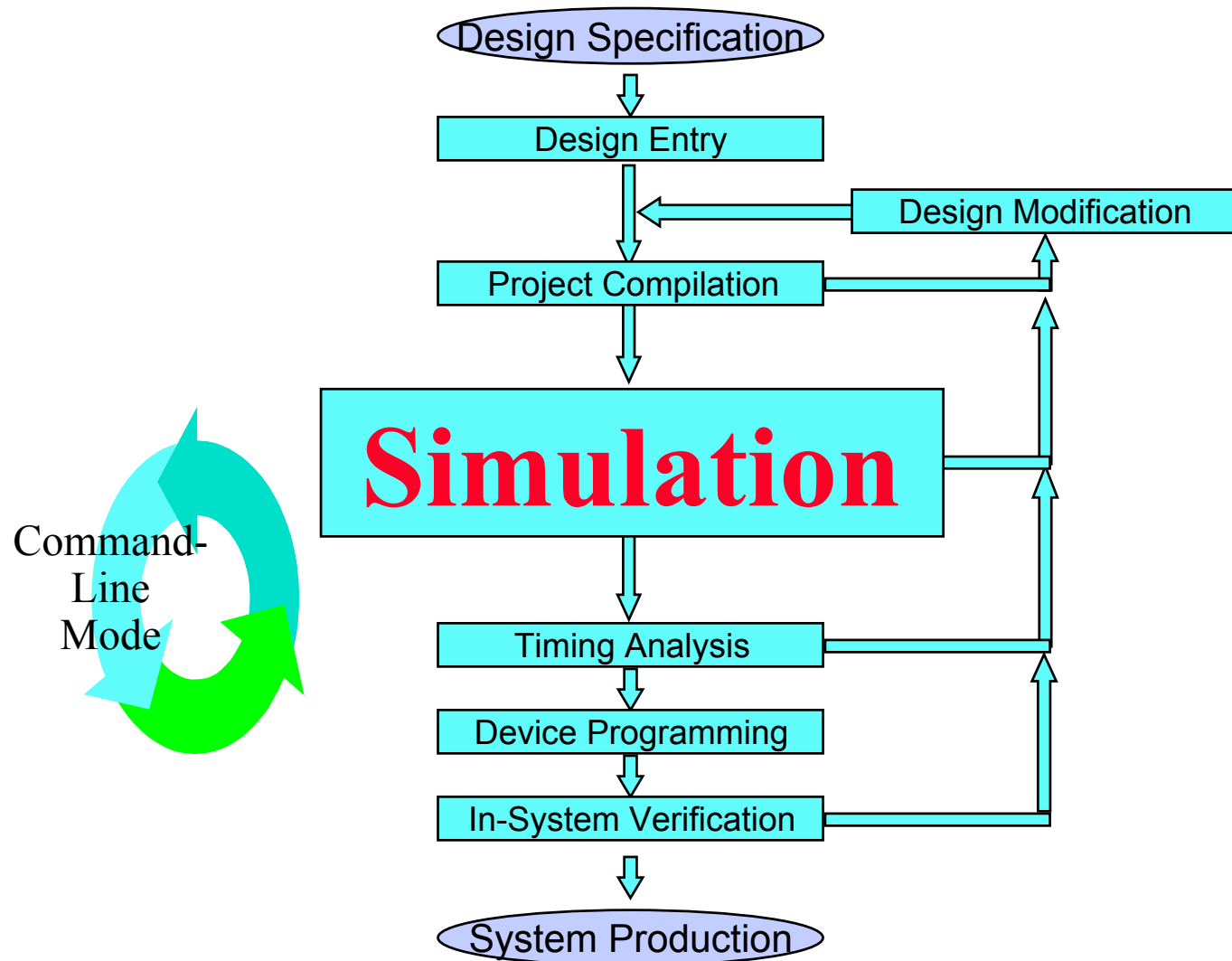
- ◆ Project Verification Methodology
- ◆ MAX+PLUS II Simulator
- ◆ Functional Simulation
- ◆ Timing Simulation
- ◆ Timing Analysis



# Project Verification Methodology

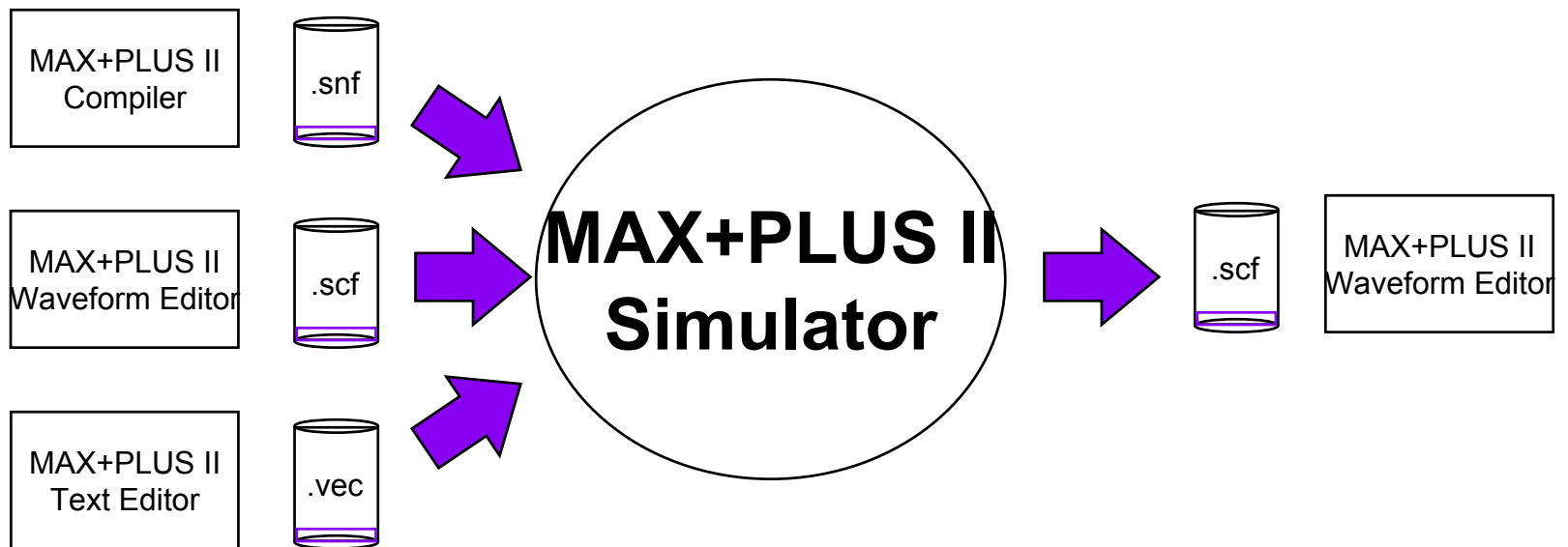








# MAX+PLUS II Simulator





# MAX+PLUS II Simulation

## ◆ Create Simulation Stimulus

- Waveform
- Vector

## ◆ Run Functional Simulation

- Fast compilation
- Logical model only, no logic synthesis
- All nodes are retained and can be simulated
- Outputs are updated without delay

## ◆ Run Timing Simulation

- Slower compilation
- Timing model: logical & delay model
- Nodes may be synthesized away
- Outputs are updated after delay



# Simulation Waveform

## ◆ Stimulus Waveform

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

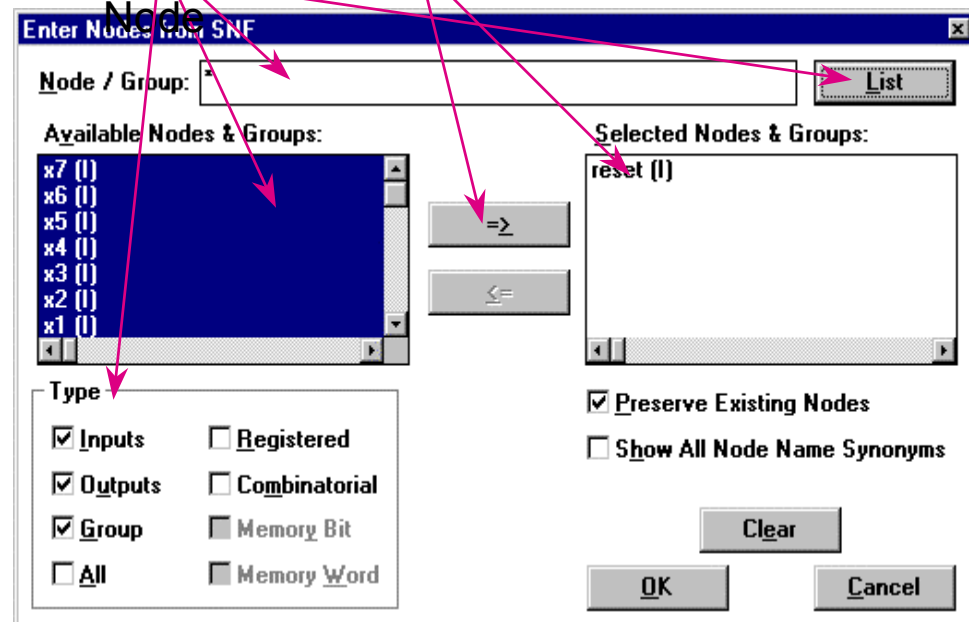
## ◆ Reference Compare waveform

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



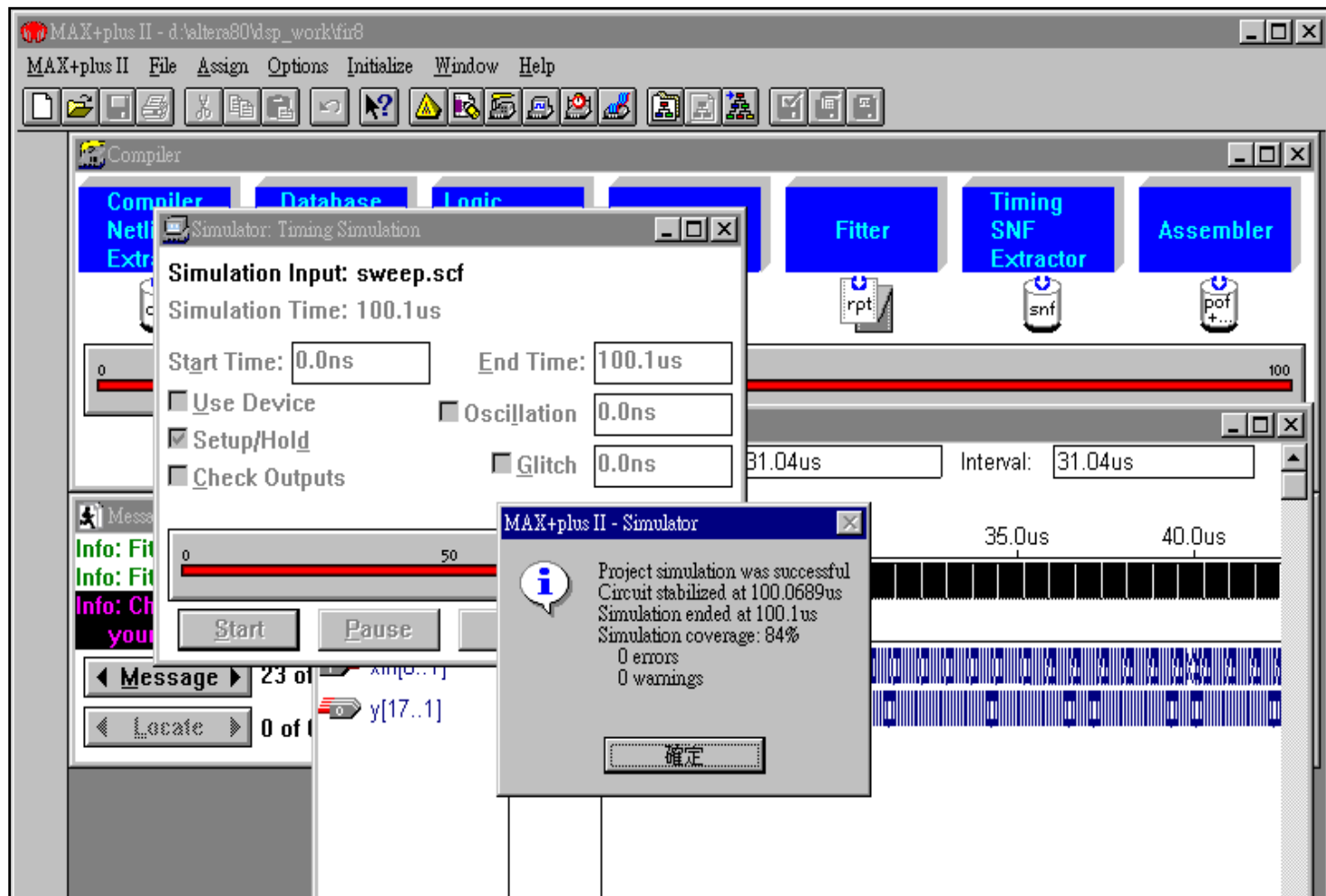
# Create Waveform Simulation Stimulus

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





# Simulator Environment



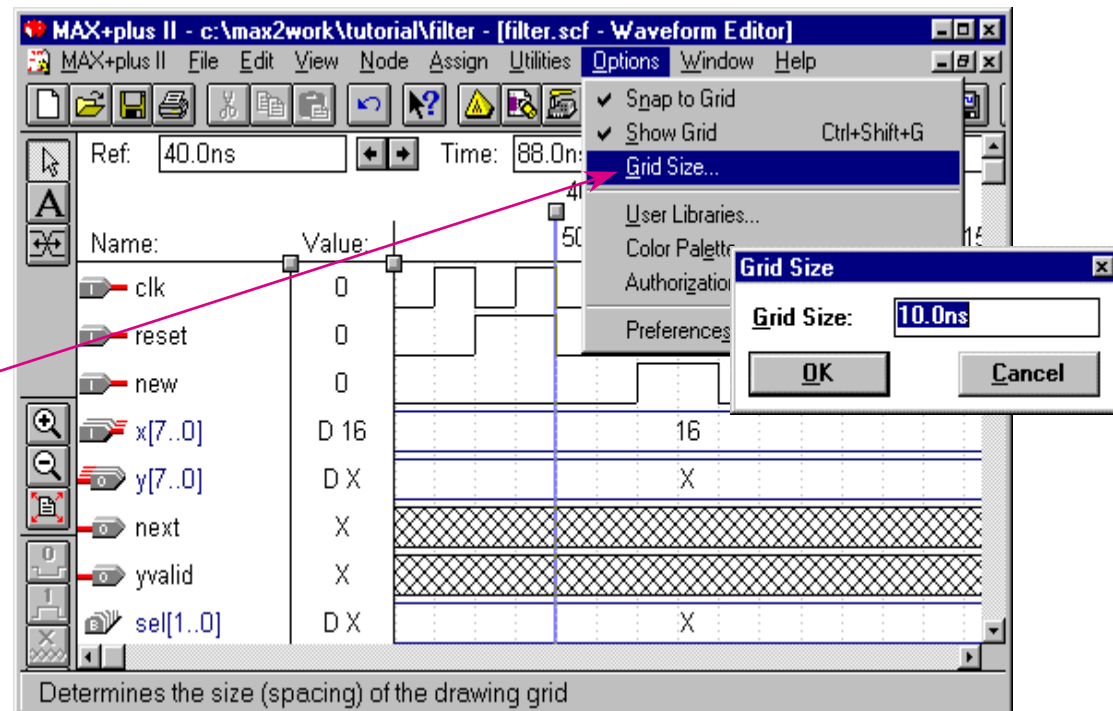


# Grid Control

## ◆ Snap to Grid

- On: waveforms drawn increments of grid size
- Off: waveforms can be drawn to any size

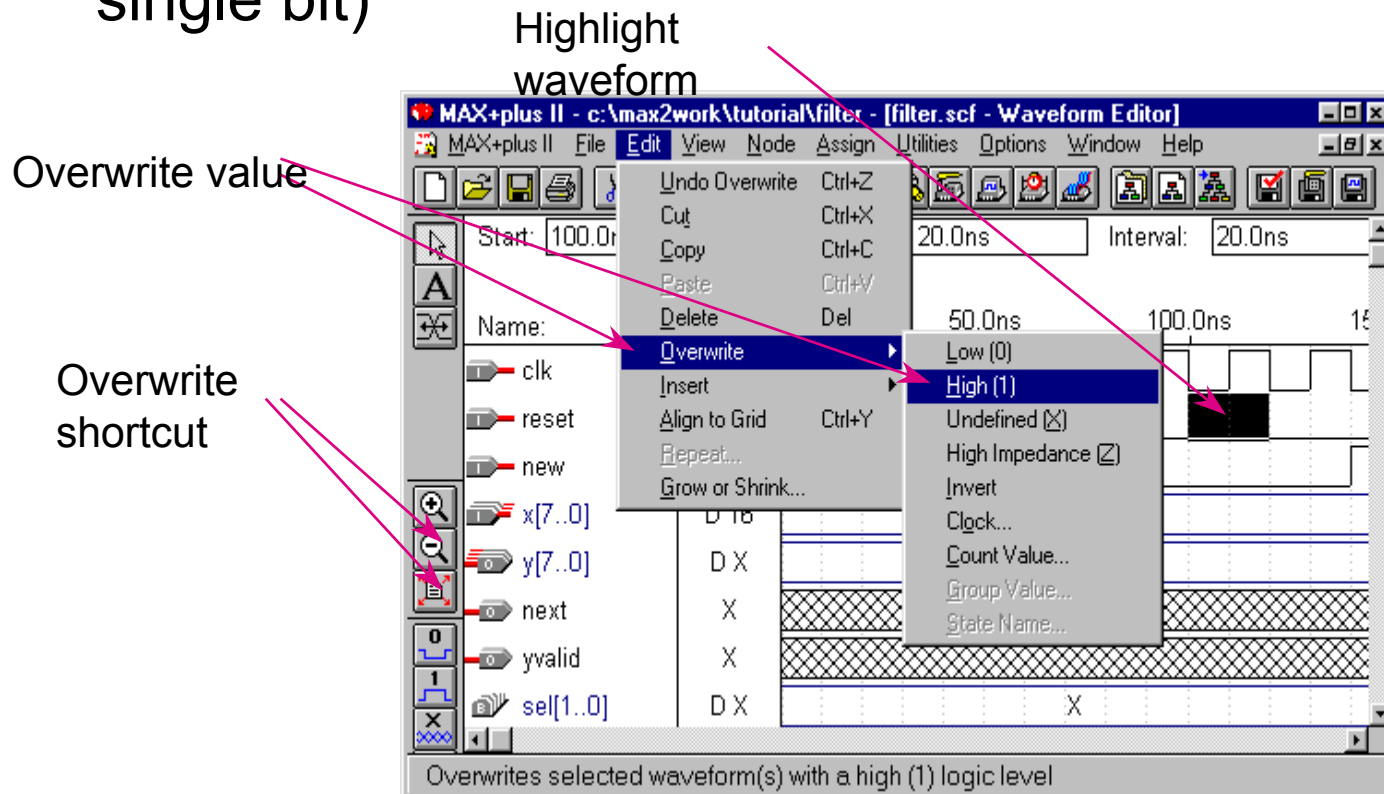
Set Grid  
size





# Draw Stimulus Waveform

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





# Create Clock Waveform

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

The screenshot shows the MAX+plus II Waveform Editor interface. The main window displays a list of signals on the left, including 'clk', 'reset', 'new', 'x[7..0]', 'y[7..0]', 'next', 'yvalid', 'sel[1..0]', 'h[2..0]', and 'tate\_m:2]filter'. The 'clk' signal is highlighted. A context menu is open over the 'clk' signal, with the 'Clock...' option selected. The 'Overwrite Clock' dialog box is open, showing the 'Interval' set to 0.0ns, 'To' set to 1.0us, 'Starting Value' set to 0, 'Clock Period' set to 20.0ns, and 'Multiplied By' set to 1. The 'OK' button is highlighted.

Highlight waveform

Clock shortcut

Specify clock period

Overwrites a node with a Clock waveform



# Create Counting Pattern

- Make sure your counting frequency matches your clock frequency

The screenshot shows the MAX+plus II Waveform Editor interface. The menu bar includes File, Edit, View, Node, Assign, Utilities, Options, Window, and Help. The Edit menu is open, showing options like Undo Overwrite, Cut, Copy, Paste, Delete, Overwrite, Insert, Align to Grid, Repeat, and Grow or Shrink. The Overwrite submenu is also open, showing options like Low (0), High (1), Undefined (X), High Impedance (Z), Invert, Clock, and Count Value... The Count Value... option is selected, opening the 'Overwrite Count Value' dialog box. The dialog box has fields for Interval (0.0ns to 1.0ns), Radix (Decimal), Starting Value (0), Ending Value (49), Count Type (Binary selected, Gray Code unselected), Count Every (10.0ns), Increment By (1), and Multiplied By (2). The OK and Cancel buttons are at the bottom right. The waveform editor shows a list of signals on the left: clk, reset, new, x[7..0], y[7..0], next, yvalid, sel[1..0], h[2..0], and tate\_m:2|filter. The x[7..0] signal is highlighted. The waveform for x[7..0] is shown in the center, with a pattern of 0s and 1s. The pattern shortcut 'XC' is visible at the bottom left. The status bar at the bottom reads: 'Overwrites a single selected node or group waveform with a specified count sequence'.

Highlight waveform

Pattern shortcut

Specify counting pattern

Specify counting frequency

Overwrite Count Value

Interval: 0.0ns To: 1.0ns

Radix is: Decimal

Starting Value: 0

Ending Value: 49

Count Type: ☒ Binary ☐ Gray Code

Count Every: 10.0ns

Increment By: 1

Multiplied By: 2

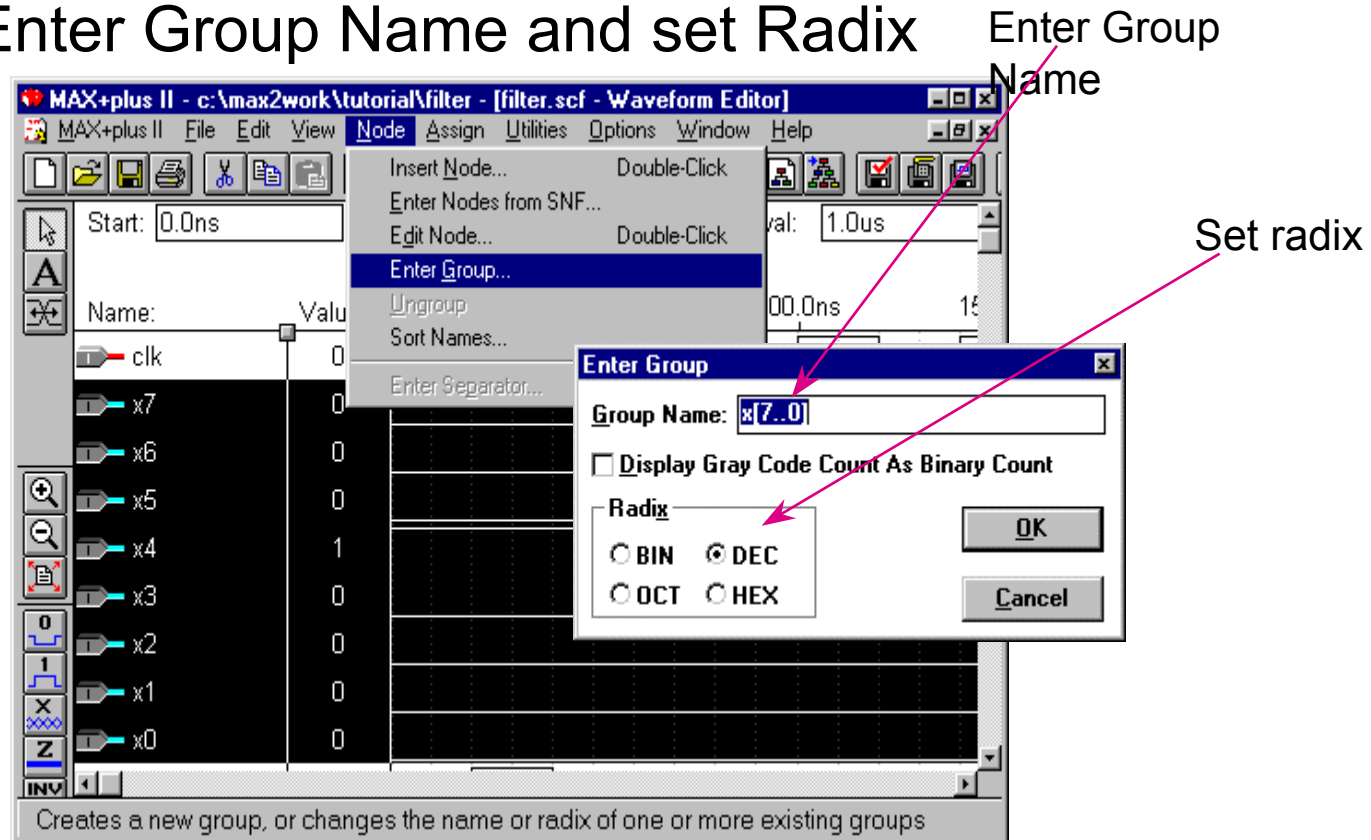
OK Cancel

Overwrites a single selected node or group waveform with a specified count sequence



# Grouping Signals and Set Radix

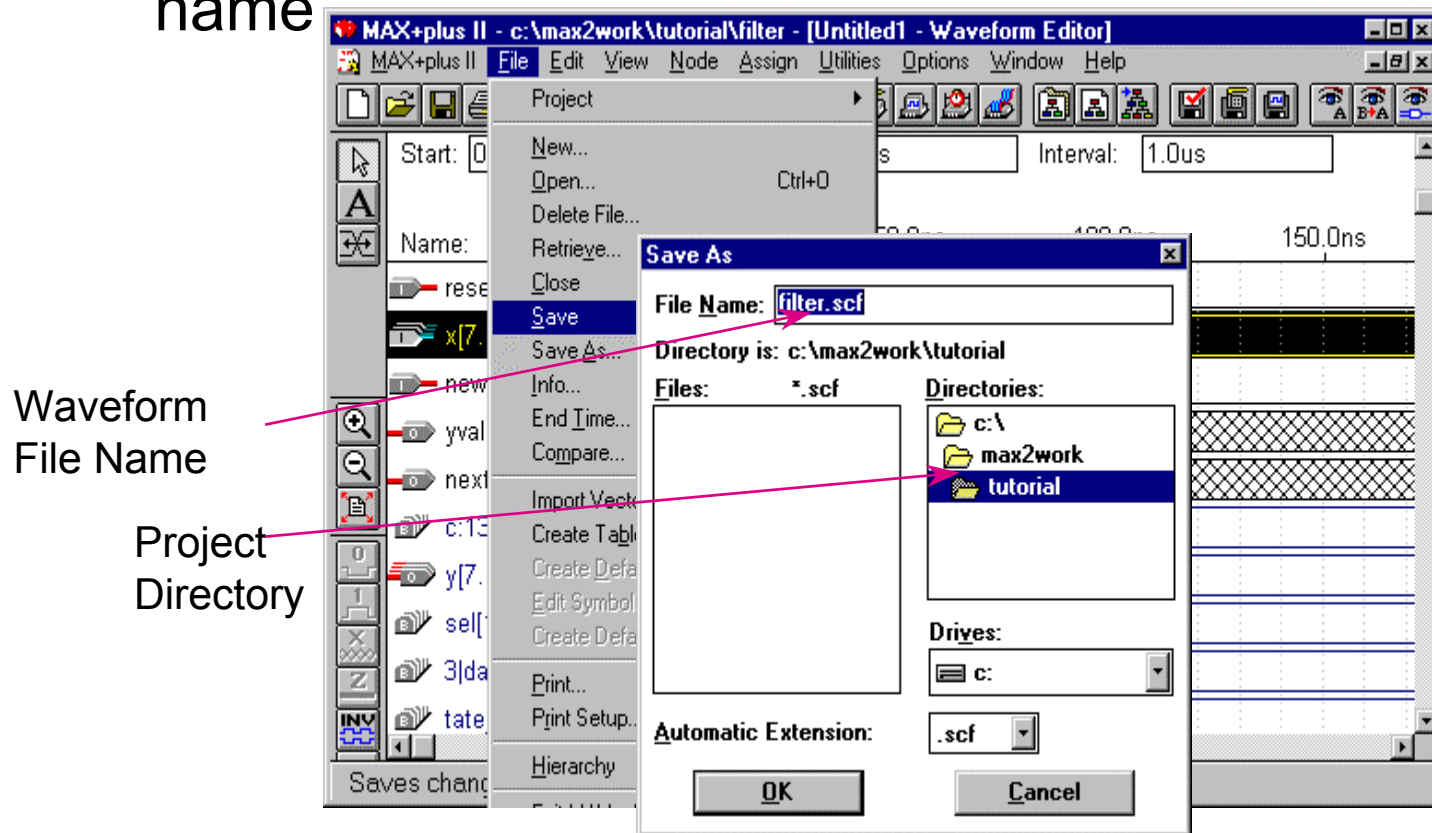
- Highlight waveforms to be grouped
  - MSB must be the top waveform
- Enter Group Name and set Radix





# Save the Waveform Stimulus File

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





# Create Vector Simulation Stimulus

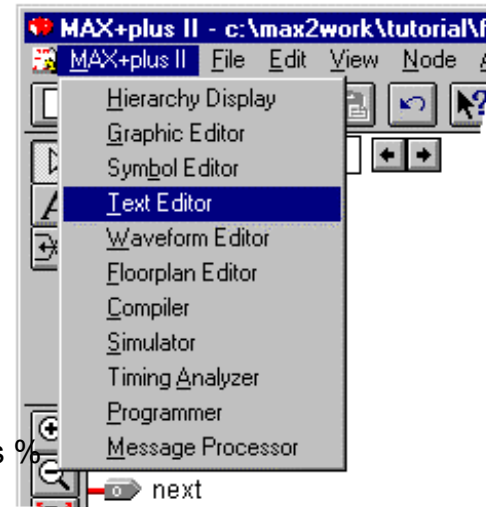
- Open **Text Editor**
- Type in vector stimulus

- Clock

```
% units default to ns %
START 0 ;
STOP 1000 ;
INTERVAL 100 ;
INPUTS CLOCK ;
PATTERN
0 1 ; % CLOCK ticks every 100 ns %
```
- Pattern

```
INPUTS A B ;
PATTERN
0> 0 0
220> 1 0
320> 1 1
570> 0 1
720> 1 1;
```
- Output

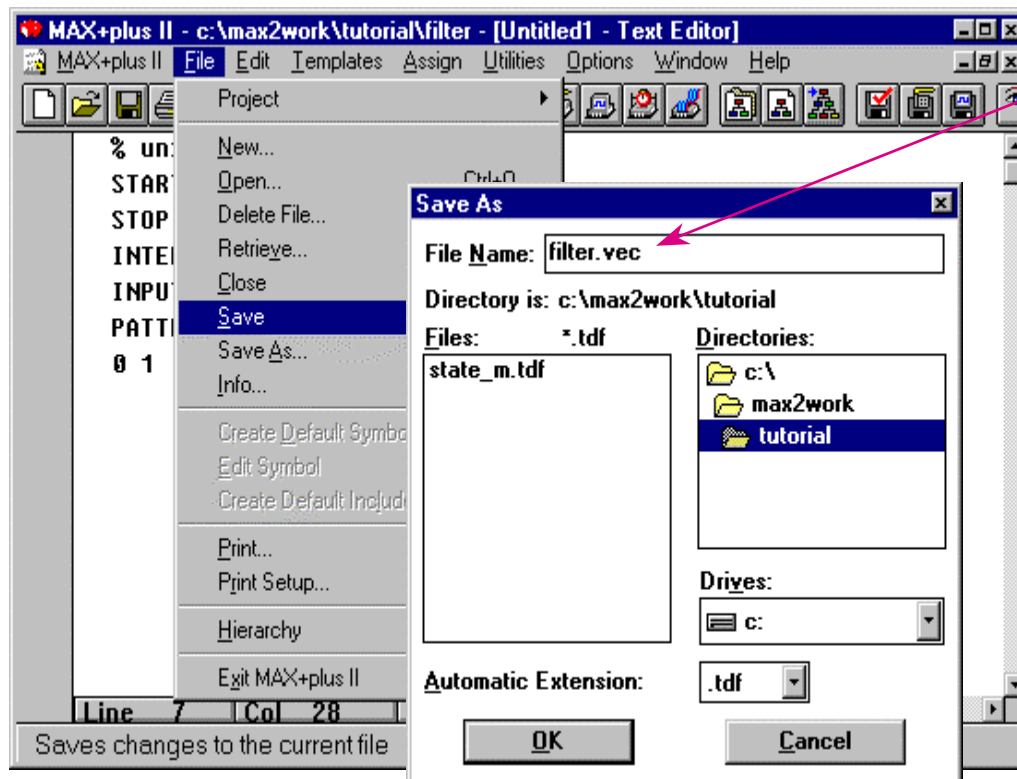
```
OUTPUTS Y1 Y0 ;
PATTERN      % check output at every Clock pulse %
= X X
= 0 0
= 0 1
= 1 0
= 1 1;
```





# Save the Vector Stimulus File

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

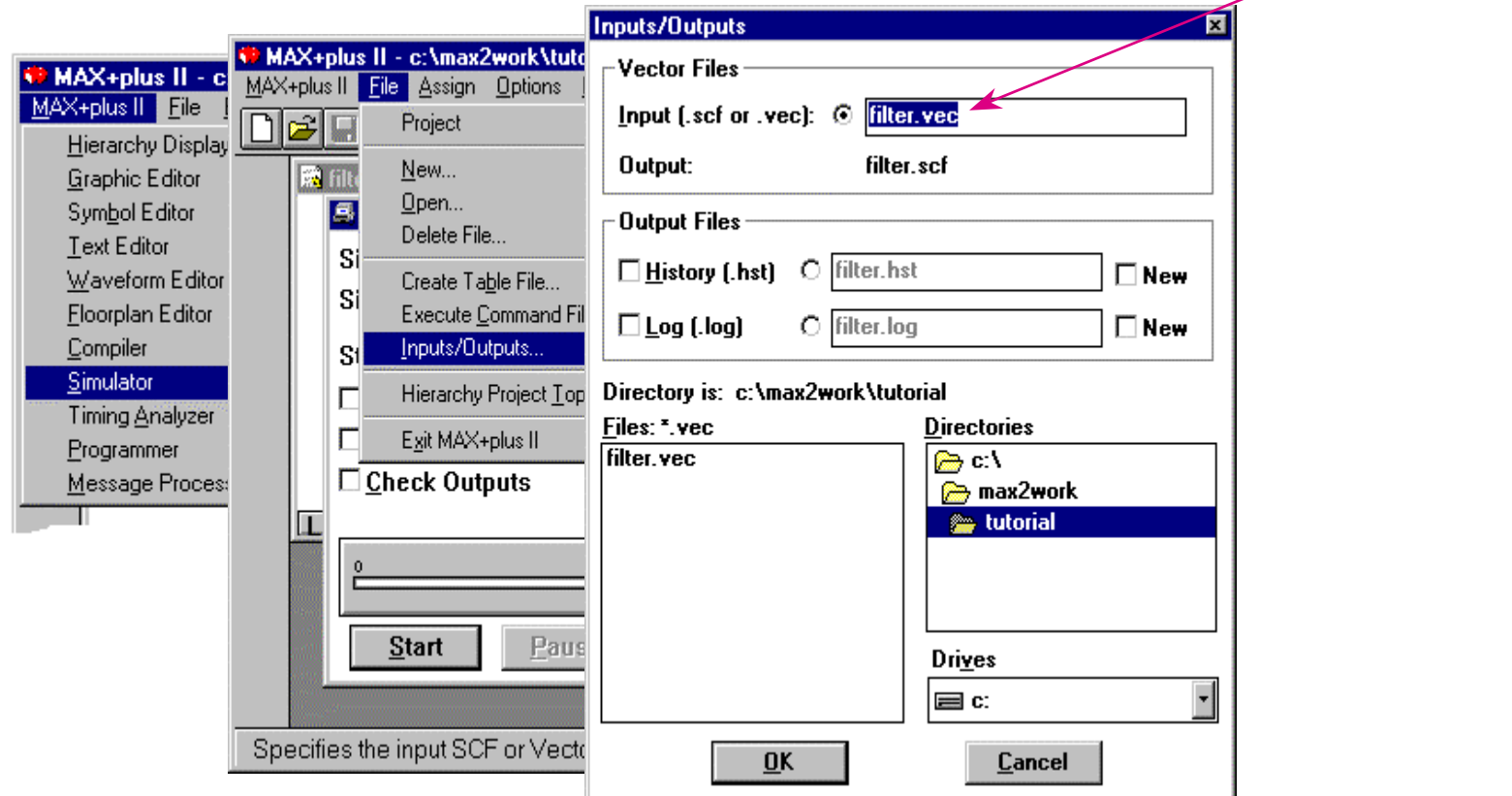


Change the extension to .vec



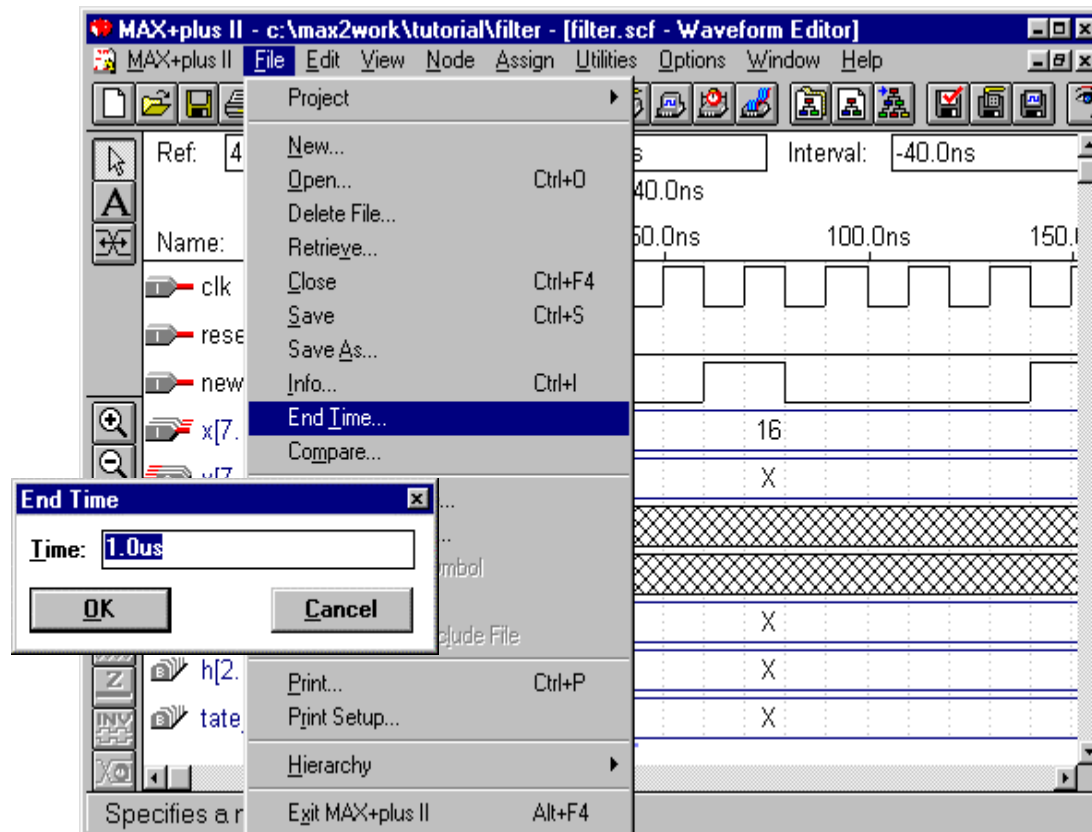
# Select Simulation Stimulus File

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





◆ Specify maximum length of simulation time with End Time





# Run Functional Simulation

- Click on Start then Open SCF to see result

Click on Start Button

Open .scf file

Output change on clock edge



# MAX+PLUS II Functional Simulation

## ◆ Use to verify operation of design

## ◆ Advantage over Timing Simulation

- Fast compilation
- All nodes are retained and can be simulated
- Outputs are updated without delay
  - Most of the time, this makes figuring out cause and effect much easier

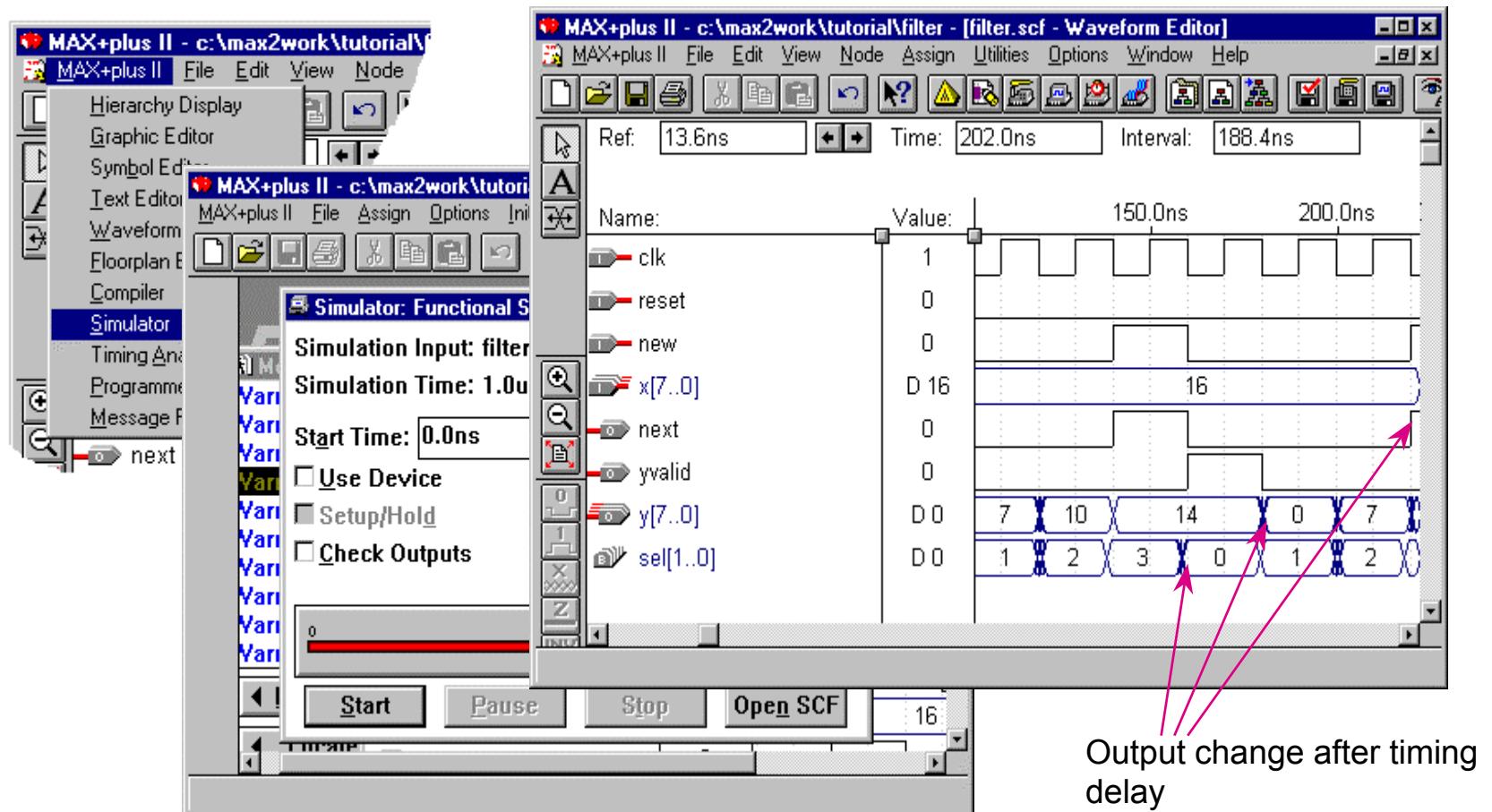
## ◆ Disadvantages

- Logical model only, no logic synthesis
- No delays in simulation
  - Oscillations, glitches and other timing related errors do not show up



# Run Timing Simulation

- Click on Start then Open SCF to see result





# MAX+PLUS II Timing Simulation

- Used to debug timing related errors
- Advantages over Functional Simulation
  - Simulation of full synthesis result
  - Outputs change after timing delay
    - Detection of oscillations, glitches and other timing related errors are possible
- Disadvantages
  - Longer compilation time
  - Combinatorial logic nodes cannot be simulated
    - Node may be transformed or removed
  - Only “Hard” nodes can be simulated
  - Timing delays make debugging more difficult because cause and effect relationships are harder to locate



# Comparing Different Simulations

## Compare Two Simulation Files

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



# Project Simulation Recommendations

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



# Simulation Input & Output Files

## ◆ Specify simulation input and output files

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

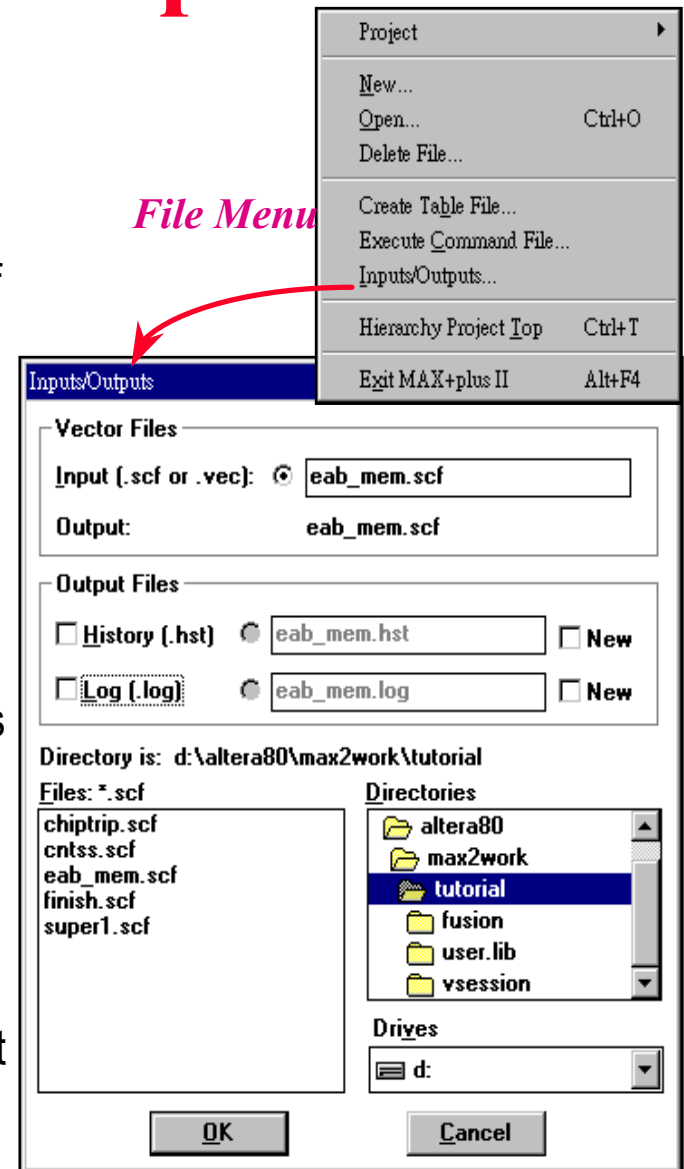
*Menu: File -> Inputs/Outputs...*

- VEC file will be converted into SCF file by Simulator
- You can specify a history(\*.hst) or log(\*.log) file to record simulation commands and outputs

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

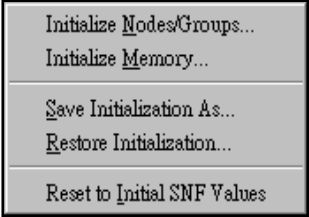
*Menu: File -> Create Table File...*

- TBL file format is a subset of VEC file format
- A TBL file can be specified as a vector input file for another simulation





# Memory Initialization



Initialize Nodes/Groups...  
Initialize Memory...  
Save Initialization As...  
Restore Initialization...  
Reset to Initial SNF Values

## *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: [LPM\_RAM\_DQ:1|altrom:sram|content]

Address: Value:

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

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

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

OK Cancel



# Memory Initialization File Formats

Initialize Memory

Memory:

Address: Value:

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

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

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;

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

*HEX file example*

*MIF 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



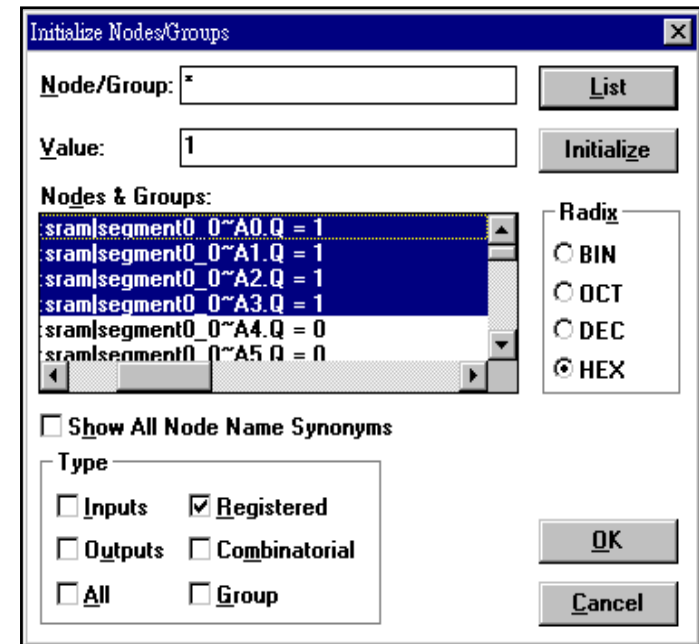
# Node/Group Initialization

## ◆ 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...**

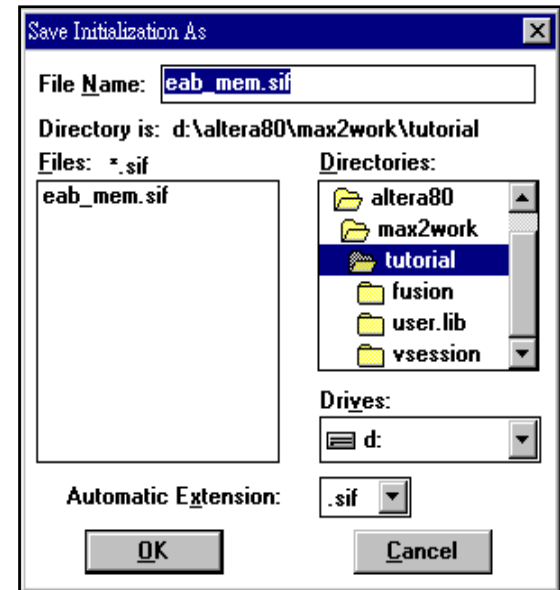
- To retrieve initialized node, group, and memory values stored in a SIF file

**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*

A screenshot of the 'Breakpoint' dialog box. The 'Breakpoint Name' field contains 'ab'. The 'Conditions' section has a 'Node/Group' dropdown set to 'ram\_we', an 'Operator' dropdown set to '>->', and a 'Value' field set to '1'. The 'Radix' section has radio buttons for BIN, DEC, OCT, and HEX, with HEX selected. There are 'Add Condition' and 'Delete Condition' buttons. On the right side, there are buttons for 'OK', 'Cancel', 'Add', 'Delete', 'Change', 'Enable', and 'Disable'. At the bottom, there is a list of 'Existing Breakpoints' with two entries: '\*ab:ram\_we >-> 1;' and '\*aa:.TIME = 5.0us;'. A checkbox labeled 'Show All Node Name Synonyms' is also present.



# Monitoring Options

## ◆ Setup time & hold time

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

## ◆ Glitch

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

## ◆ Oscillation

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



# Project Simulation Summary

## ◆ Two types of simulation

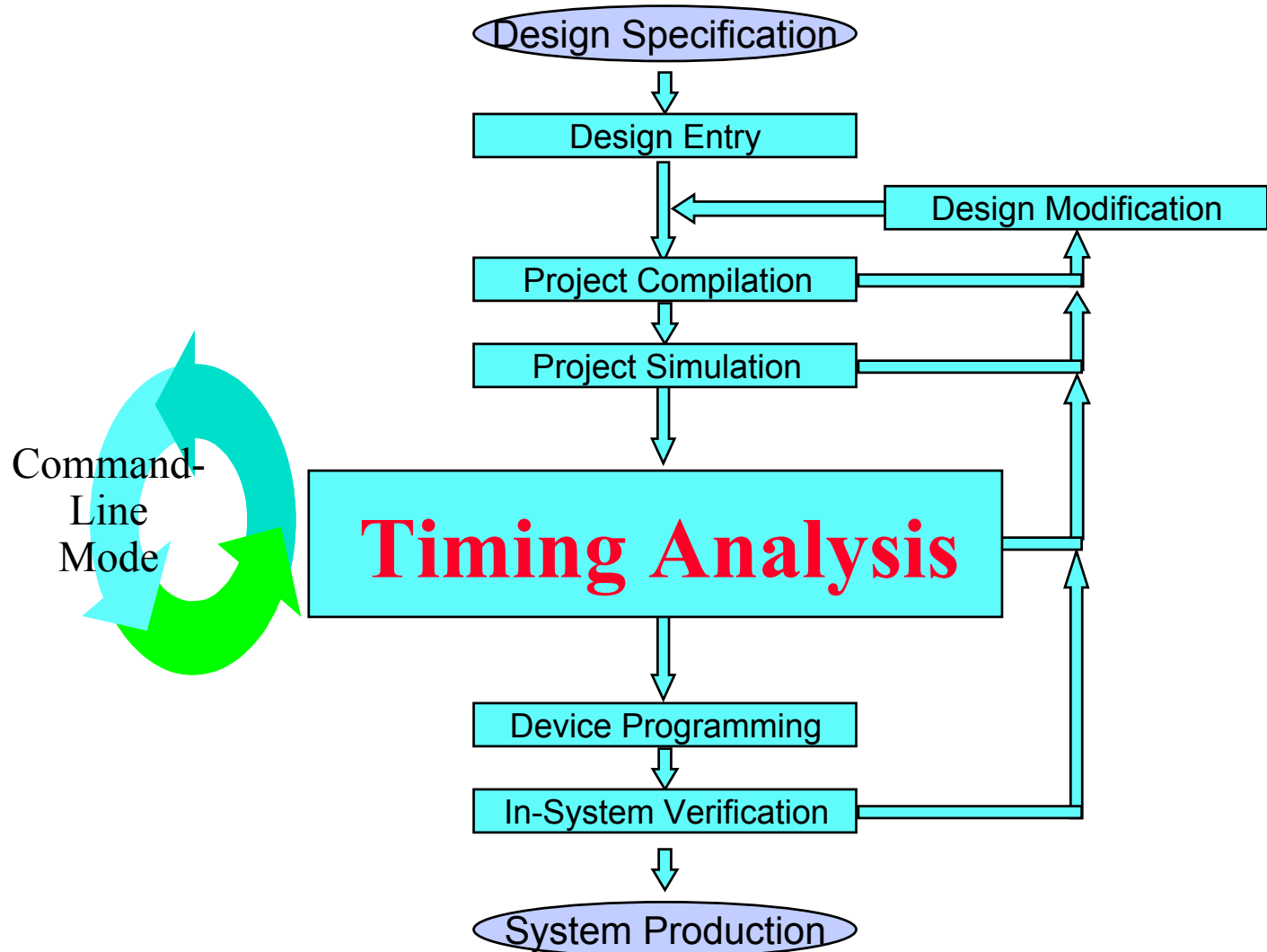
- Functional simulation
  - No logic synthesis
  - No delay model
  - All nodes can be simulated
- Timing simulation
  - Logic synthesis
  - Delay model
  - Only hard nodes can be simulated

## ◆ Two types of stimulus file

- Waveform
- Vector

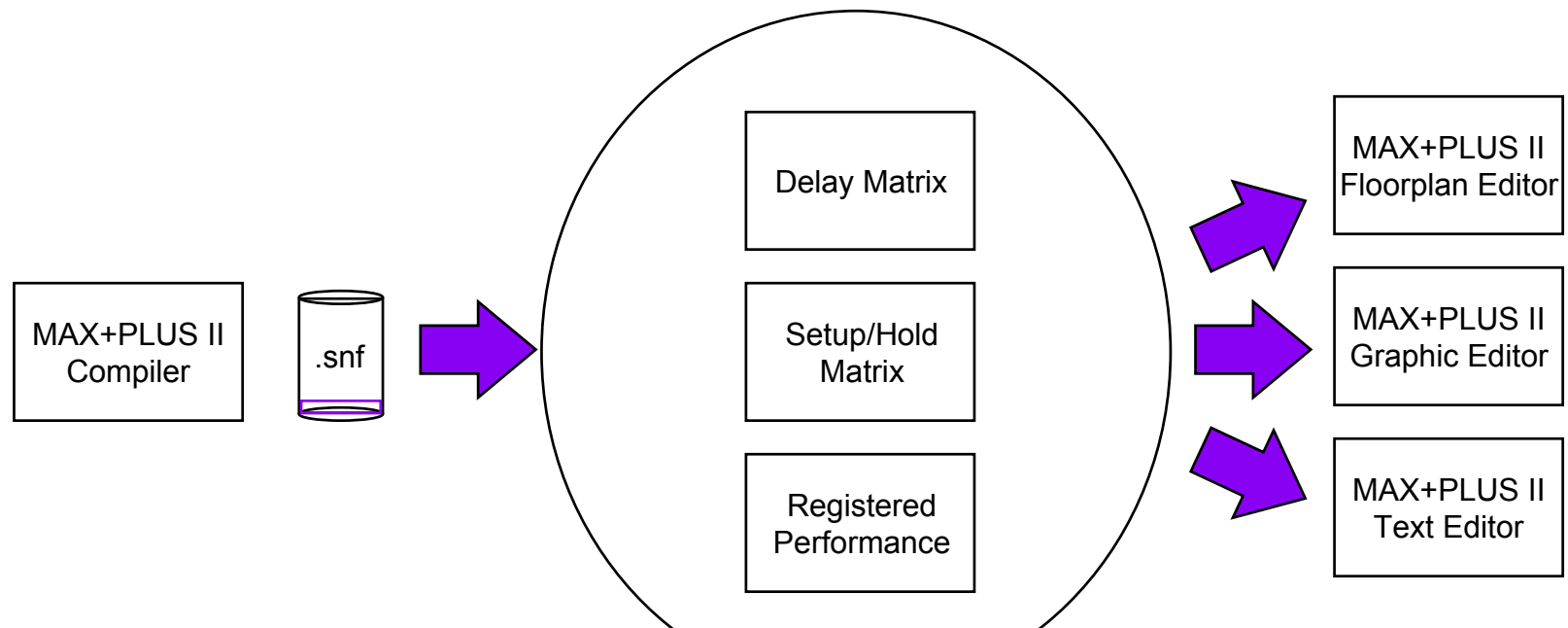
## ◆ Simulation result is stored in .scf file







# MAX+PLUS II Timing Analyzer



**MAX+PLUS II  
Timing Analyzer**



# Project Timing Analysis

## ◆ Timing Analyzer is a static timing analyzer

## ◆ Three forms of timing analysis

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

## ◆ Source of delay path can be located in

- Design file
- Floorplan Editor



# 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

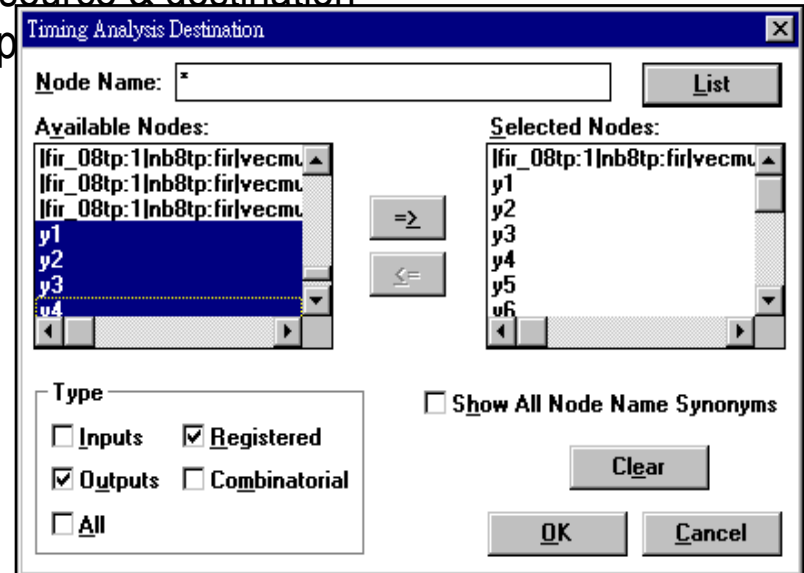
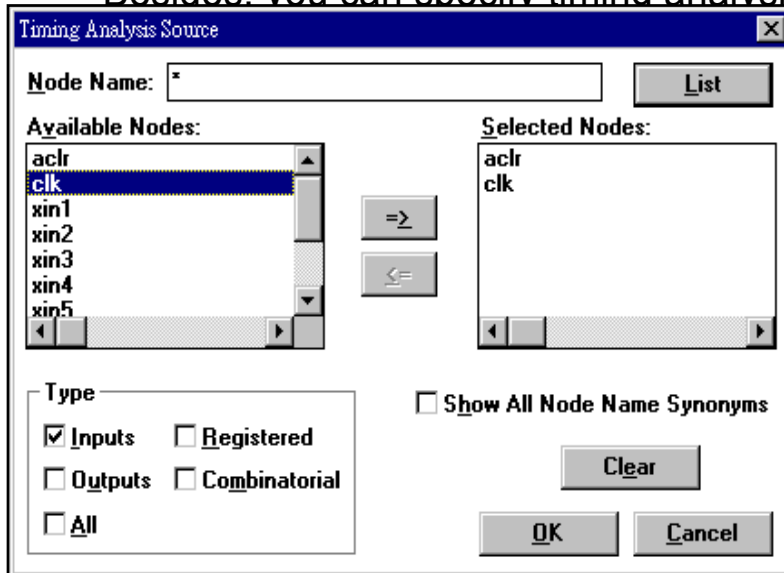
Timing Analysis Source...	Ctrl+Alt+S
Timing Analysis Destination...	Ctrl+Alt+D
Timing Analysis Cutoff...	Ctrl+Alt+C

Menu: Node -> Timing Analysis Source...

*Node Menu*

Menu: Node -> Timing Analysis Destination...

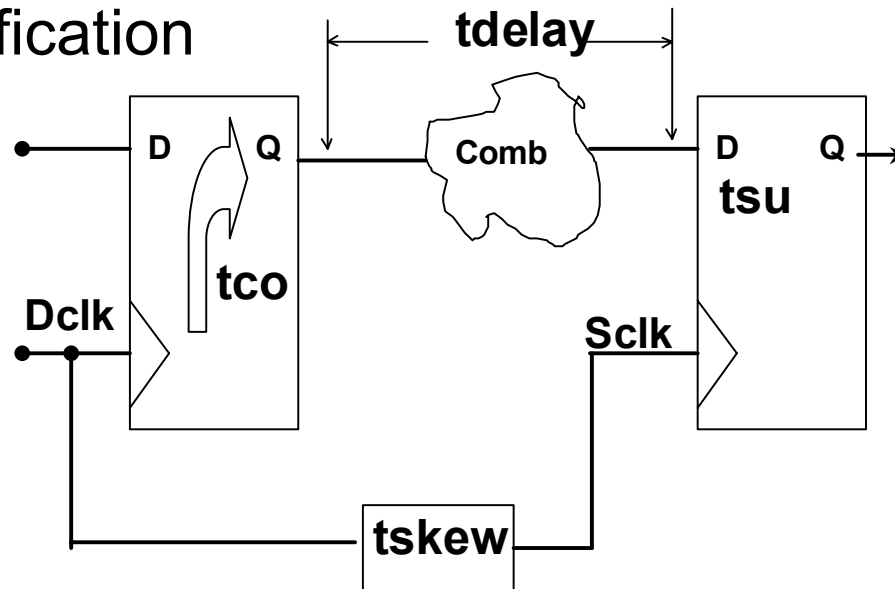
- Besides, you can specify timing analysis source & destination





# Registered Performance Analysis

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



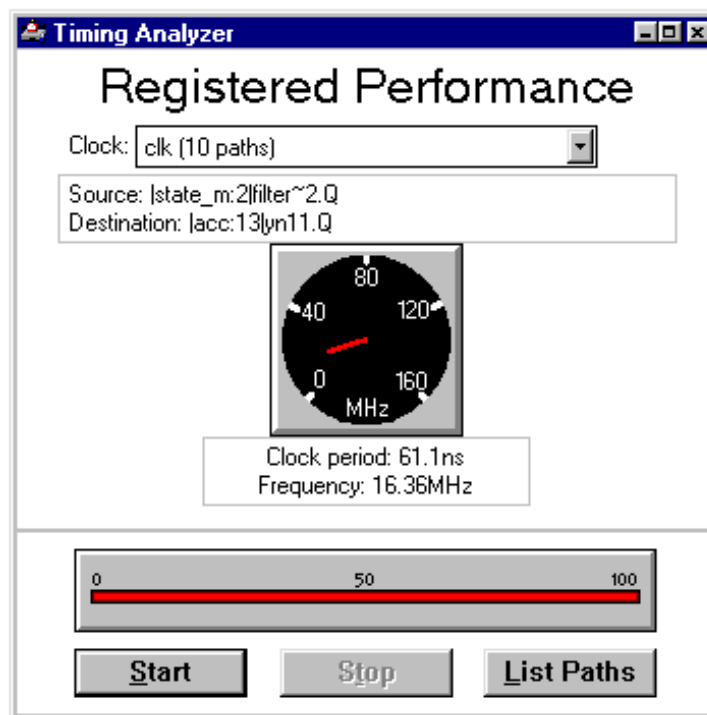
- Clock period =  $t_{co} + \text{delay} + t_{su} + t_{skew}$

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




# Run Registered Performance Analysis

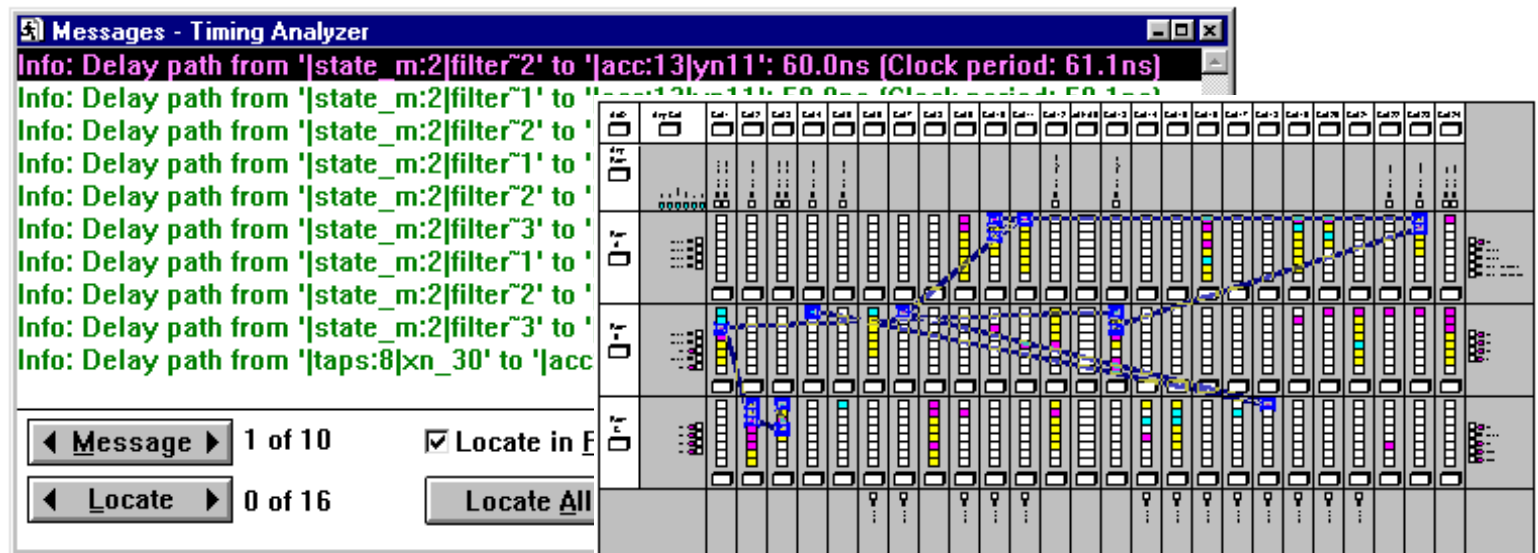
- ◆ Click on Start
- ◆ Source/Destination, Clock period and Frequency of the longest path are displayed
- ◆ Click on List Paths to trace delay path





# Tracing Delay Path In Floorplan Editor

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





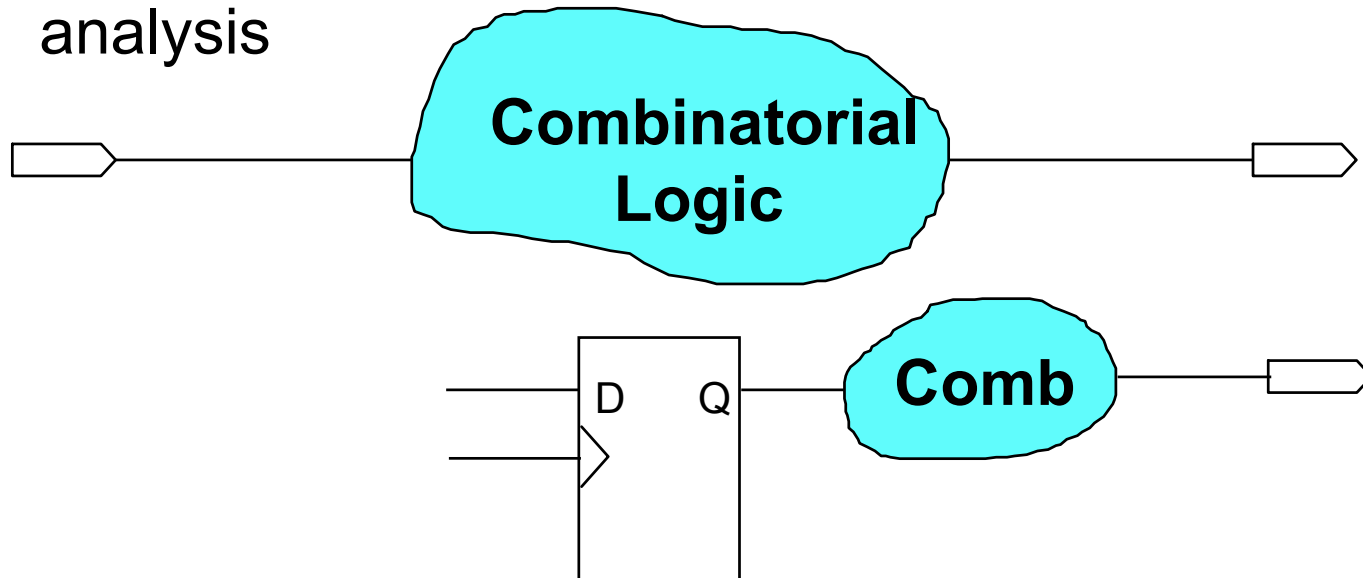
# Application of Registered Performance

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



# Delay Matrix Analysis

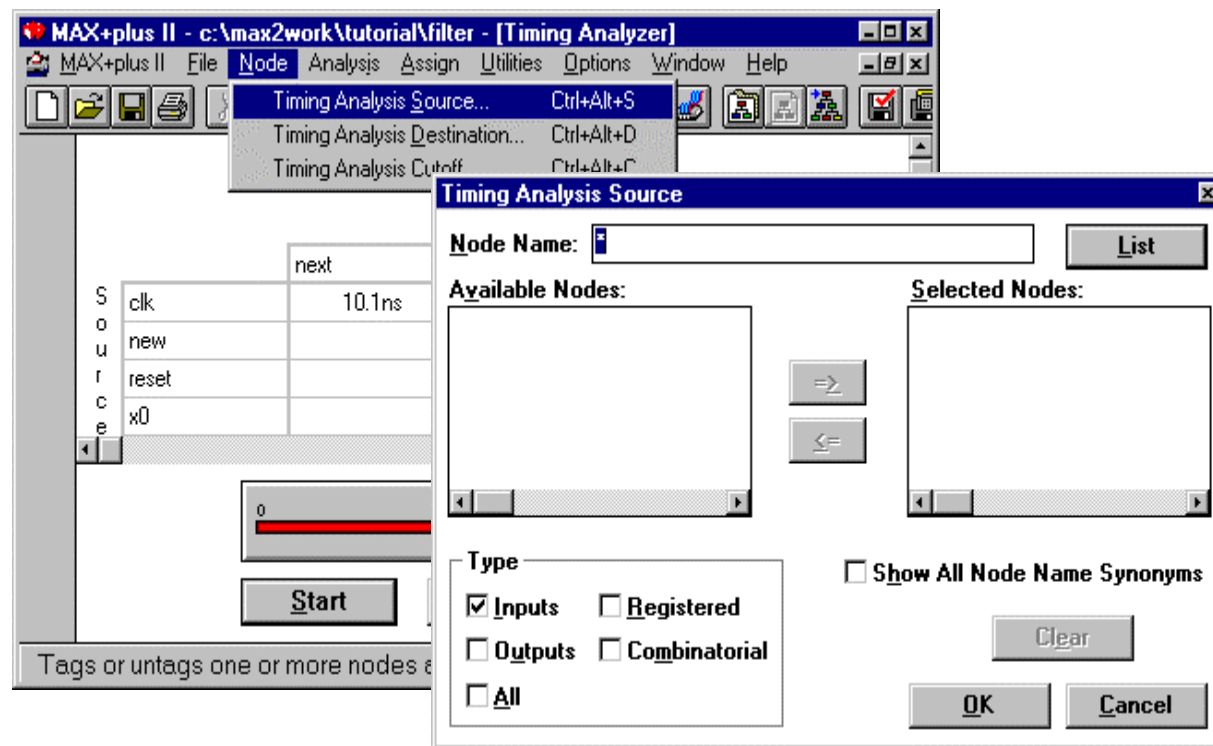
- Calculates combinatorial logic delays
- Typically used to evaluate input pin to output pin delay
- Internal point to point delay analysis is possible by setting node source and destination for analysis





# Delay Matrix Source and Destination

- Set Source and Destination to be analyzed





# Useful Analysis Options

## ■ Time Restrictions

- Show All Path
- Show Only Longest Paths
- Show Only Shortest Paths

## ■ Cell Width

- Control matrix display

## ■ Cut Off I/O Pin Feedback

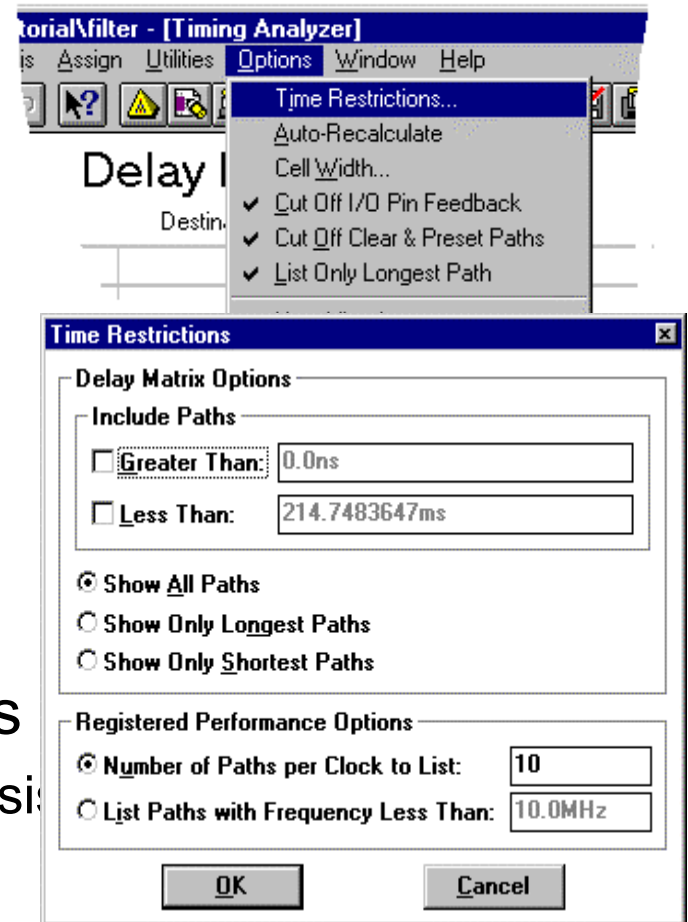
- See next page

## ■ Cut Off Clear & Preset Paths

- No clear or preset delay analysis

## ■ List Only Longest Path

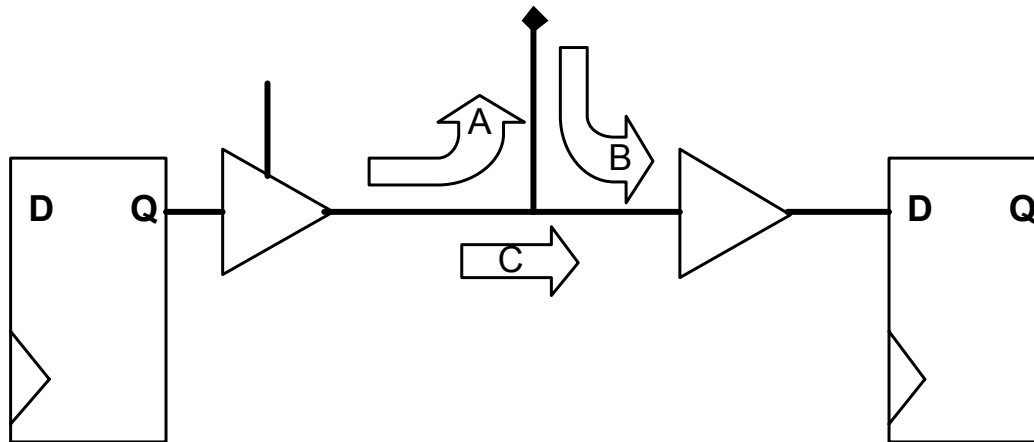
- List Path lists only longest path between two points





# Cut Off I/O Pin Feedback

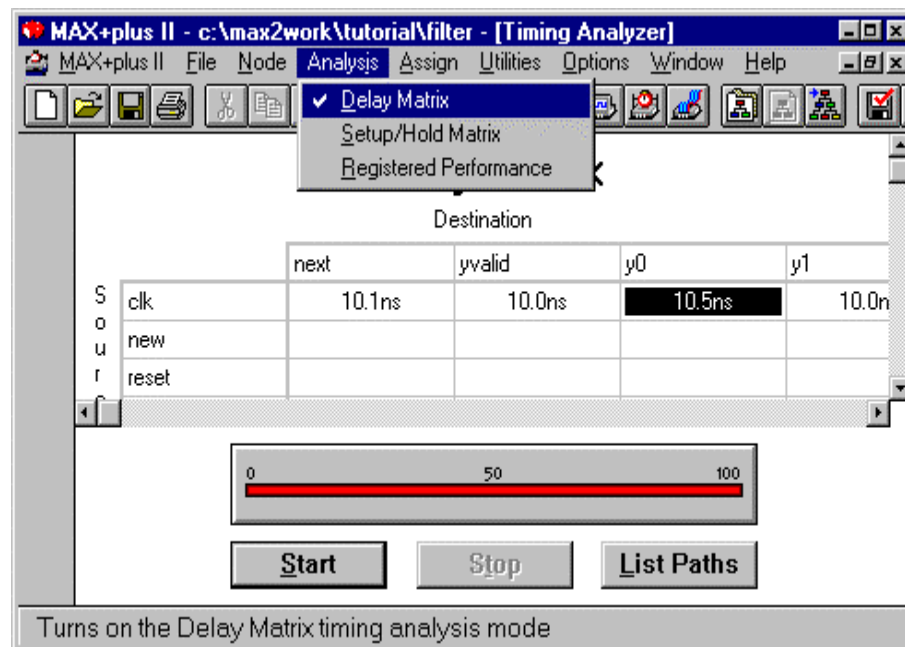
- ◆ Used to break bi-directional pin from the analysis
- ◆ When on, paths A and B true C false
- ◆ When off, path A, B and C are true





# Run Delay Matrix Analysis

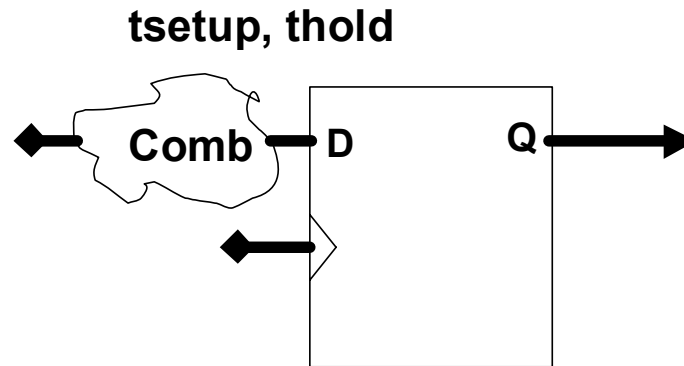
- Select Delay Matrix Analysis and click on Start button
- Matrix shows all paths, longest path, or shortest path depending on Time Restrictions option selected
- Use List





# Setup/Hold Matrix Analysis

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



## ◆ Setup

- $t_{\text{setup}} = t_{\text{data}} - t_{\text{clock}} + t_{\text{setup}}$

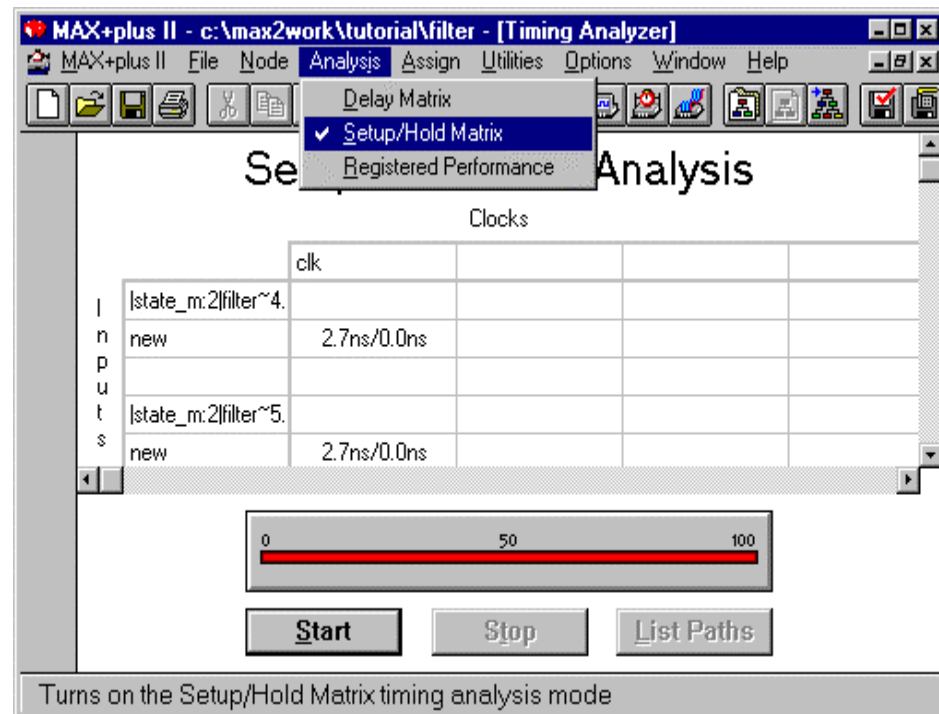
## ◆ Hold

- $t_{\text{hold}} = t_{\text{clock}} - t_{\text{data}} + t_{\text{hold}}$



# Run Setup/Hold Matrix Analysis

- ◆ Click on Start button
- ◆ Setup/Hold times are displayed with respect to the clocks



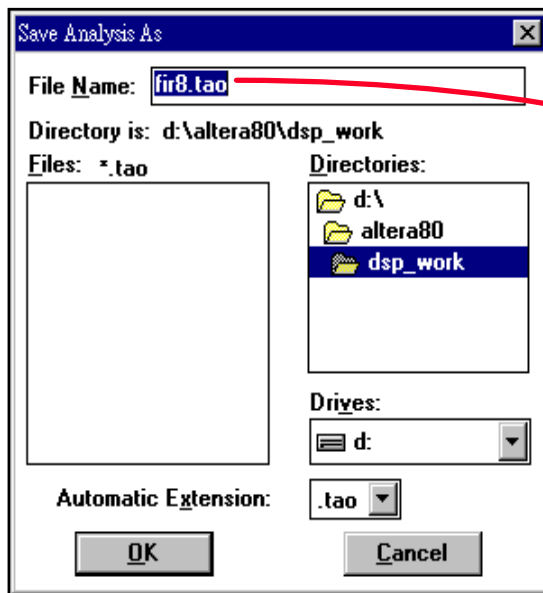


# Saving Timing Analysis Results

## ◆ Save the current Timing Analyzer results to a TAO File

- Timing Analyzer can save the information in the current timing analysis display to an ASCII-format Timing Analyzer Output file (\*.tao)

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



		Destination		
		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 interface. The main window displays a 'Delay Matrix' for a circuit. The matrix lists various signals as destinations and shows their delays from other signals. The 'clock' signal is highlighted in the matrix, showing a delay of 10.5ns to 'at\_altera'.

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

Below the matrix is a progress bar and buttons for 'Start', 'Stop', and 'List Paths'. The 'List Paths' button is circled in orange. A red arrow points from this button to the 'Messages - Timing Analyzer' window.

The 'Messages - Timing Analyzer' window shows a message: 'Info: Delay path from 'clock' to 'at\_altera': 10.5ns'. Below this message are buttons for 'Message', 'Locate', and 'Locate All'. The 'Locate' button is circled in orange. A red arrow points from this button to the 'auto\_max.blf - Text Editor' window.

The 'auto\_max.blf - Text Editor' window shows a VHDL code snippet for a machine state transition. The code is as follows:

```
auto_max.blf - Text Editor
)
VARIABLE
street_map : MACHINE
    OF BITS (q2,q1,q0)
    WITH STATES (
        YC,
        mp1d,
        ep1d,
        gdf,
        cnf,
        ...
    );
```



# Recommended Verification Flow

## ◆ Functional simulation

- Perform functional simulation to verify the design functionality

## ◆ Timing Analysis

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

## ◆ Timing simulation

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

## ◆ On-board test

- Program FPGA/CPLD device(s) and test the function & timing in system



# Timing Analysis Recommendations

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



# Project Timing Analysis Summary

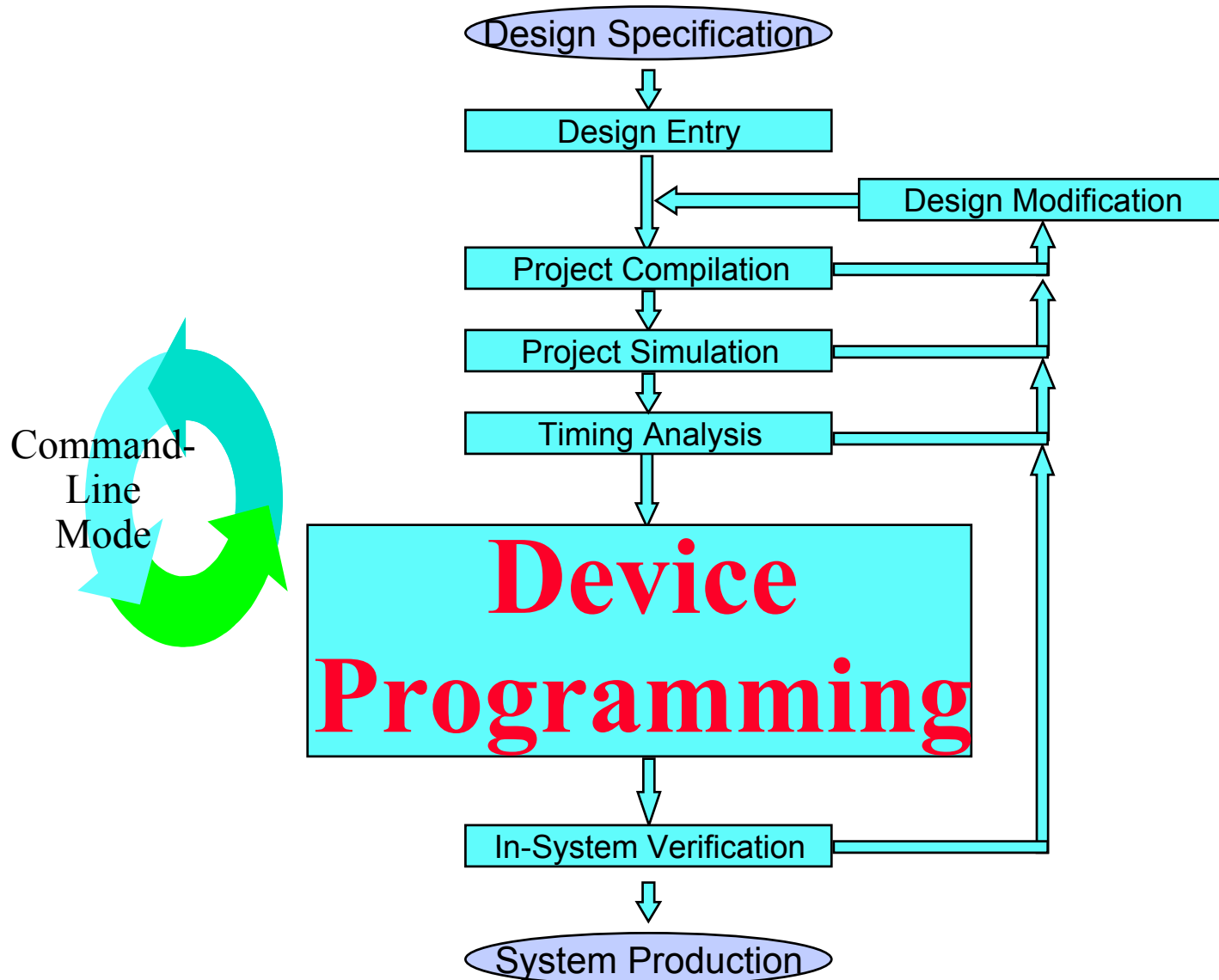
- ◆ **Timing Analyzer is a static timing analyzer**
- ◆ **Three modes of Timing Analysis**
  - Registered Performance
  - Delay Matrix
  - Setup/Hold Matrix
- ◆ **Provides ability to trace path through Floorplan Editor or design file**



# Device Programming

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







# Altera Provide Method

## ◆ Altera provide different methods for

- Program Device
  - MAX family
- Configure Device
  - FLEX family



# MAX Device

## ◆ Use Altera Stand Alone Programmer (ASAP2)

- <http://www.altera.com/html/products/asap2.html>

## ◆ Through JTAG port with ByteBlaster

- JTAG for Single Device (MAX or FLEX)
- JTAG Chain for Multiple Device (MAX & FLEX)
- JAM for Single/Multiple Device (MAX & FLEX)

## ◆ 3rd Programmer

- Data I/O
  - <http://www.data-io.com>
- BP MicroSystem
  - <http://www.bpmicro.com>



# FLEX Device

## ◆ Through JTAG port with ByteBlaster

- JTAG for Single Device (FLEX or MAX)
- JTAG Chain for Multiple Device (FLEX & MAX)

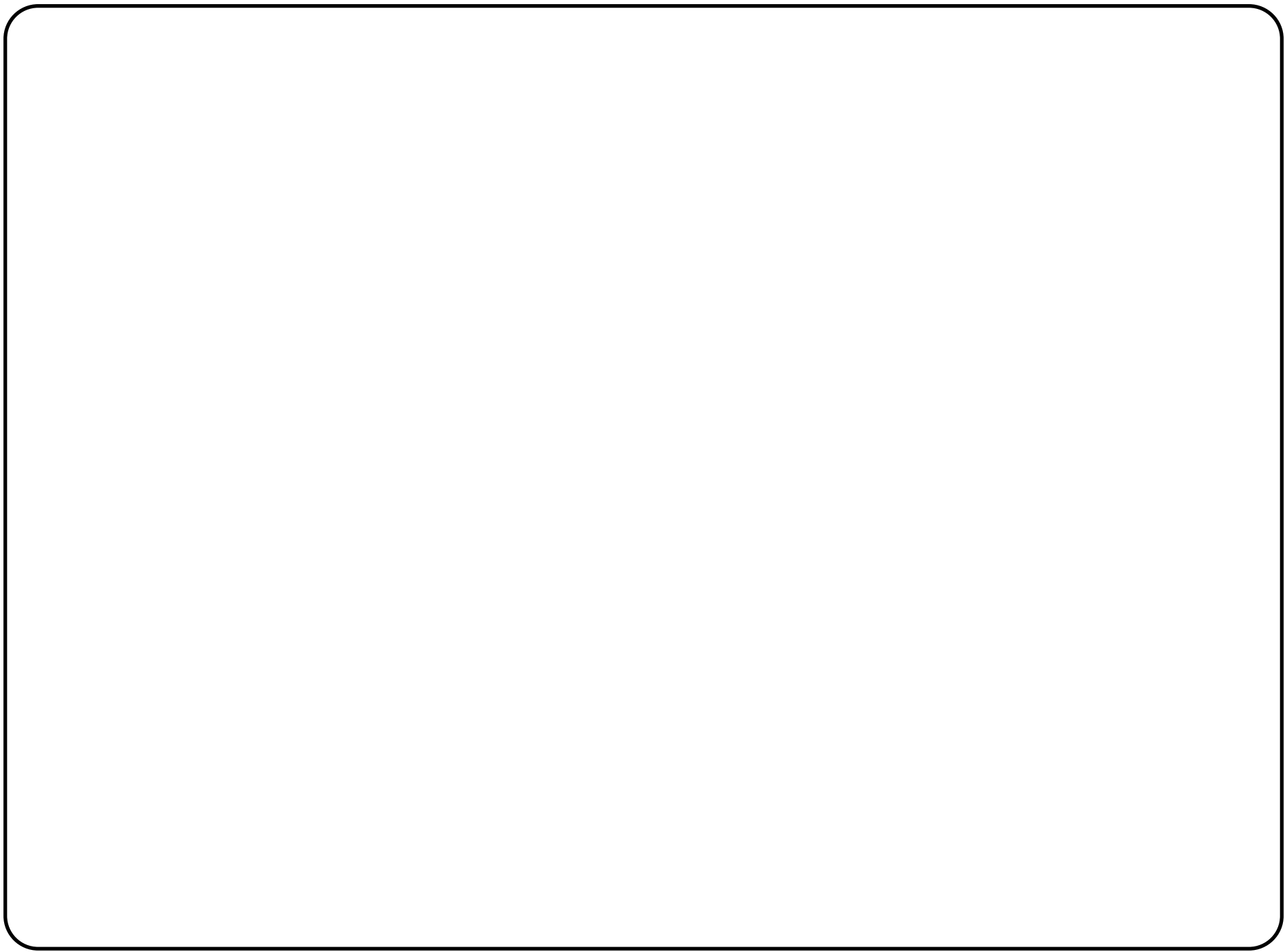
## ◆ Through PS port with ByteBlaster

- FLEX for Single Device
- FLEX Chain for Multiple Device

## ◆ Serial PROM

- EPC1 (1Mbits, good for 6K/8K/10K)
- EPC1441(441Kbits, good for 6K/8K/10K10, 10K20, 10K30)
- EPC1213 (213Kbits, only for 8K)
- EPC1064 (64Kbits, only for 8K)







# Configuration File Sizes

## ◆ FLEX device configuration file sizes

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

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

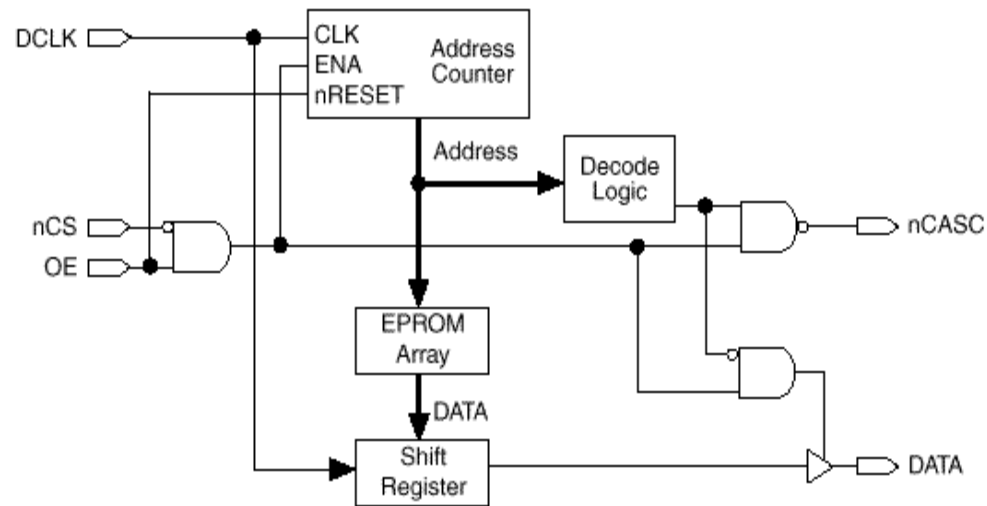
## ◆ Altera's serial configuration EPROMs for FLEX devices

- 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)

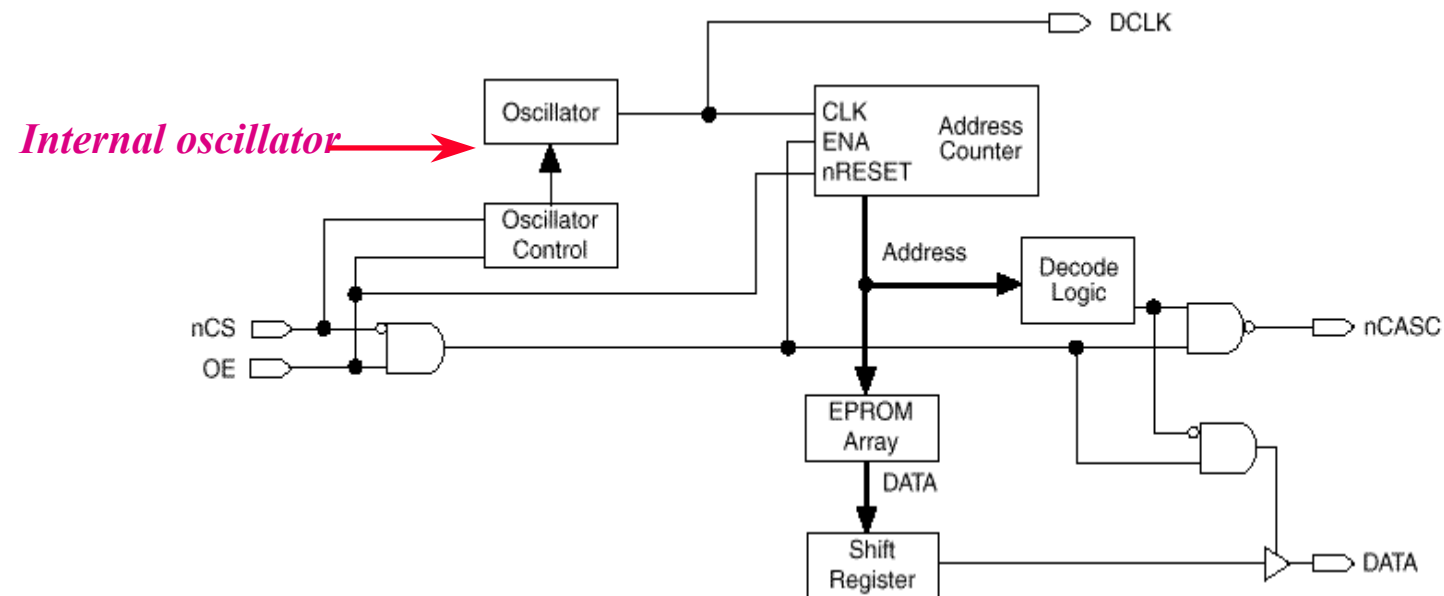
◆ EPC1064, EPC1213, or EPC1 in FLEX 8000A mode





# 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.



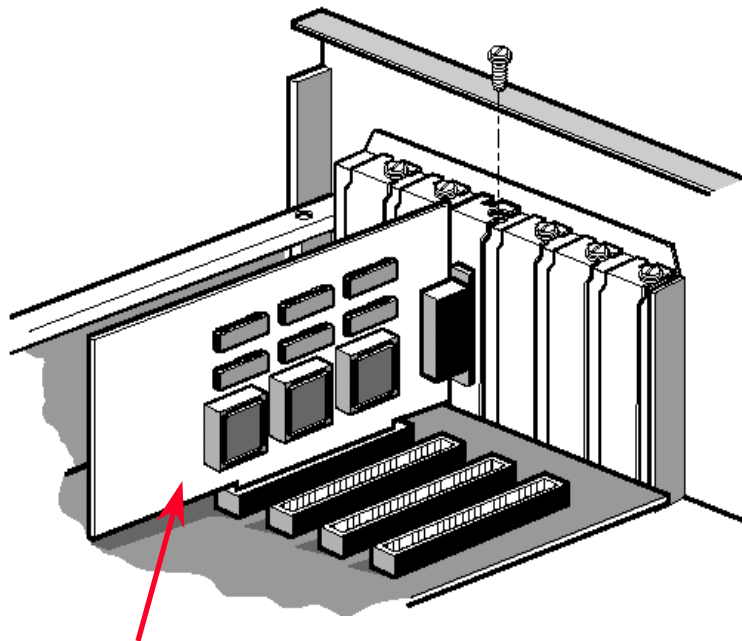
# Altera Stand-Alone Programmer

## ◆ PL-ASAP2: Altera stand-alone hardware programmer

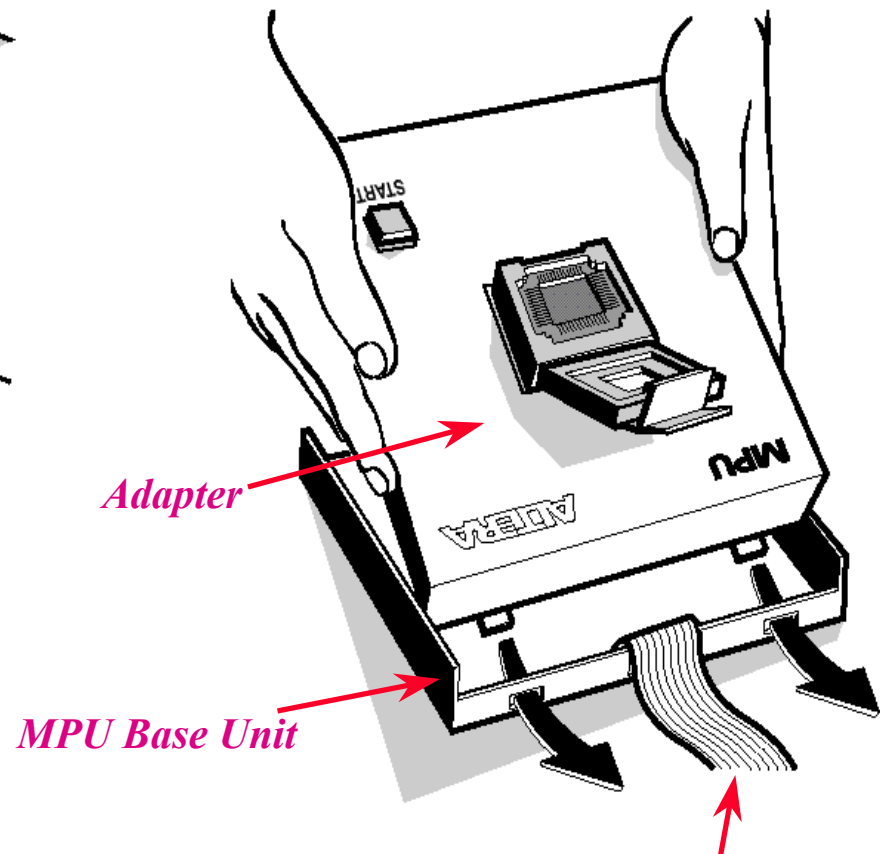
- The Altera stand-alone programmer, PL-ASAP2, together with the appropriate programming adapters, supports device configuration and programming for Altera devices
  - All MAX devices
  - Altera serial configuration EPROM: EPC1/V, EPC1064/V, EPC1213
- PL-ASAP2 includes an LP6 Logic Programmer card, an 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*



*MPU Base Unit*

*Connect the 25-pin flat ribbon cable to the LP6 card*



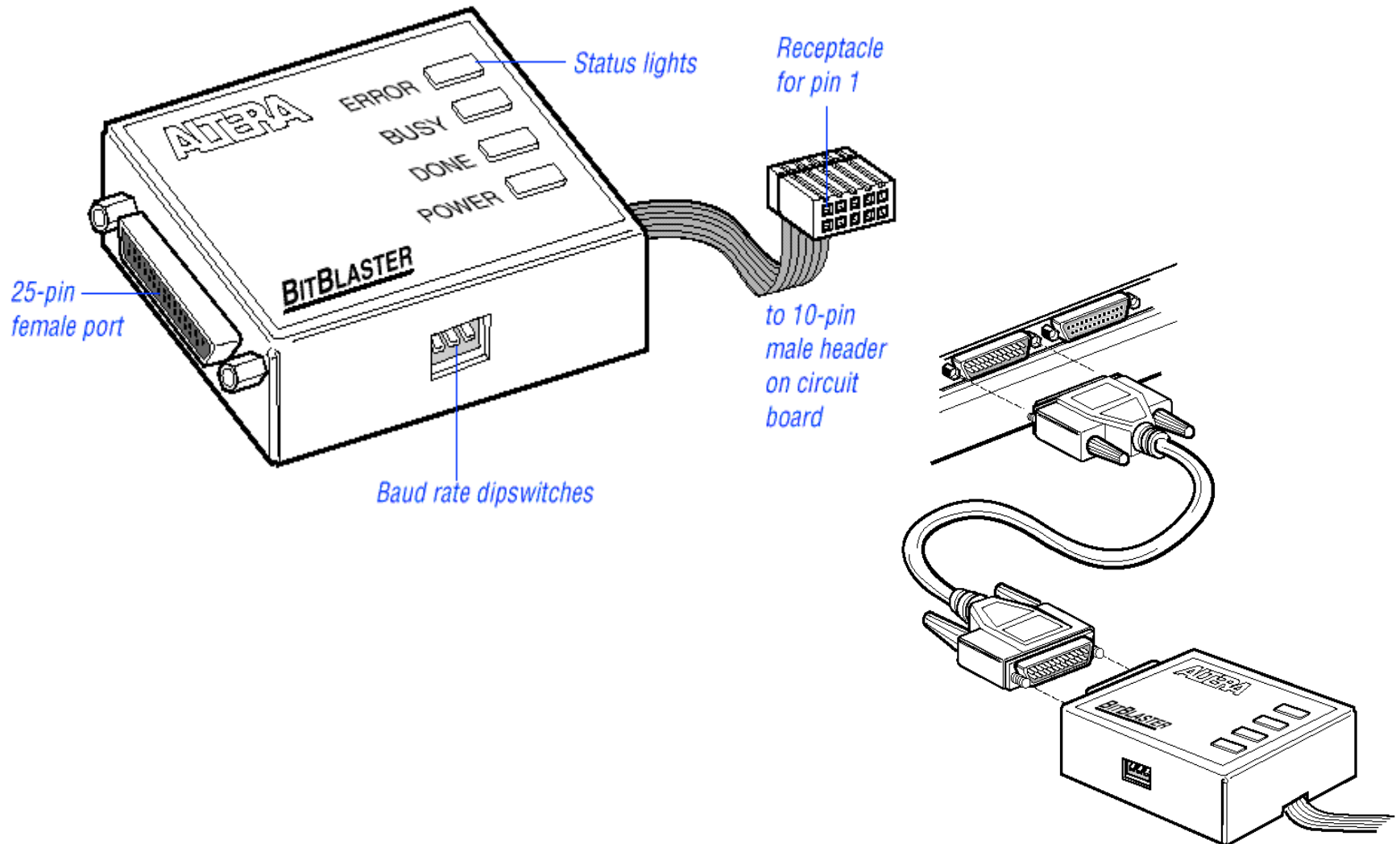
# BitBlaster Download Cable

## ◆ Altera BitBlaster serial download cable

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



# Installing BitBlaster





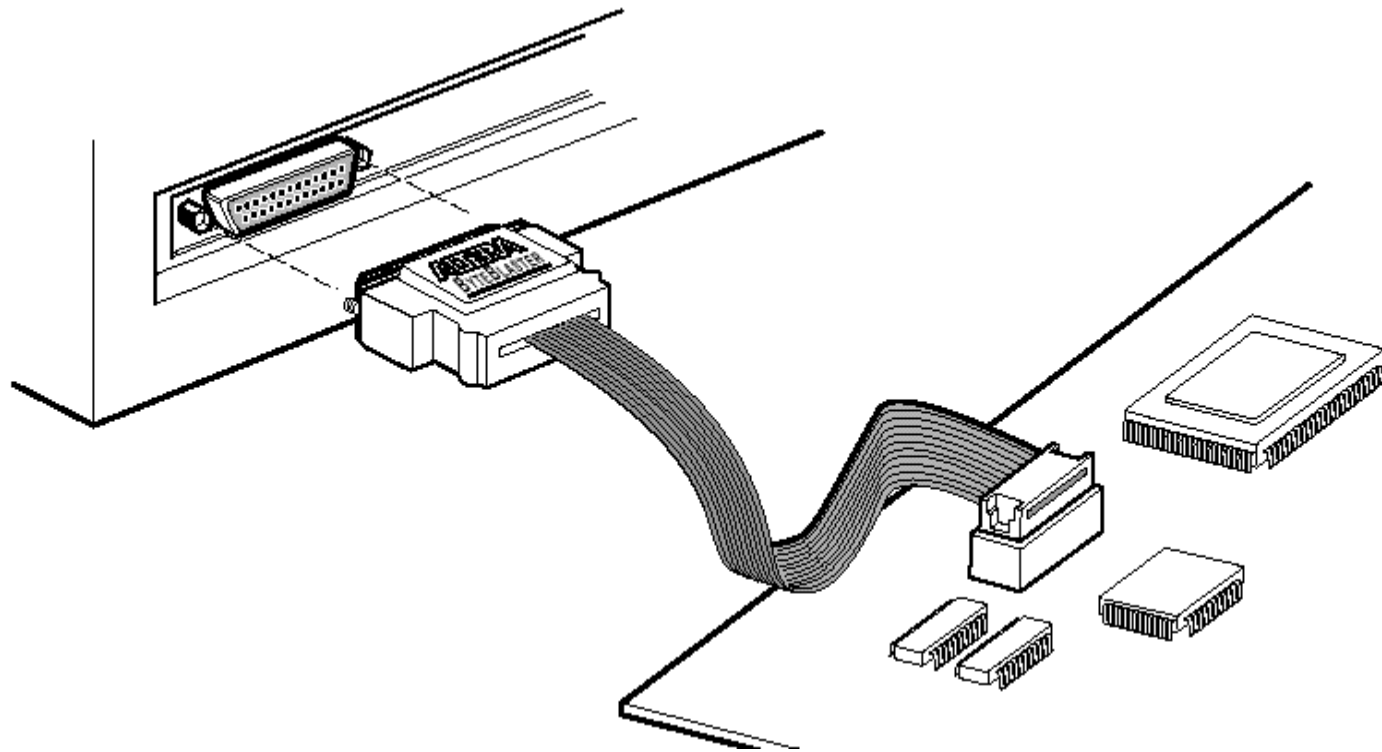
# ByteBlaster Download Cable

## ◆ 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



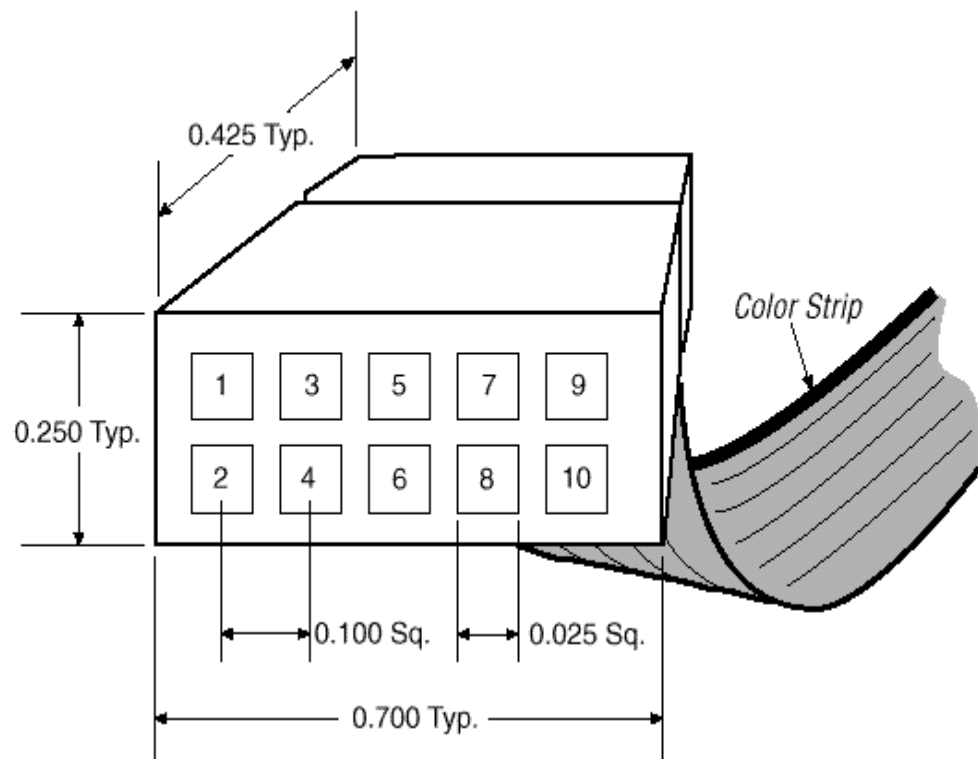
# Installing ByteBlaster





# 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**

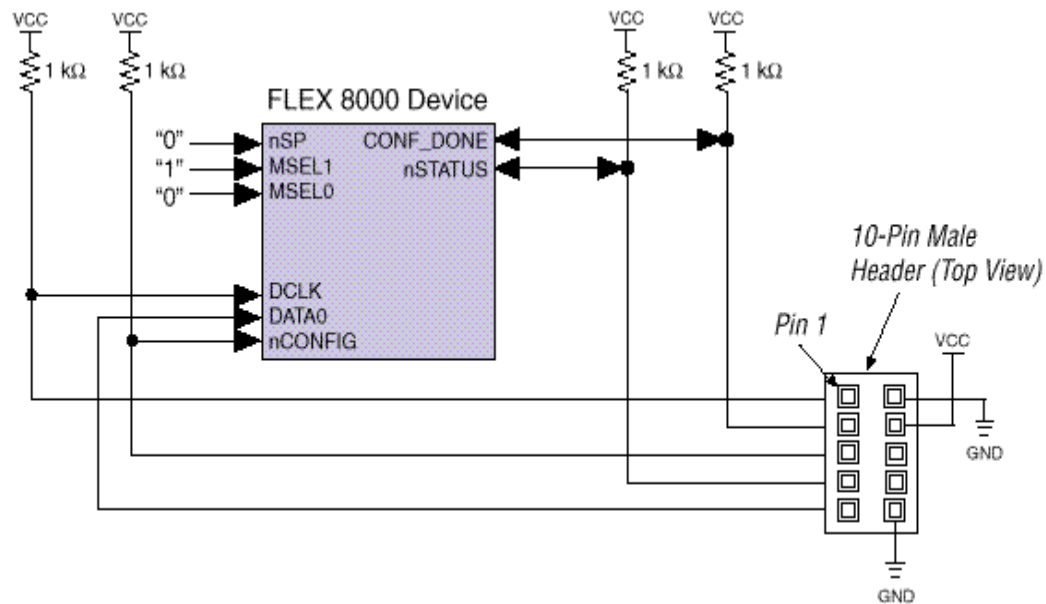
- FLEX 8000A Active Serial(AS) configuration with serial configuration EPROM
- FLEX 10K Passive Serial(PS) configuration with EPC1 configuration EPROM

## **◆ Parallel EPROM configuration**

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

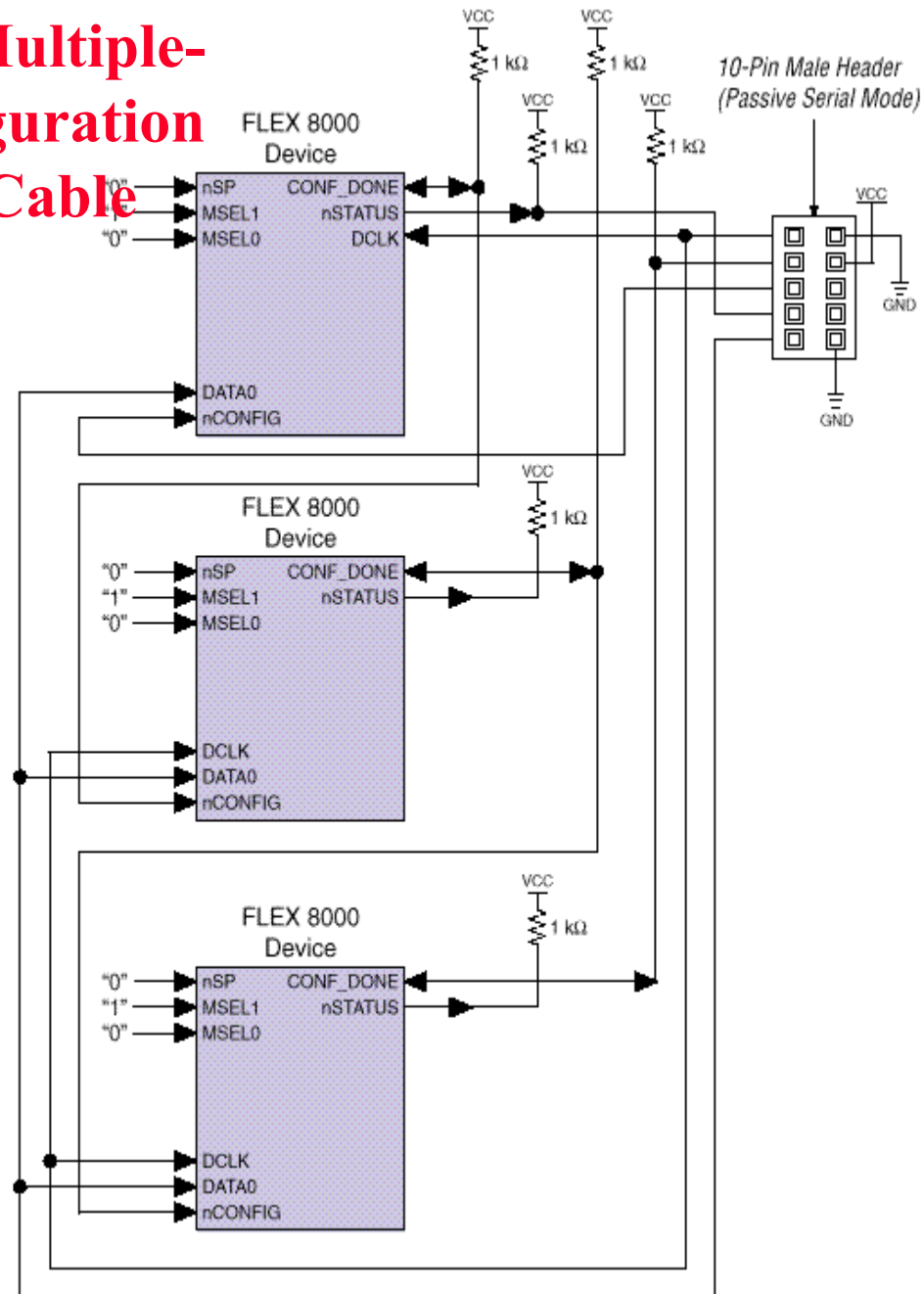


# FLEX 8000A Single-Device PS Configuration with the Download Cable



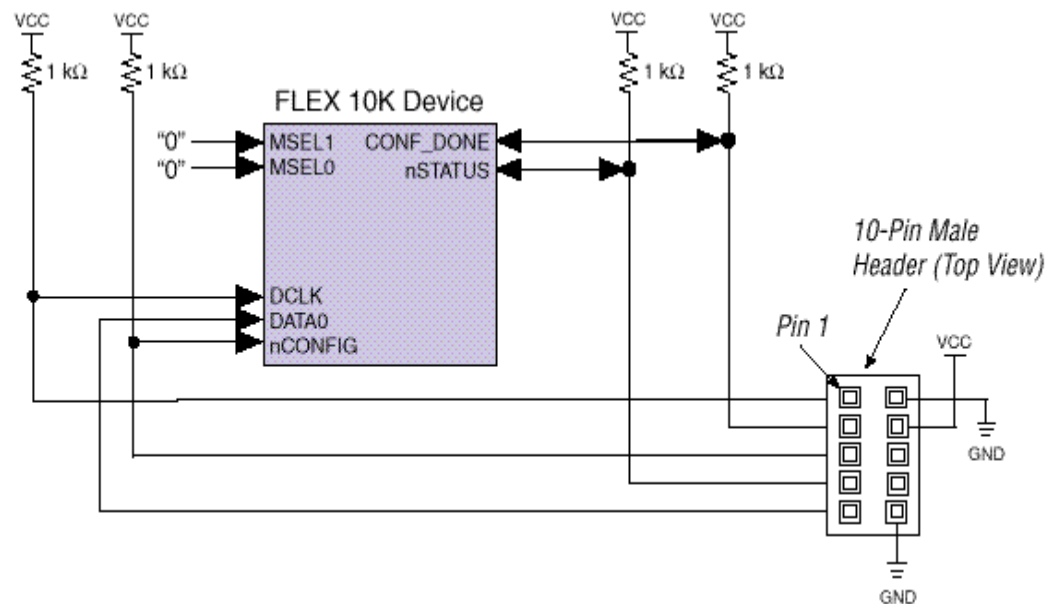


# FLEX 8000A Multiple-Device PS Configuration with the Download Cable





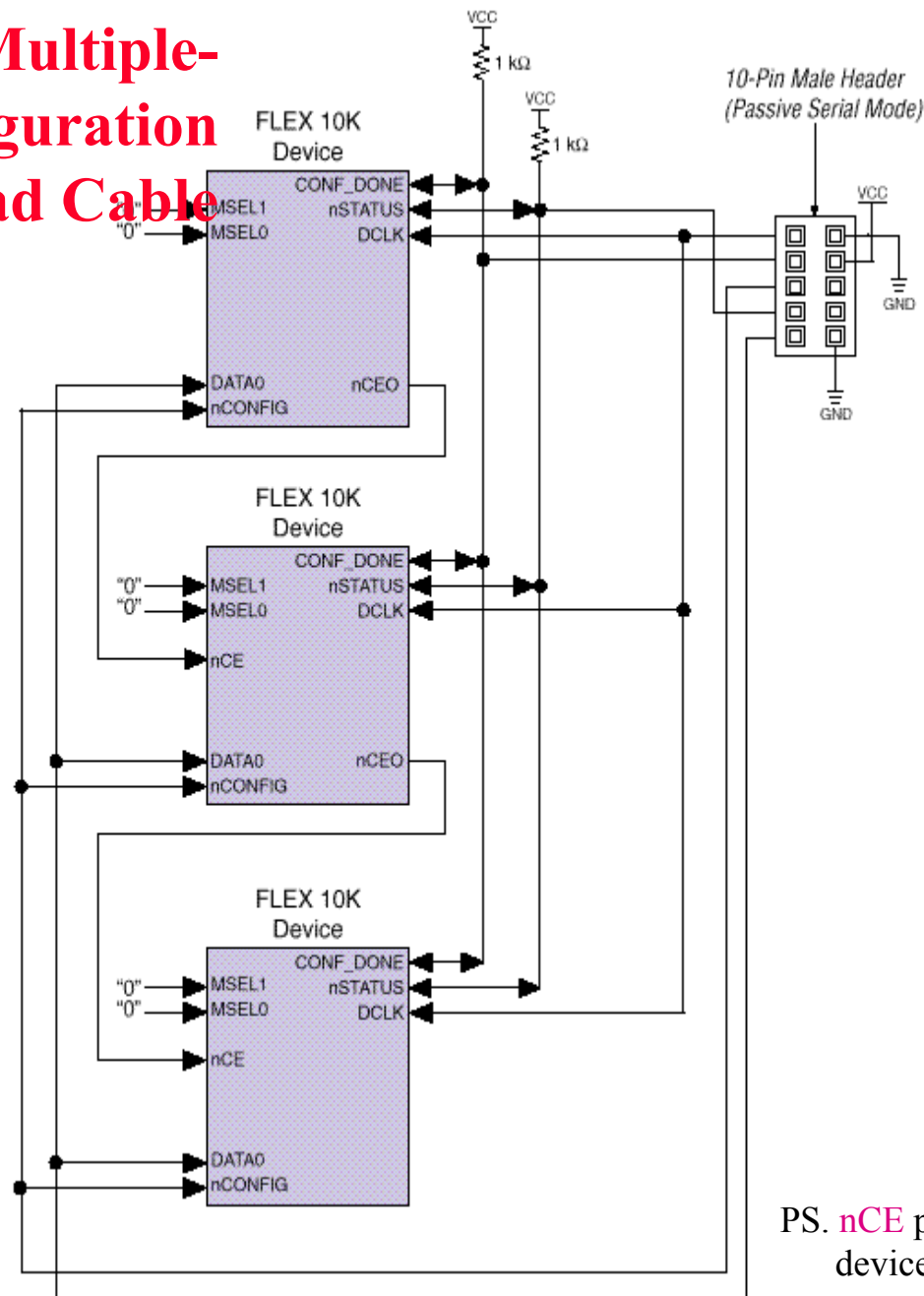
# FLEX 10K Single-Device PS Configuration with the Download Cable



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



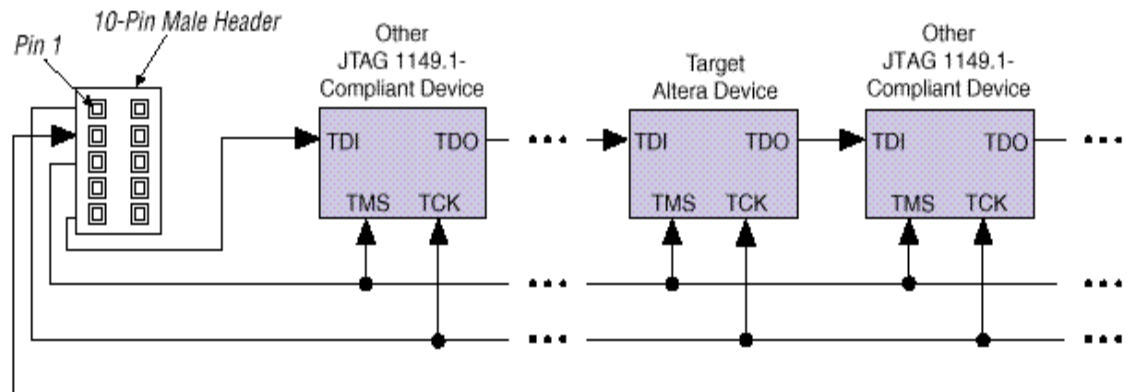
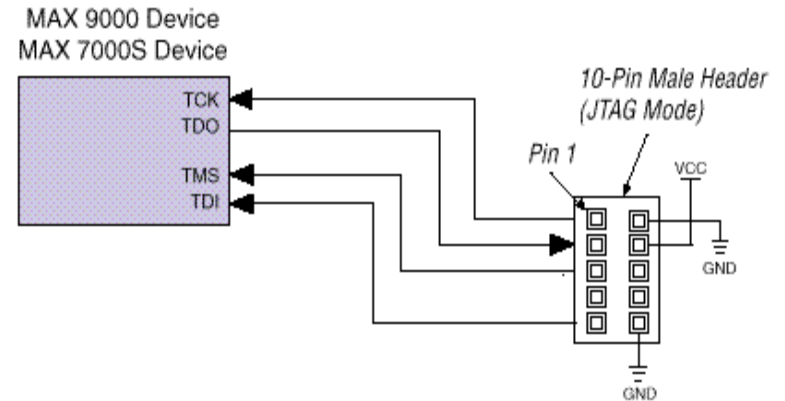
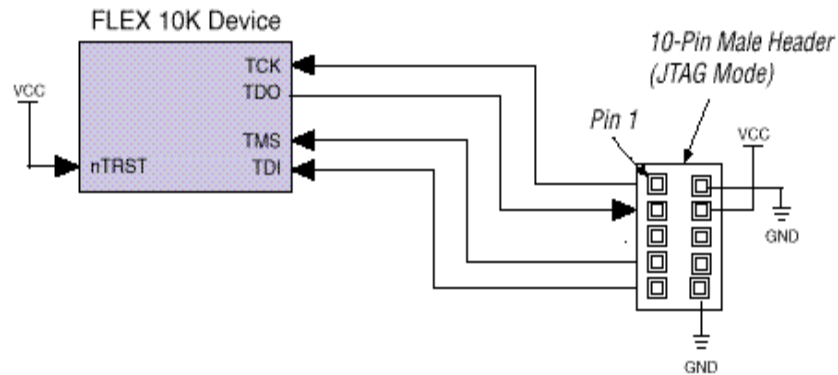
# FLEX 10K Multiple-Device PS Configuration with the Download Cable



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

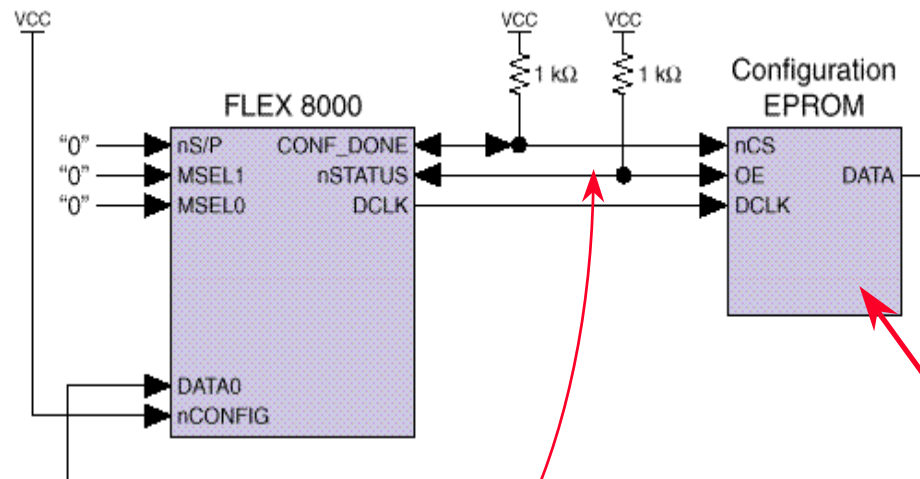


# JTAG Configuration with the Download Cable





# FLEX 8000A Configuration EPROM Configuration (AS Mode)

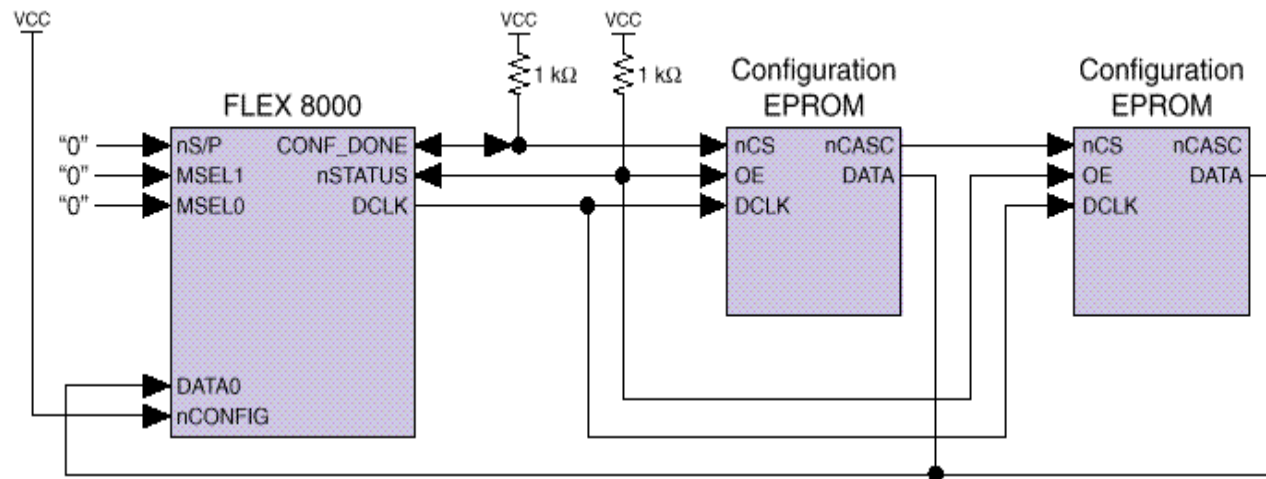


AS configuration with automatic reconfiguration on error (nStatus pin is connected to OE pin of the configuration EPROM and when "Auto-Restart Configuration on Frame Error" option bit is turned on)

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

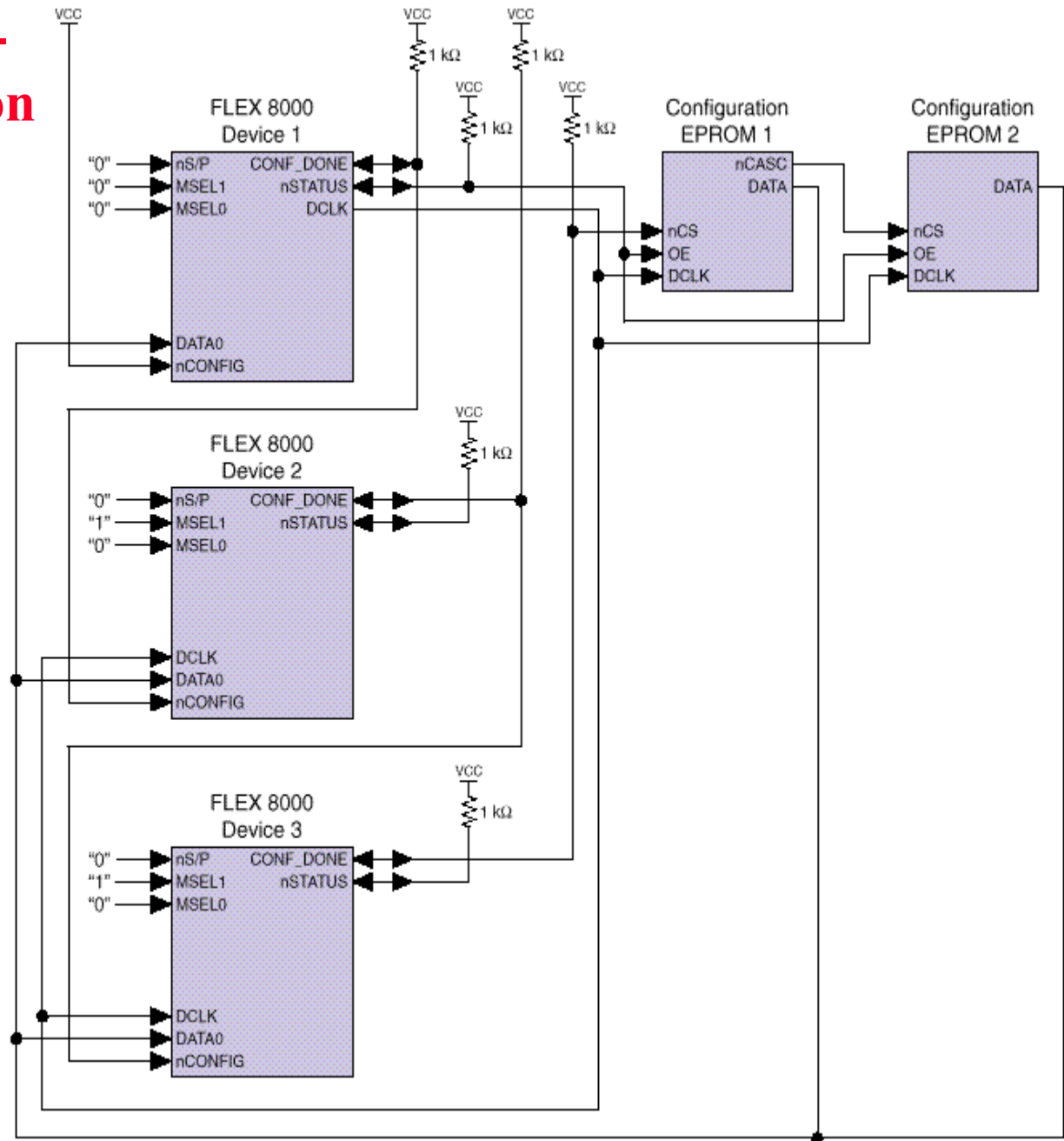


# FLEX 8000A Multiple Configuration EPROMs Configuration



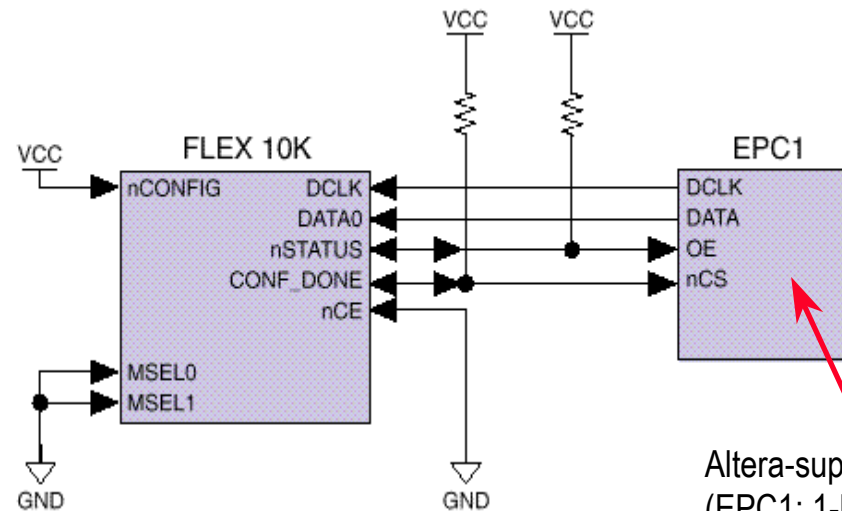


# FLEX 8000A Multi-Device Configuration EPROM





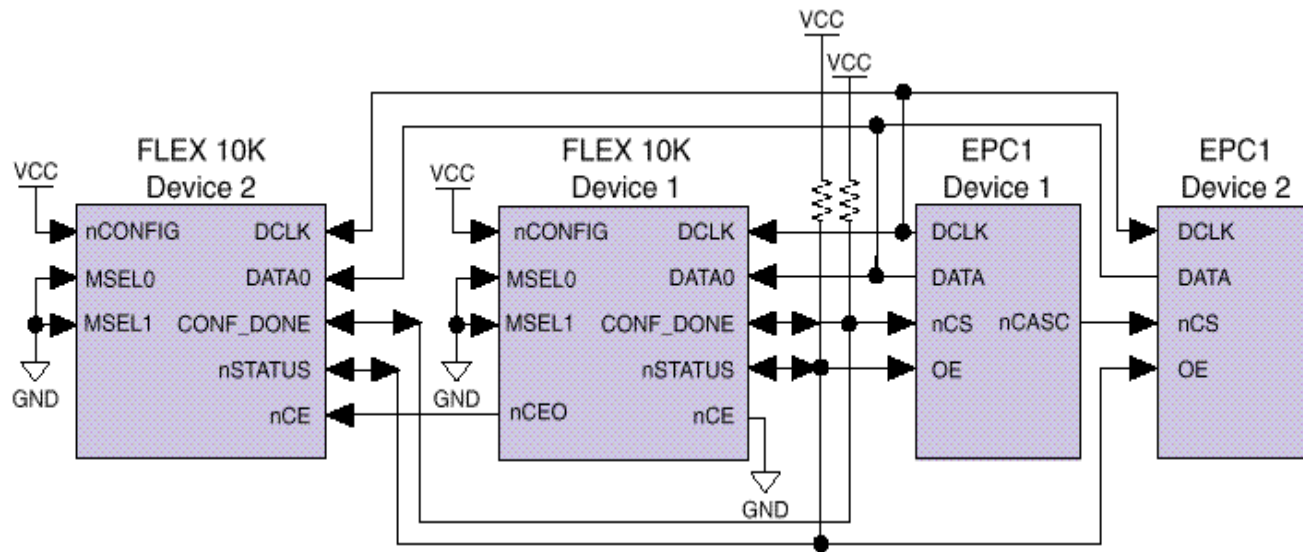
# FLEX 10K Configuration EPROM Configuration (PS Mode)



Altera-supplied serial EPC1 configuration EPROM  
(EPC1: 1-Mbit EPROM)

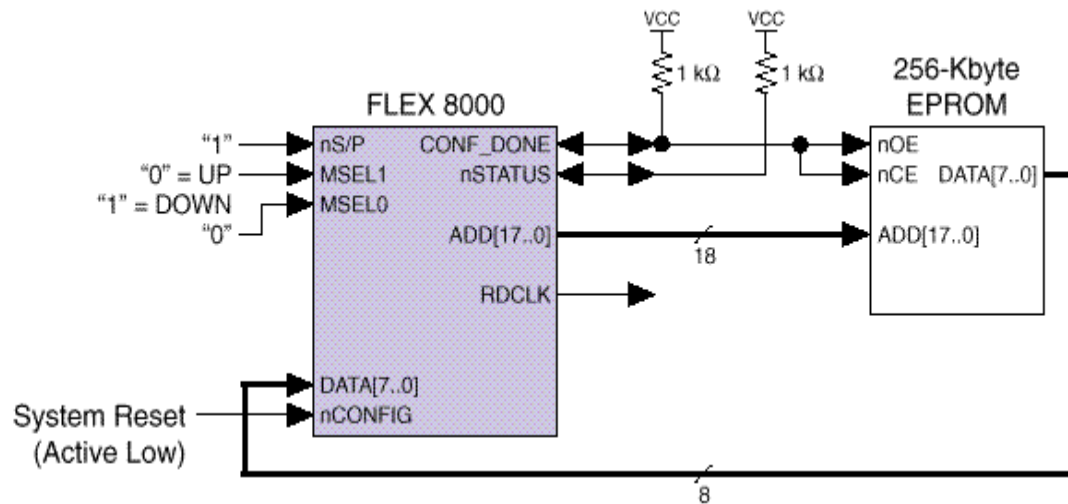


# FLEX 10K Multi-Device Configuration EPROM Configuration



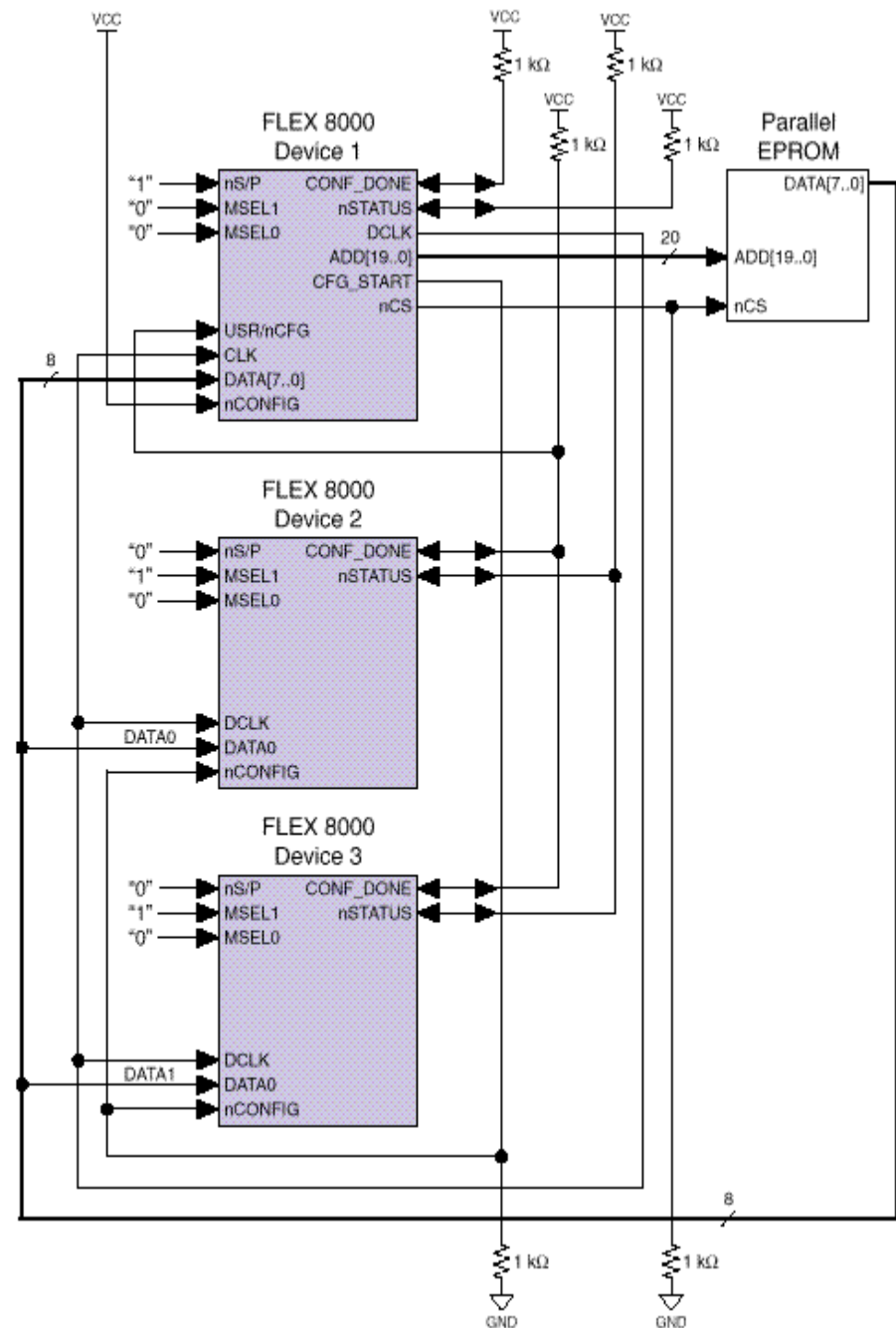


# FLEX 8000A APU & APD Configuration





# FLEX 8000A Multi-Device APU & Configuration APD

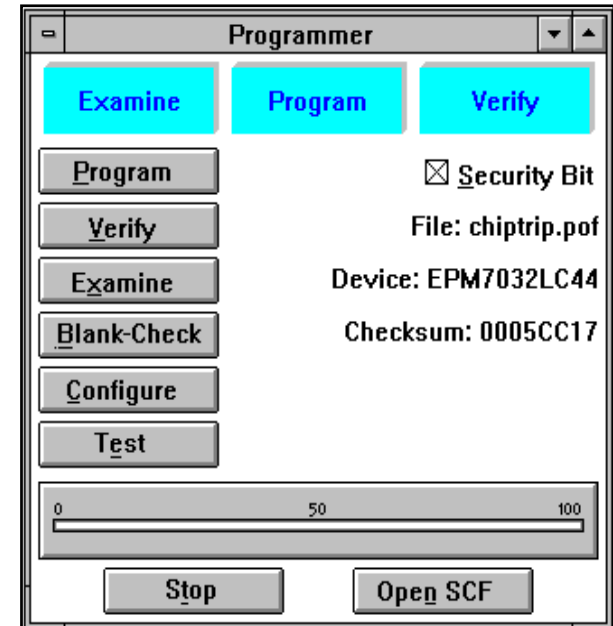




# MAX+PLUS II Programmer

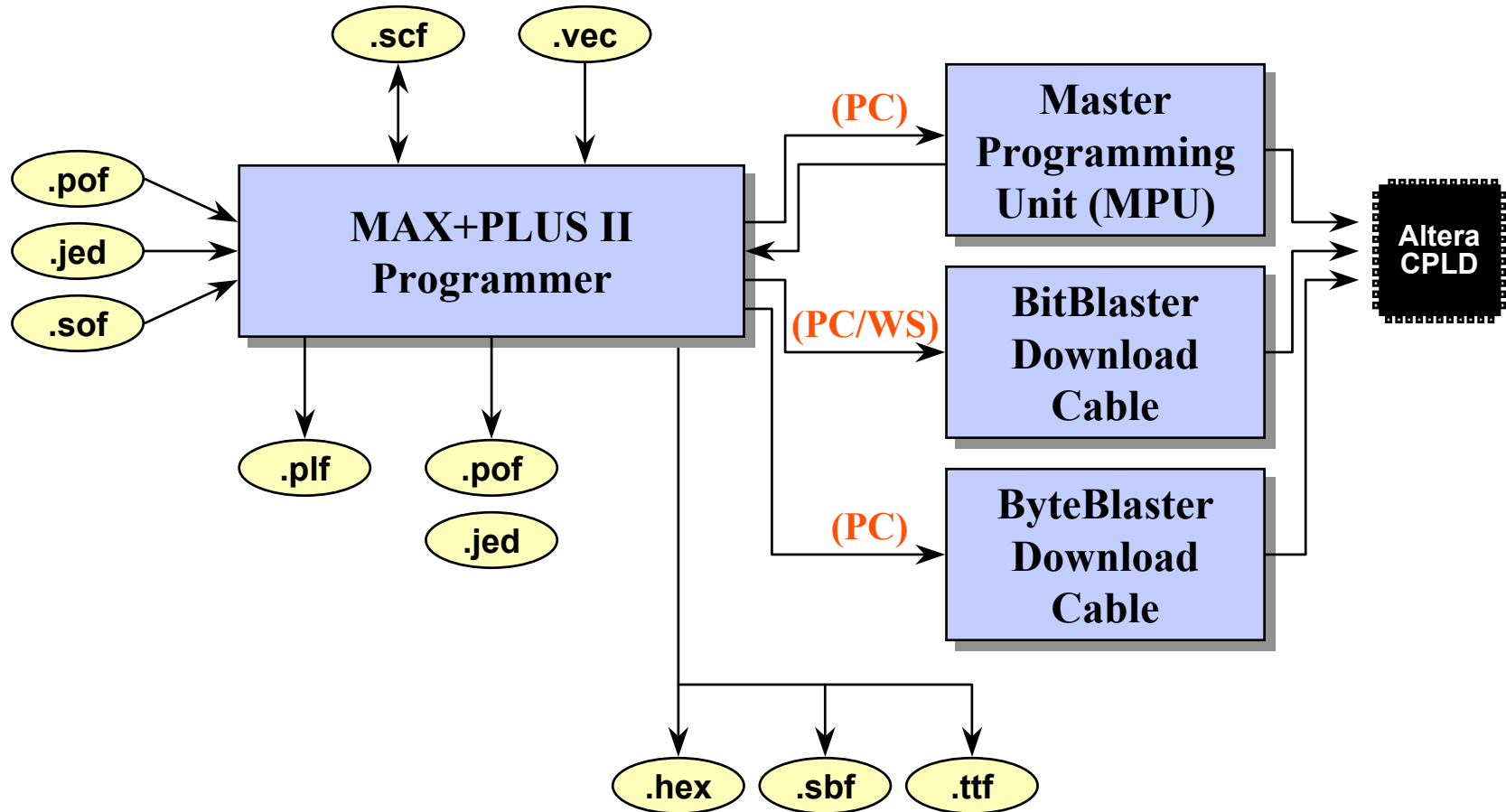
## ◆ To program or configure Altera devices

- 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

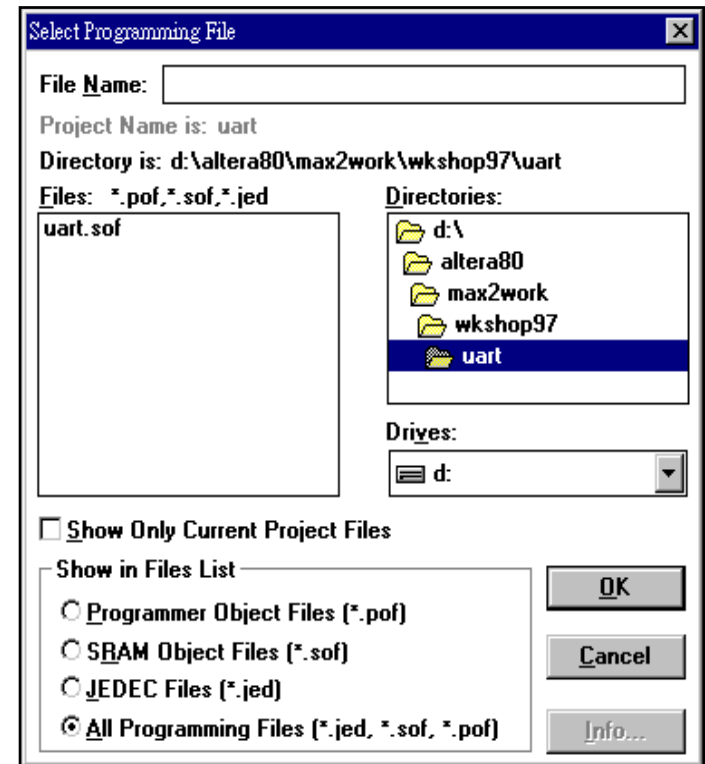
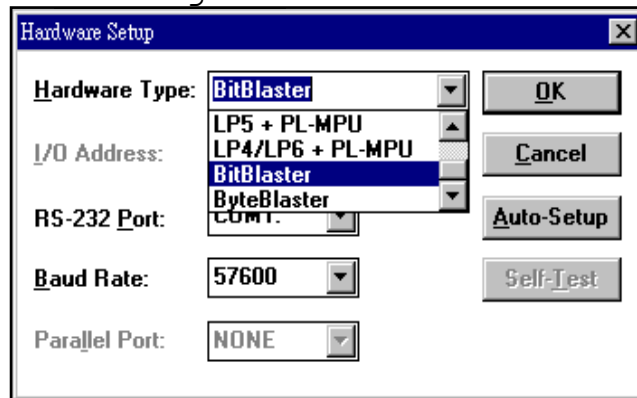
- Setup the hardware

*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

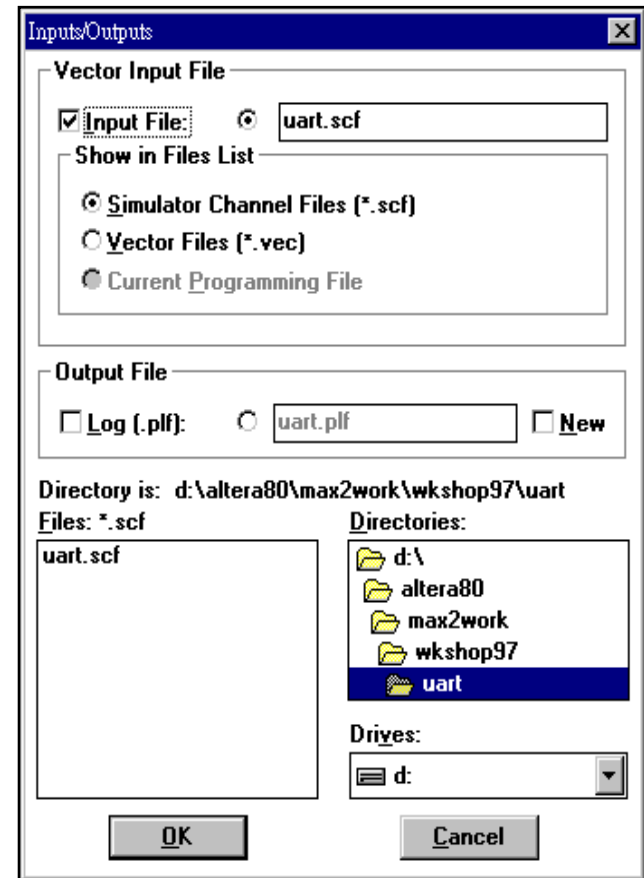




# Functional Test on Device

## ◆ 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



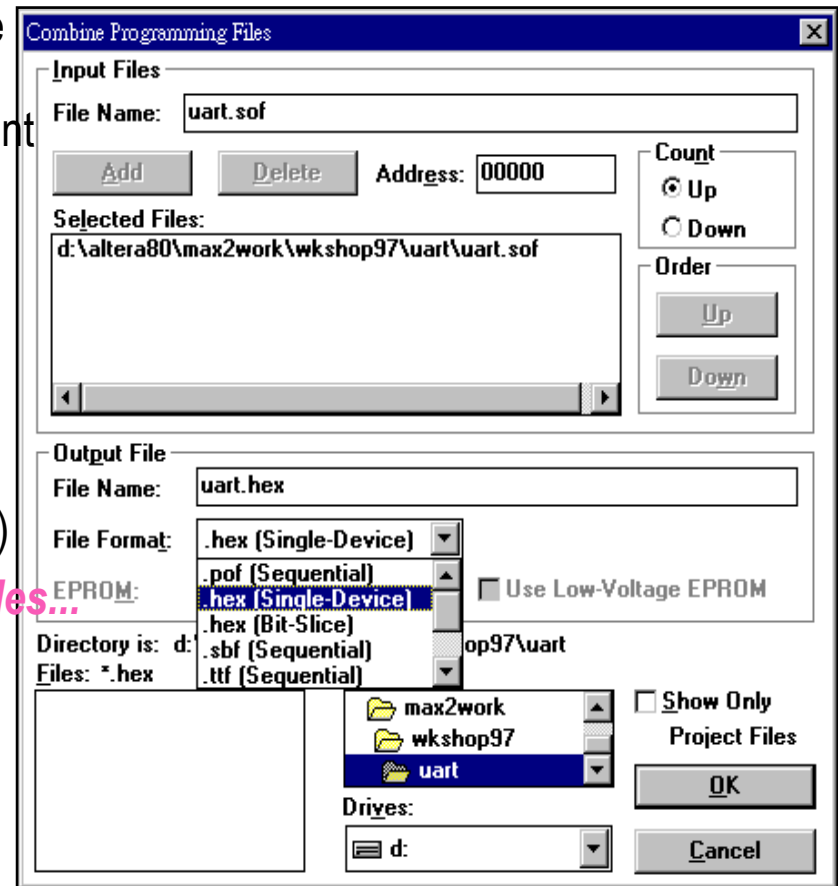


# Converting or Combining Programming Files

## ◆ 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...

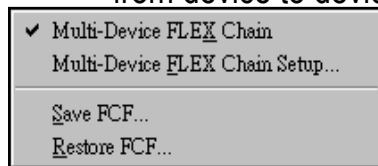




# Configuring Multiple FLEX Devices

## ◆ 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

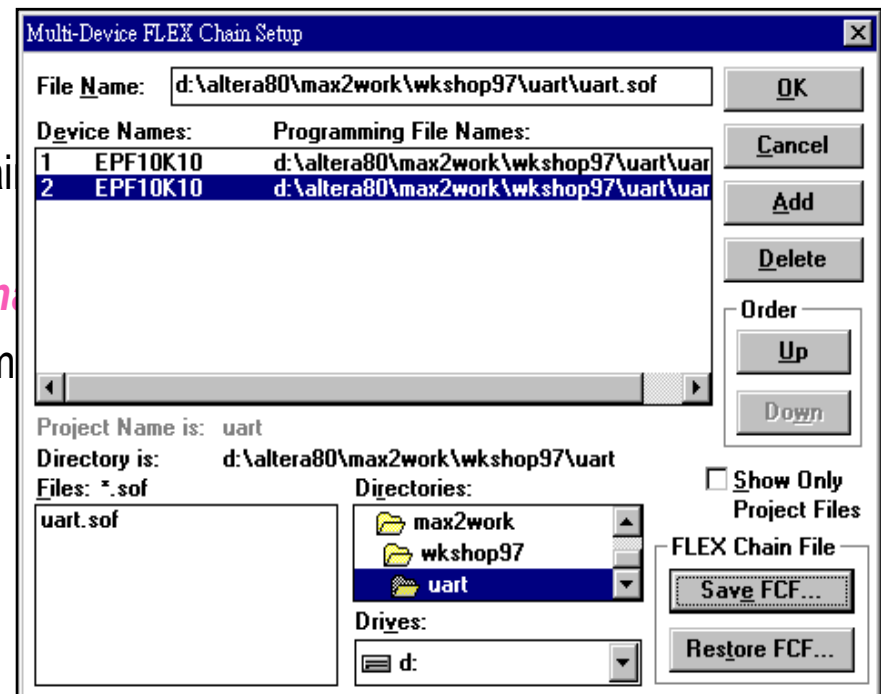
Menu: *FLEX -> Save FCF...*

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

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

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

- Click Configure button on Programm... to start configuration

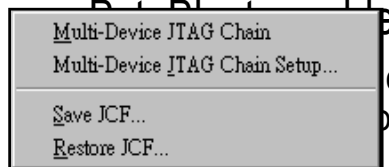




# Programming Multiple JTAG Devices

## ◆ 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



chain: a series of devices through which programming and/or configuration data is downloaded 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

The screenshot shows the 'Multi-Device JTAG Chain Setup' dialog box. It has a title bar with a close button. The main area is divided into several sections. At the top, there are two input fields: 'Device Name:' with a dropdown menu showing 'EPF10K10', and 'Programming File Name:' with a text box containing 'd:\altera80\max2work\wkshop97\uart\u'. To the right of these fields are 'OK' and 'Cancel' buttons. Below these fields are two buttons: 'JTAG Device Attributes...' and 'Select Programming File...'. The main body of the dialog contains two columns: 'Device Names:' and 'Programming File Names:'. The 'Device Names' column lists two items: '1 EPF10K10' and '2 EPM7064S'. The 'Programming File Names' column lists two items: 'd:\altera80\max2work\wkshop97\uart\uar' and 'd:\altera80\max2work\tutorial\chiptrip.pof'. To the right of this list are buttons for 'Add', 'Delete', 'Delete All', 'Order', 'Up', and 'Down'. Below the list, a status line reads 'List contains 2 devices with total instruction register length of 20'. At the bottom, there is a section for 'Use Hardware' with a checkbox and a text box containing 'Hardware has not been used to detect JTAG chain information', and a 'Detect JTAG Chain Info' button. On the far right, there is a section for 'JTAG Chain File' with 'Save JCF...' and 'Restore JCF...' buttons.

Device Name:	Programming File Name:
EPF10K10	d:\altera80\max2work\wkshop97\uart\u

Device Names:	Programming File Names:
1 EPF10K10	d:\altera80\max2work\wkshop97\uart\uar
2 EPM7064S	d:\altera80\max2work\tutorial\chiptrip.pof

List contains 2 devices with total instruction register length of 20

Use Hardware ☐ Hardware has not been used to detect JTAG chain information

Detect JTAG Chain Info

JTAG Chain File

Save JCF... Restore JCF...

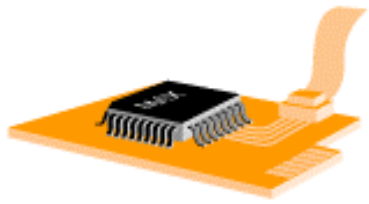


# Details about Device Programming

## ◆ 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*





# Getting Help

## ◆ CIC technical support: 周育德

- Phone : (03)5773693 ext. 148
- Email : [steven@cic.edu.tw](mailto:steven@cic.edu.tw)
- News : [nsc.cic](http://nsc.cic)
- WWW : [http://www.cic.edu.tw/chip\\_design/design\\_intr/altera/](http://www.cic.edu.tw/chip_design/design_intr/altera/)
- ftp-site : <ftp://ftp.cic.edu.tw/pub> (140.126.24.62) under [/pub/doc/manual/Altera](#)

## ◆ To buy Altera chips, hardware or demo boards:

- Contact Galaxy Far East Corp. 茂綸公司楊樂麗小姐 (03)578-6766 ext. 220

## ◆ Altera technical support on Internet

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



# **MAX+plus II Lab**

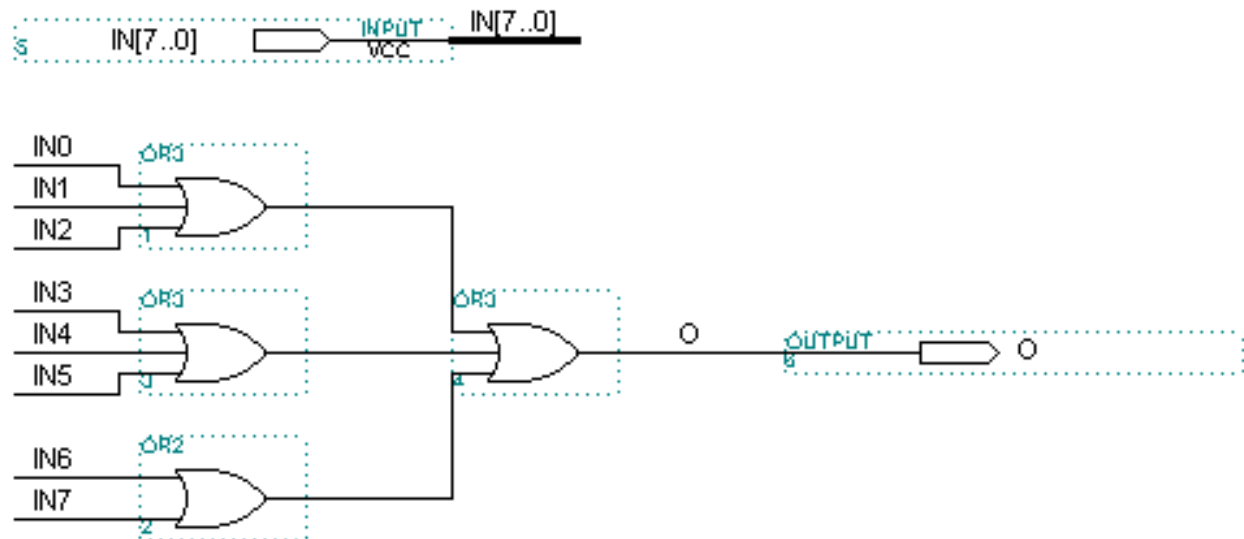
## **Fibonacci Generator**



# Lab 1 - myor8 I

## ◆ 用基本的邏輯閘電路

- File > New (Graphic Editor File - .gdf)
- File > Save As... (myor8.gdf)
- File > Project... > Set Project to Current File
- 繪出下圖





# Lab 1 - myor8 II

## ◆ Save and Check

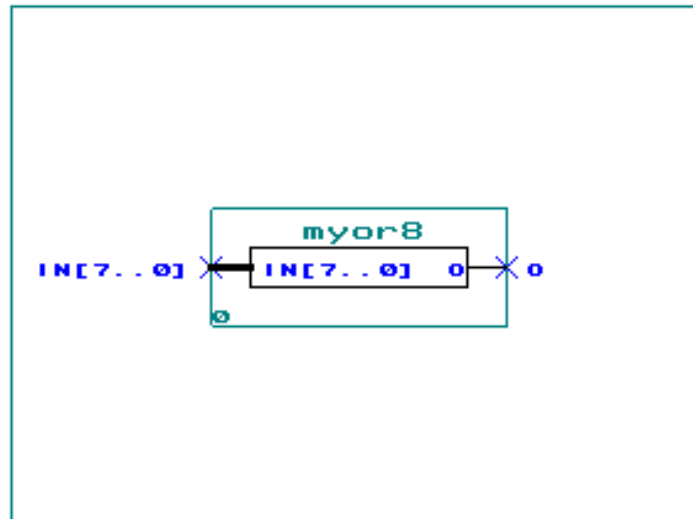
- File > Project ... > Save & Check (Ctrl + K)

## ◆ Generate Symbol

- File > Create Default Symbol

## ◆ View Symbol

- File > Open ( myor8.sym)





# Lab 2 - Disbounce I

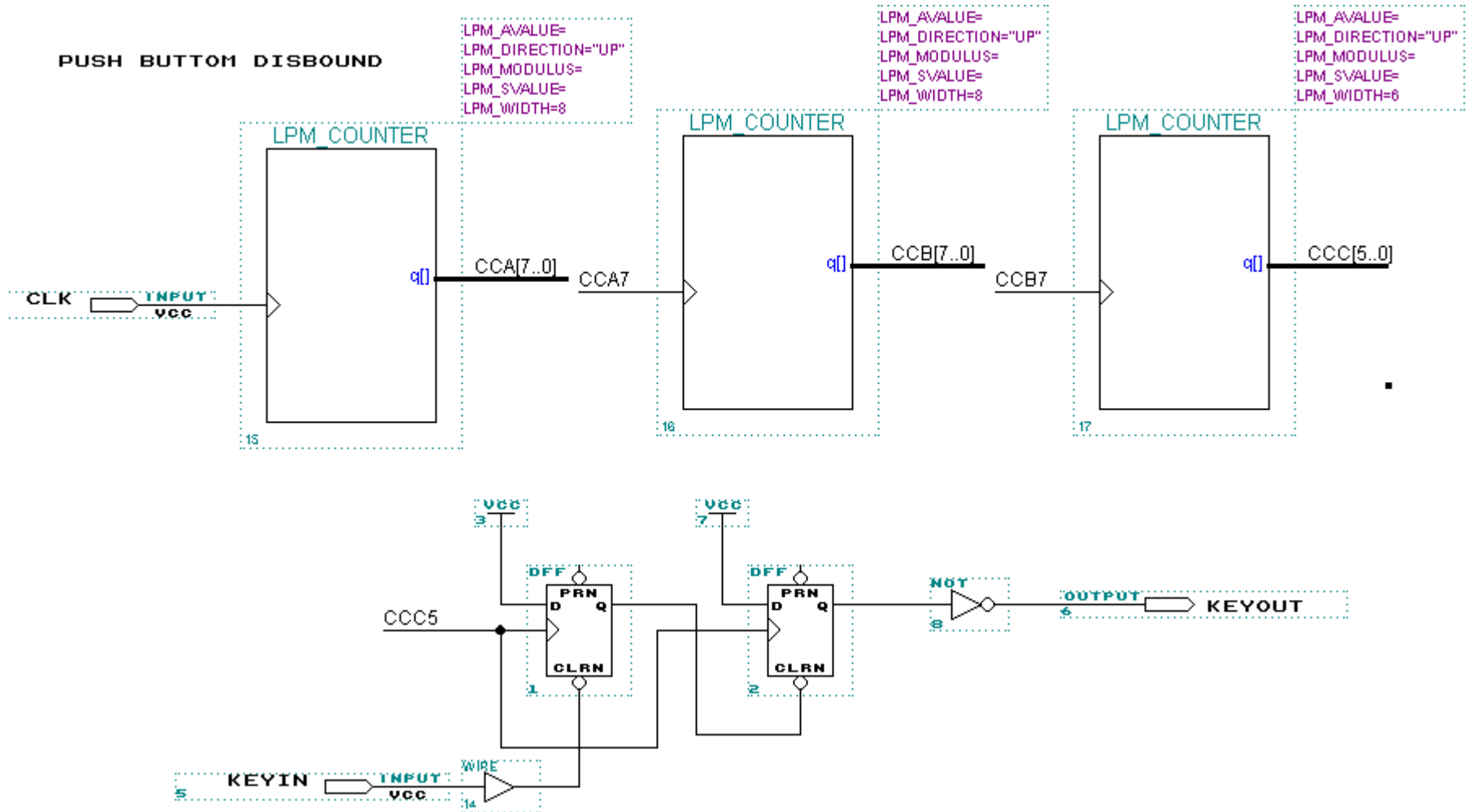
## ◆用Schematic設計防止彈跳電路

- File > New (Graphic Editor File - .gdf)
- File > Save As... (disbounce.gdf)
- File > Project... > Set Project to Current File
- 繪出下頁電路, 請注意LPM\_COUNTER的I/O與parameters的設定
- File > Project ... > Save & Check (Ctrl + K)
- File > Create Default Symbol



# Lab 2 - Disbounce II

## PUSH BUTTON DISBOUND





# Lab 3 - 7segment

## ◆用AHDL來設計七段顯示器解碼電路

- File > New
  - (Text Editor File - .tdf)
- File > Save as
  - (7segment.tdf)
- 輸入AHDL
- File > Project > Save & Check (Ctrl + K)
- File > Create Default Symbol

```

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

```



# Lab 4 - fib\_top I

◆ **File > New (Graphic Editor File - .gdf)**

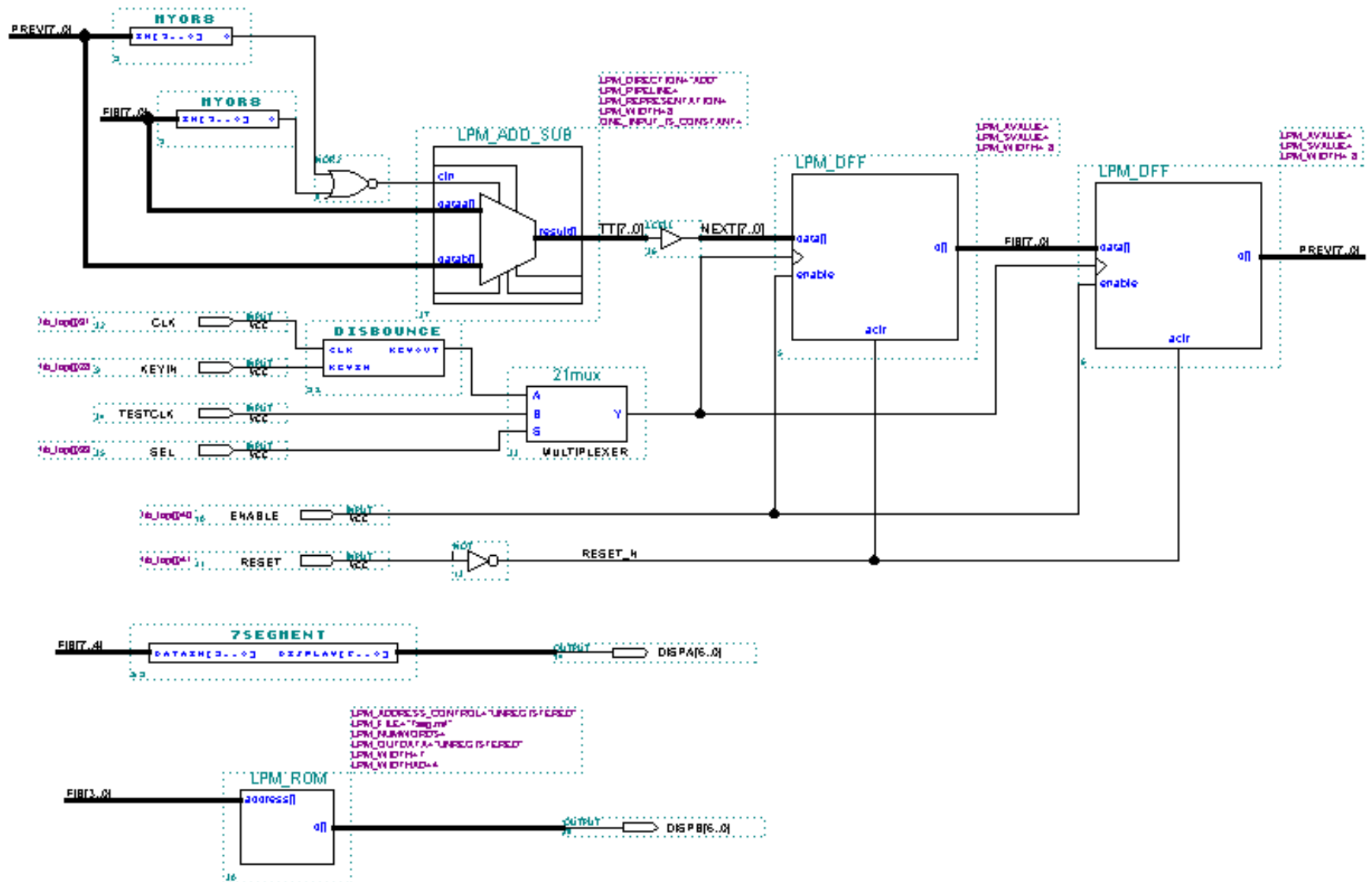
◆ 依據下列數圖完成電路

◆ **File > Project > Save & Check (fib\_top.v)**

- File > New (Graphic Editor File - .gdf)
- File > Save As... (fib\_top.gdf)
- File > Project... > Set Project to Current File
- 繪出下頁電路, 請注意LPM的I/O與parameters的設定
- File > Project ... > Save & Check (Ctrl + K)
- File > Create Default Symbol



# 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

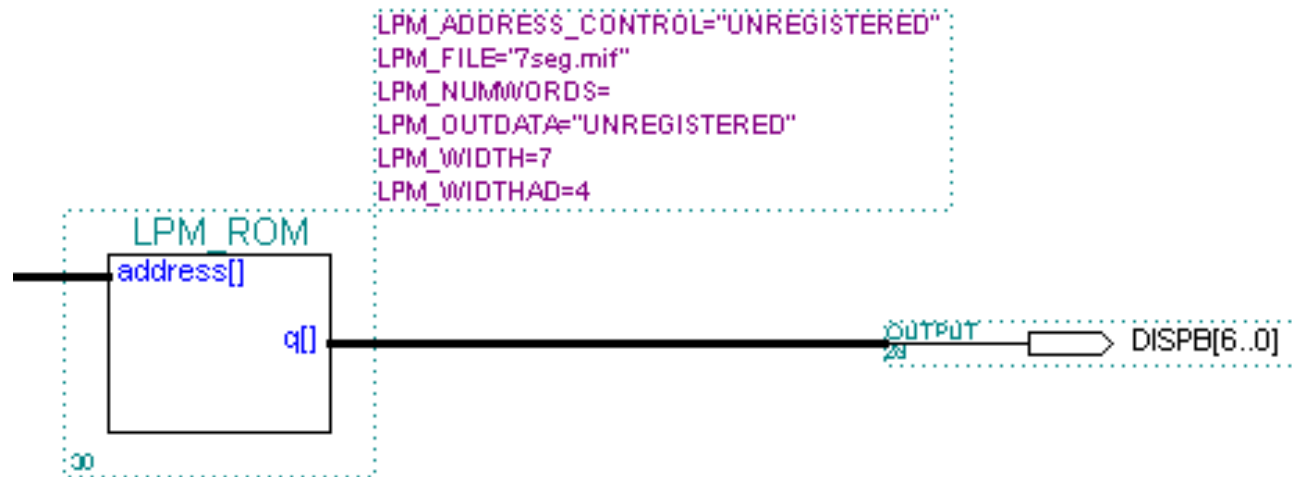
```
WIDTH = 7;  
DEPTH = 16;
```

- LPM\_FILE - 7seg.mif

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

```
CONTENT BEGIN
```

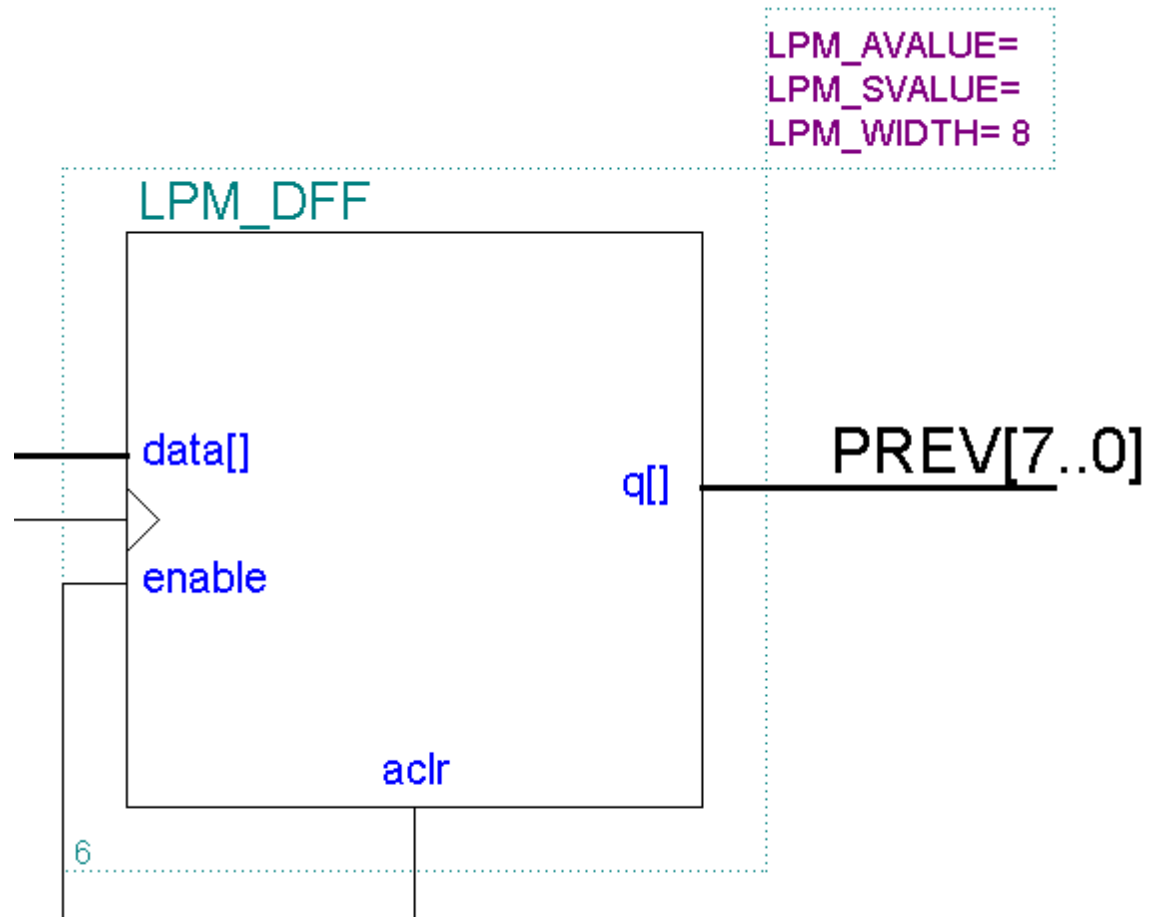
```
0: 1000000;  
1: 1111001;  
2: 0100100;  
3: 0110000;  
4: 0011001;  
5: 0010010;  
6: 0000010;  
7: 1011000;  
8: 0000000;  
9: 0010000;  
A: 0001000;  
B: 0000011;  
C: 1000110;  
D: 0100001;  
E: 0000110;  
F: 0001110;  
END;
```





# Lab 4 - fib\_top IV

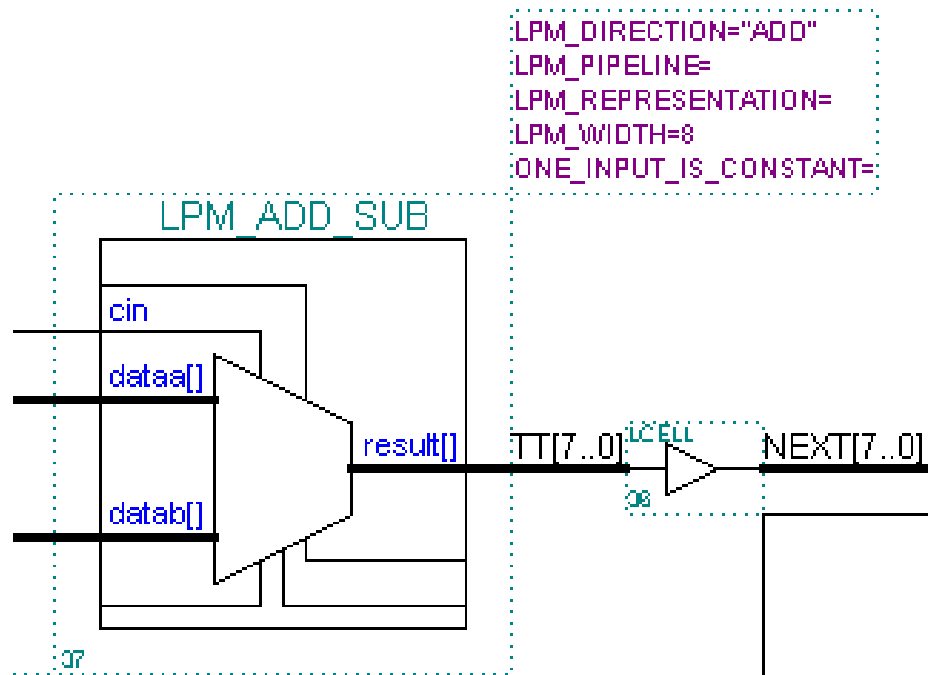
## ◆ LPM\_REG





# Lab 4 - fib\_top V

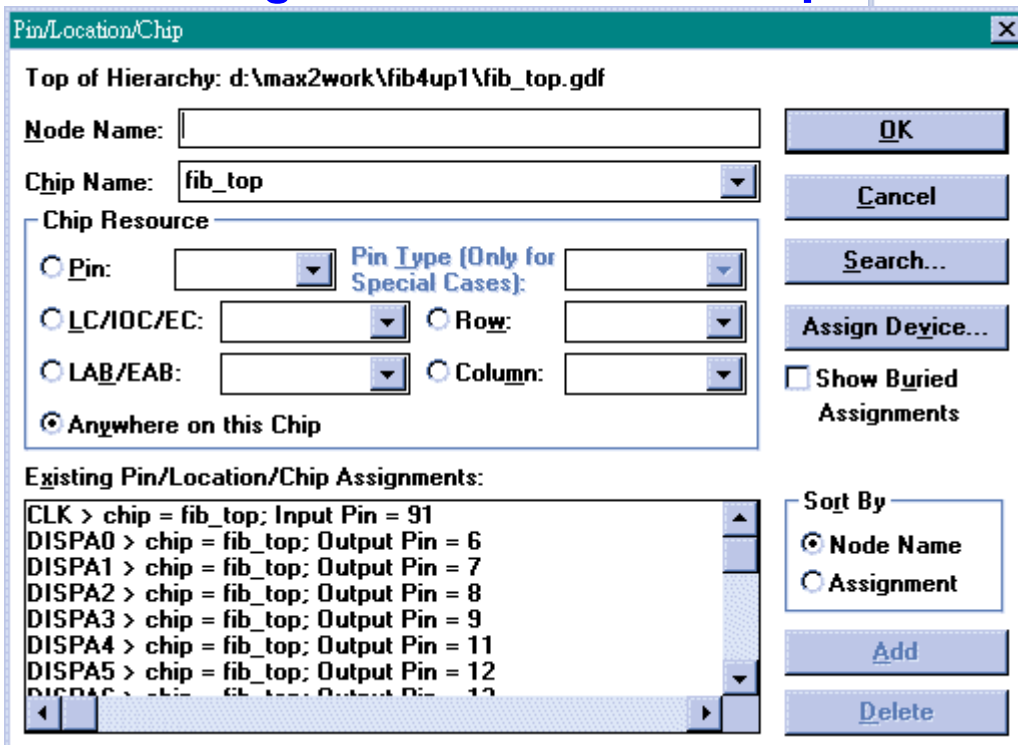
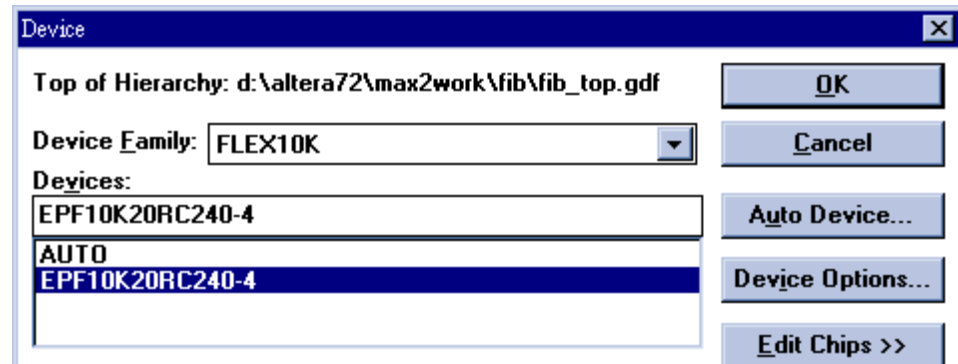
## ◆ LPM\_ADD\_SUB





# Lab 5 - Compile I

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



Fastest Speed Grades

Current Synthesis Regardless of Device or Speed Grade Changes

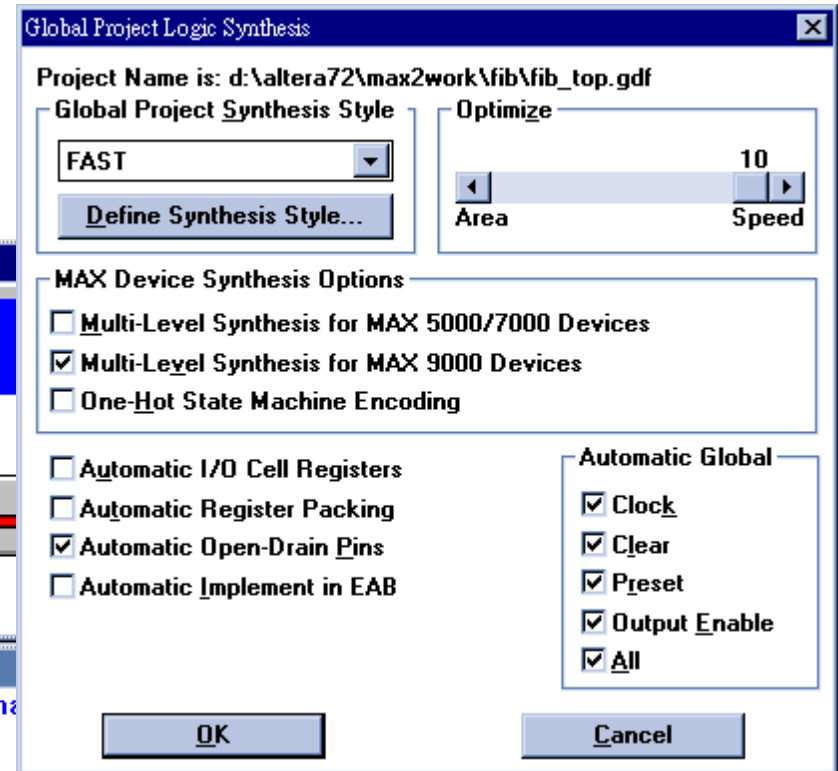
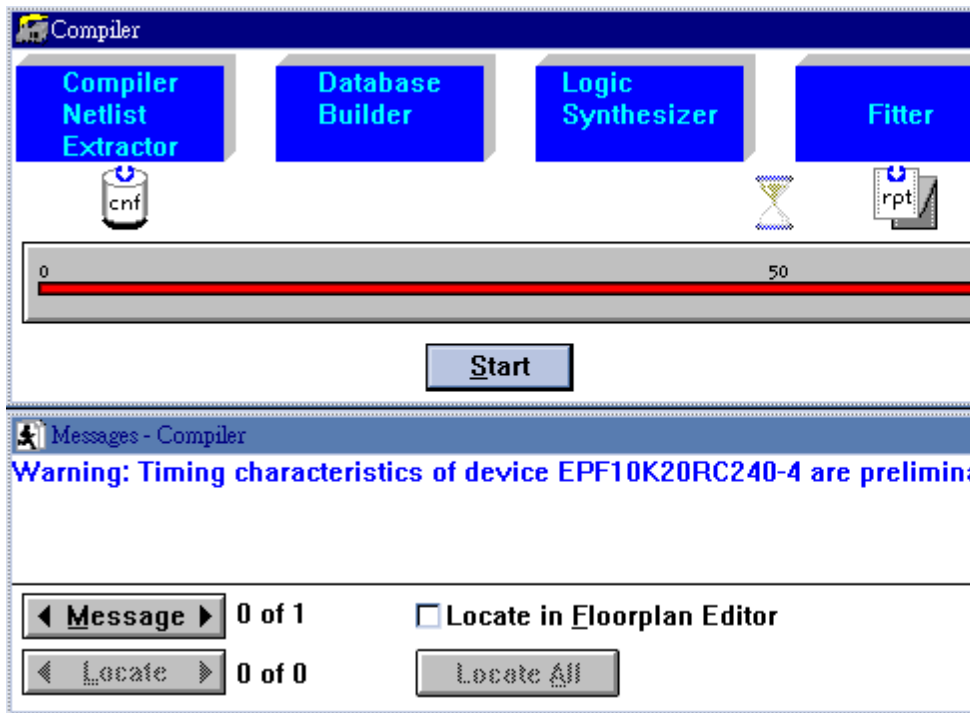


# Lab 5 - Compile II

## ◆ 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

[illegible]



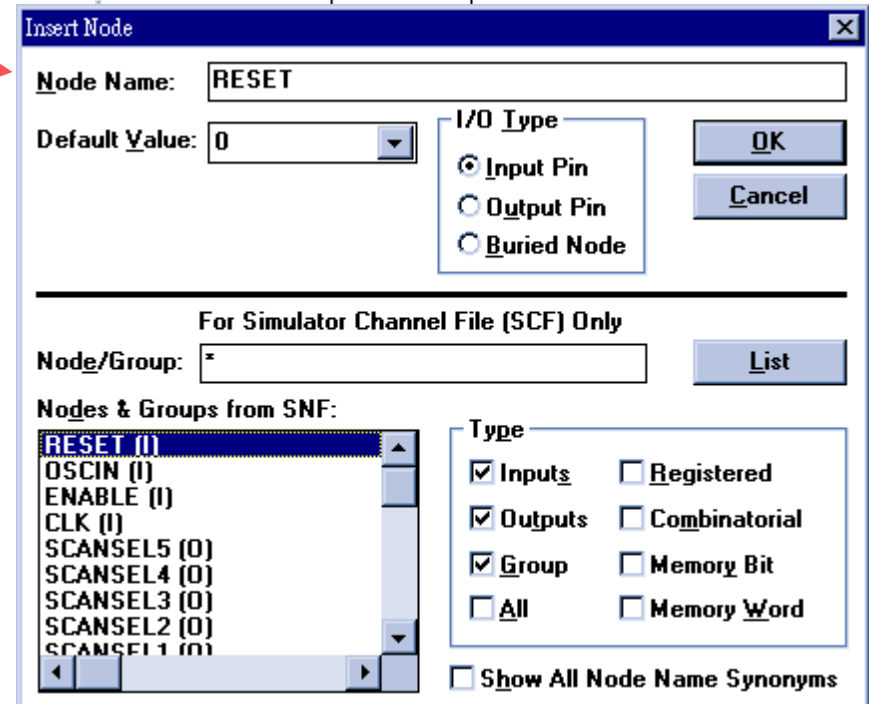
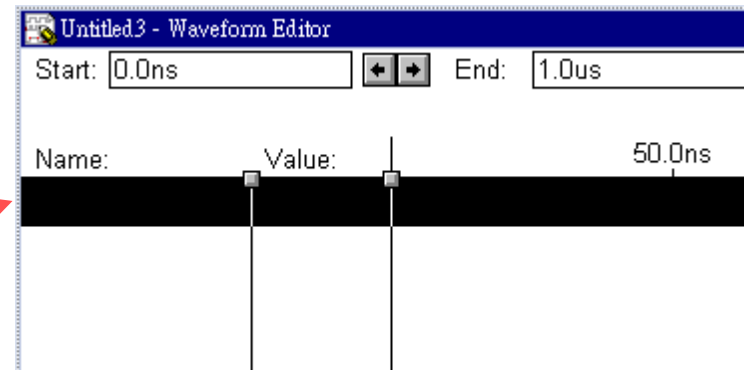
# Lab 8 - Timing Simulation I

## ◆ Create a SCF file

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

## ◆ Insert nodes in SCF

- Double click on "name"
- Fill Node Name
  - TESTCLK, SEL, RESET, ENABLE, PREV[7..0], FIB[7..0], NEXT[7..0], DISPA[6..0], DISPB[6..0]





# Lab 8 - Timing Simulation II

## ◆ Change Grid Size

- Options > Grid Size (20.0ns)

## ◆ Set End Time

- File > End Time (5us)

## ◆ Draw Waveforms

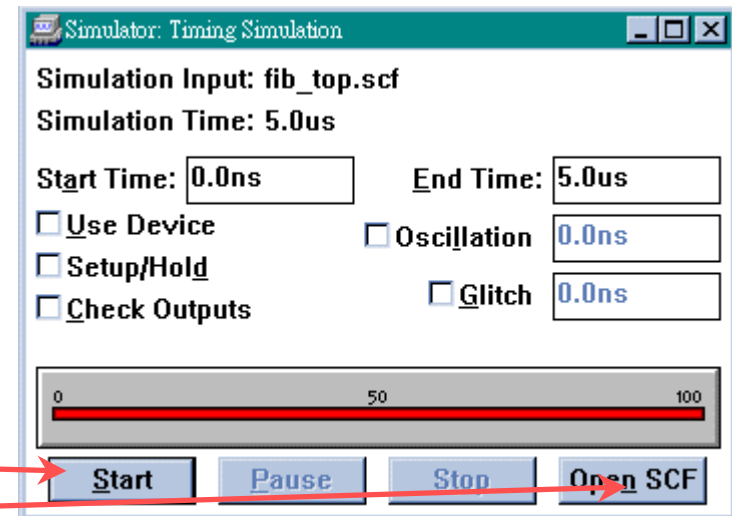
- TESTCLK (clock, period 40ns)
- SEL ( 0)
- RESET (0ns 0, 120ns 1, 1.0us 0, 1.12ns 1)
- ENABLE (0ns 1, 1.6us 0, 2.0us 1)

## ◆ Save SCF File

- File > Save as (fib\_top.scf)

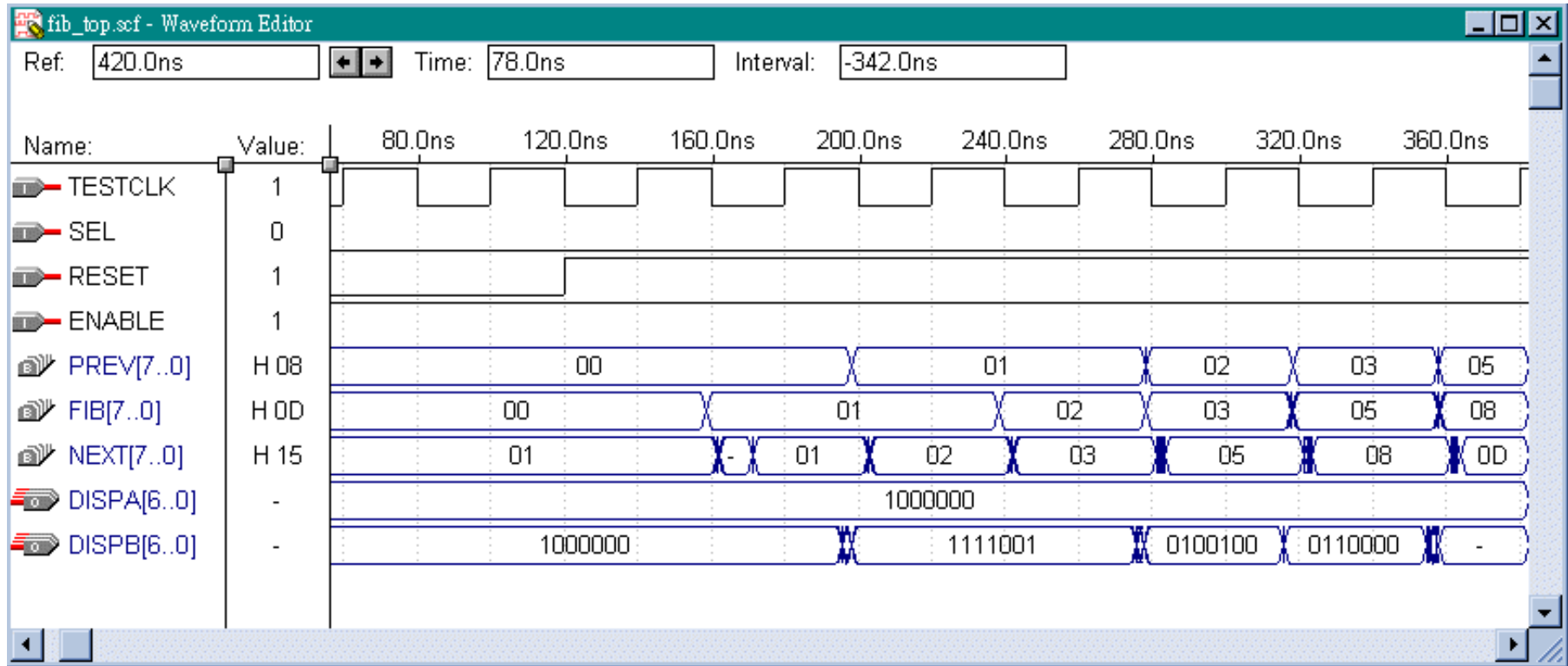
## ◆ Run Simulator

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





# Lab 8 - Timing Simulation III



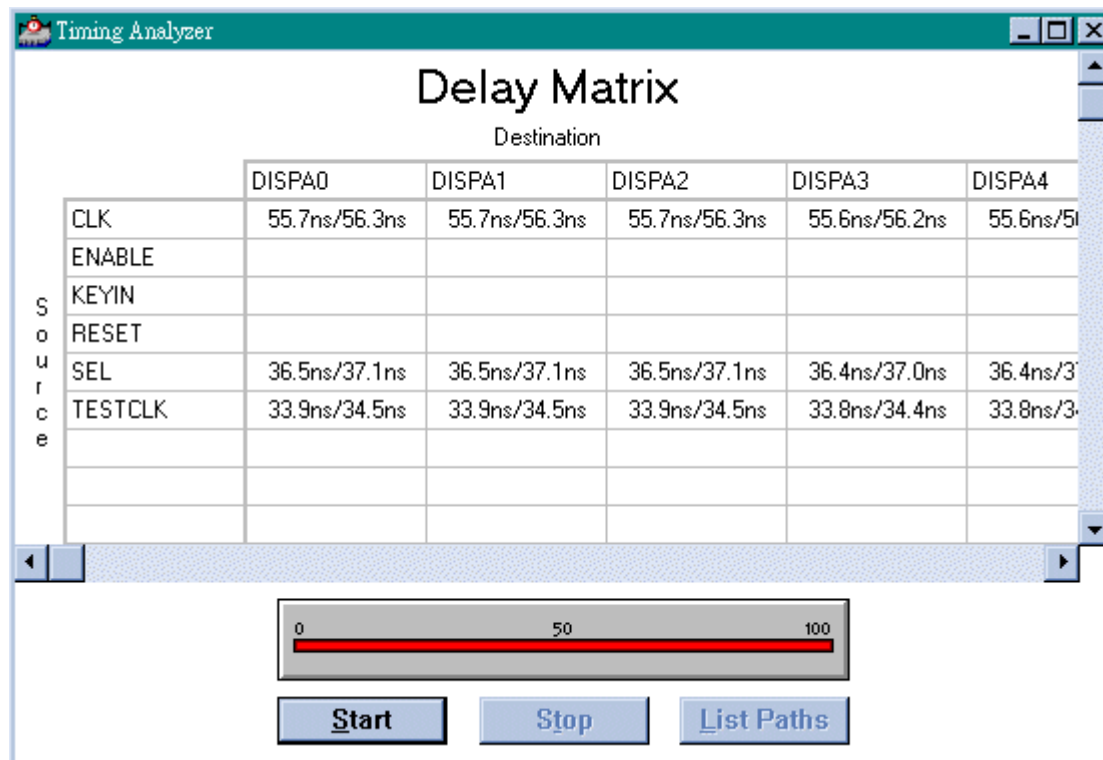


# Lab 9 - Timing Analysis I

◆ MAX+plus II > Timing Analyzer

◆ Analysis > Delay Matrix

- Press “Start”





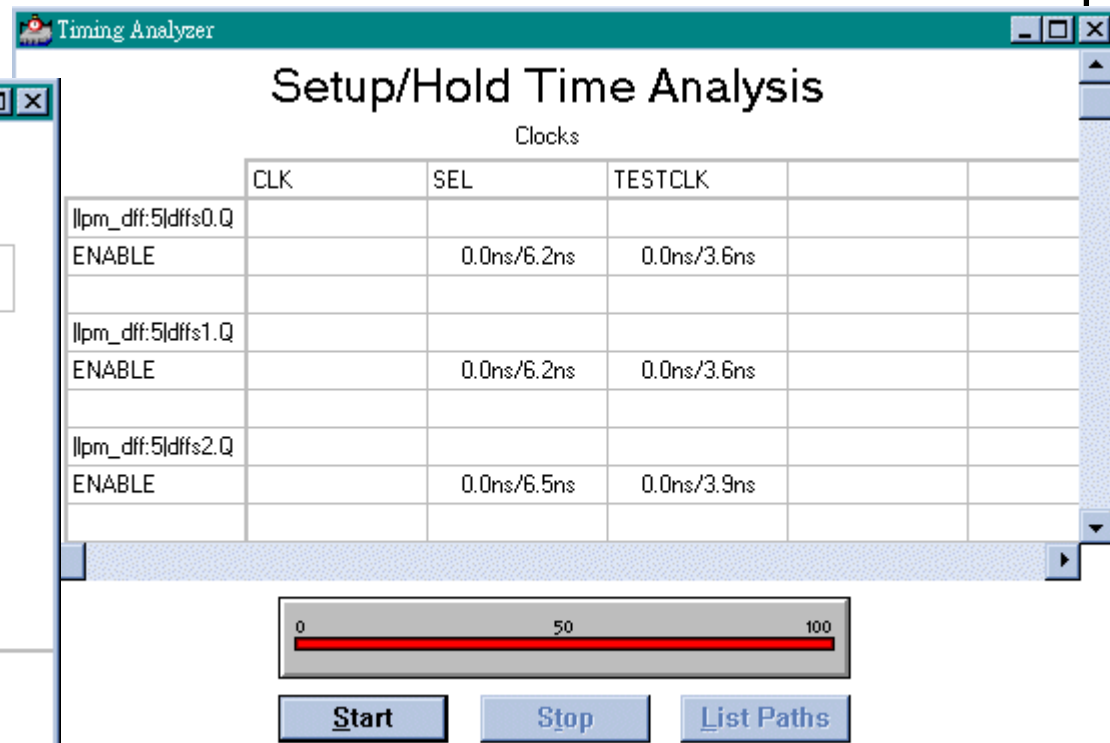
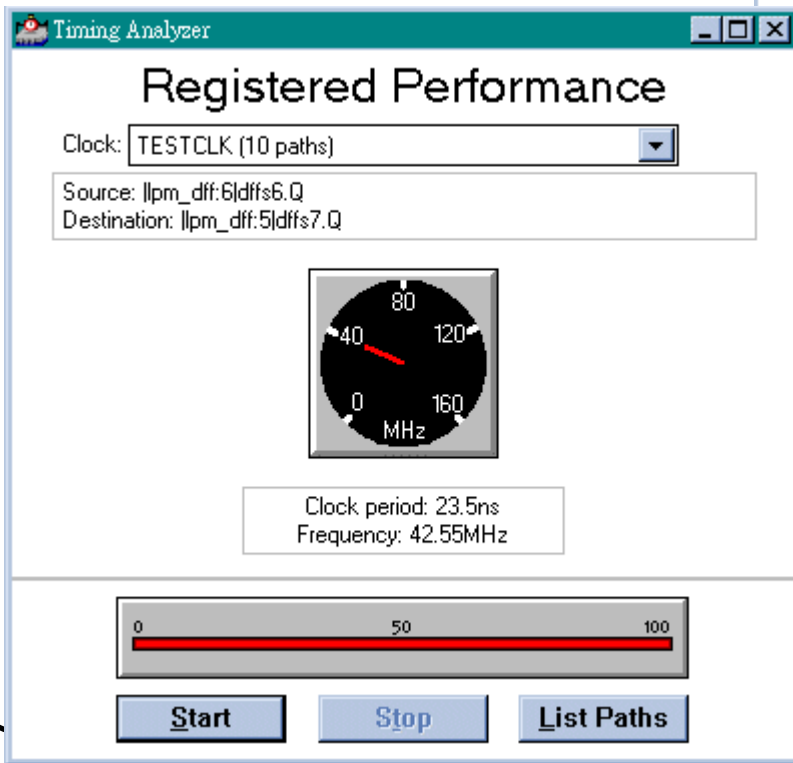
# Lab 9 - Timing Analysis II

## ◆ Analysis > Setup/Hold Matrix

- Press “Start”

## ◆ Analysis > Registered Performance

- Press “Start”





# Lab 10 - Programmer I

## ◆ 連接硬體

- 適當連接 Byteblaster, UP1, Adapter, Parallel Port

## ◆ 開啓 Programmer

- MAX+PLUS II > Programmer

## ◆ 設定 Byteblaster

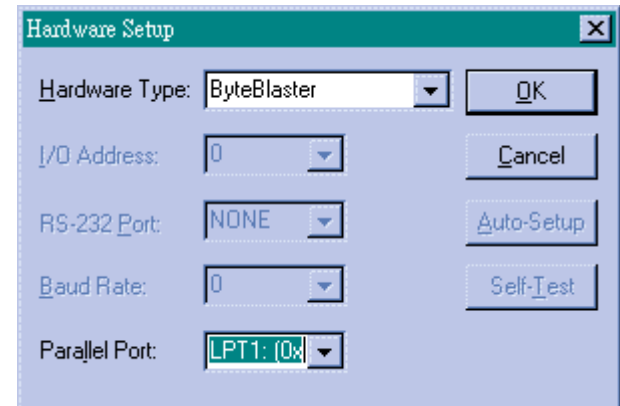
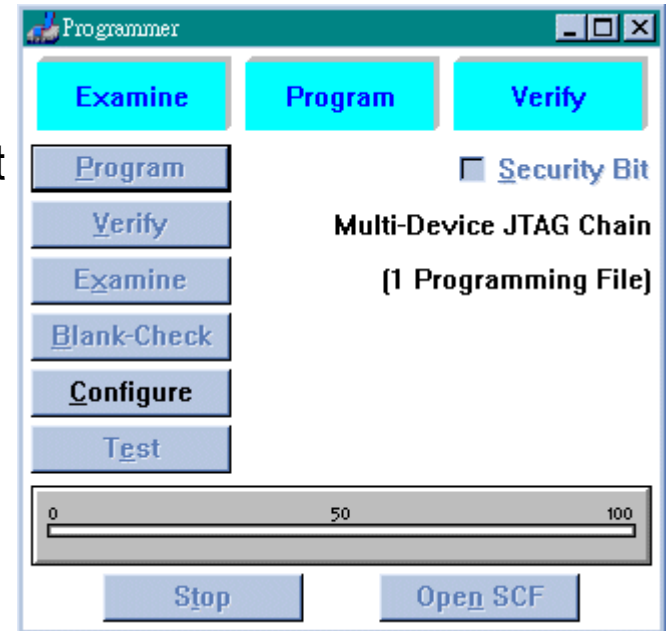
- Options > Hardware Setup

## ◆ 設定 Multi-Device JTAG Chain

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

## ◆ 設定 Multi-Device JTAG Chain Setup

- JTAG > Multi-Device JTAG Chain Setup...
- Device Name - 10K20
- Programming File Name - fib\_top.sof
  - 記得要按 Add





# Lab 10 - Programmer II

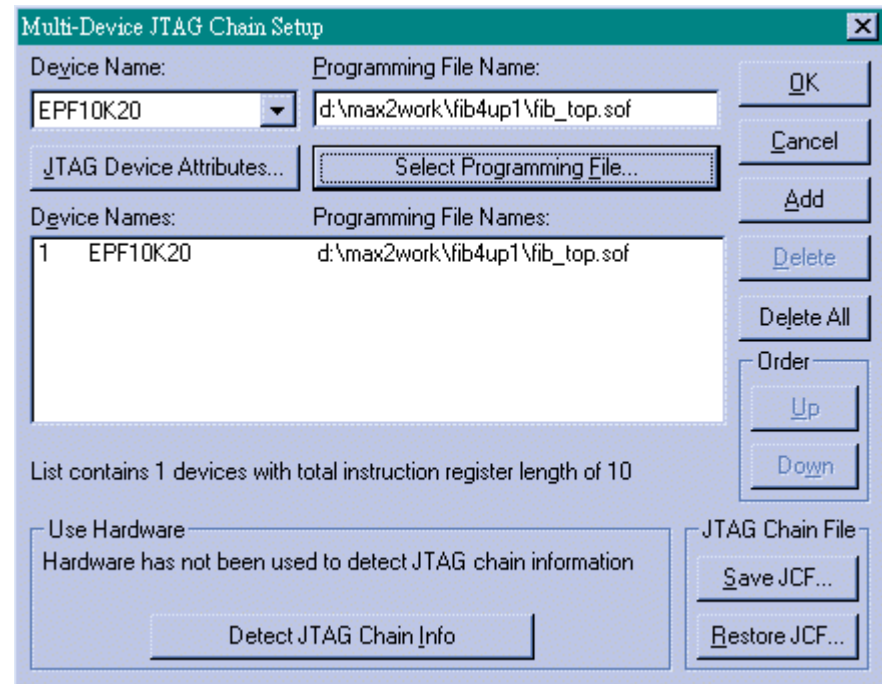
- Click “Detect JTAG Chain Info”, 如果沒有問題, 將出現下面訊息



## ◆下載電路

- Click “Configure” in Programmer Window

## ◆操作電路





# Pin Assignment

**SEL : PIN = 39;**

**CLK : INPUT\_PIN = 91;**

**KEYIN : PIN = 28;**

**DISPB6 : OUTPUT\_PIN = 24;**

**DISPB5 : OUTPUT\_PIN = 23;**

**DISPB4 : OUTPUT\_PIN = 21;**

**DISPB2 : OUTPUT\_PIN = 19;**

**DISPB3 : OUTPUT\_PIN = 20;**

**DISPB1 : OUTPUT\_PIN = 18;**

**DISPB0 : OUTPUT\_PIN = 17;**

**DISPA6 : OUTPUT\_PIN = 13;**

**DISPA5 : OUTPUT\_PIN = 12;**

**DISPA4 : OUTPUT\_PIN = 11;**

**DISPA3 : OUTPUT\_PIN = 9;**

**DISPA2 : OUTPUT\_PIN = 8;**

**DISPA1 : OUTPUT\_PIN = 7;**

**DISPA0 : OUTPUT\_PIN = 6;**

**ENABLE : INPUT\_PIN = 40;**

**RESET : INPUT\_PIN = 41;**