FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

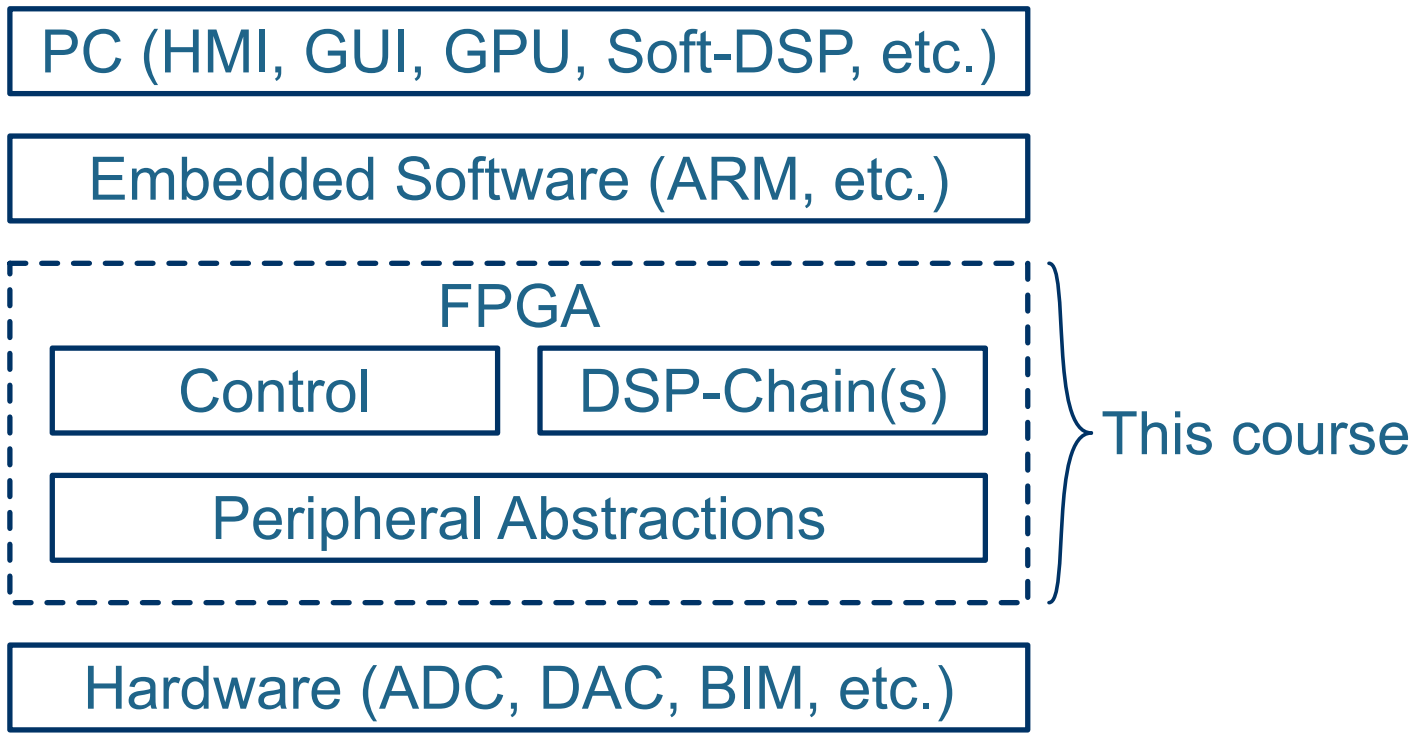
Convened by Dr Stephen Paine

Day 1 – 27 April 2022

Outline

Outline

Course Overview



Course Overview

- Overall (Morning)
  - This course specifically targets low-level development
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL (the presenter is well-versed in both)
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL (the presenter is well-versed in both)
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- Never be shy to ask questions

Course Overview

- Overall (Morning)
  - This course specifically targets low-level development (when the high-level libraries don't provide the required functionality)
  - No high-level synthesis (HLS)
  - No soft-core or embedded processors / etc.
- Practicals (Afternoon)
  - The practicals are challenging to finish in the time provided, so keep the board and work on it after the course...
  - Primarily Verilog, but feel free to implement your practicals in VHDL (the presenter is well-versed in both)
  - These are informal – feel free to structure your time as you see fit, or even do the practicals at home
- **Never be shy to ask questions**

Course Overview

- EEE5117Z (Days 1 to 3)
  - Understand the underlying physical architecture of FPGAs
  - Understand the concept of timing constraints, clock domains and other timing-related issues
  - Use the Altera tool-set, including Qsys, JTAG debugging and the general Verilog-based compilation process
- EEE5118Z (Days 4 to 6)
  - Design FPGA firmware systems on a high level
  - Design FPGA firmware blocks on a low level (i.e. RTL representations of finite state machines and pipelines)
  - Implementing FPGA firmware systems
  - Debug an FPGA firmware implementation
  - Analise timing closure issues and solve the problem such that the final design meets all timing requirements.

Course Overview

- EEE5117Z (Days 1 to 3)
  - Understand the underlying physical architecture of FPGAs
  - Understand the concept of timing constraints, clock domains and other timing-related issues
  - Use the Altera tool-set, including Qsys, JTAG debugging and the general Verilog-based compilation process
- EEE5118Z (Days 4 to 6)
  - Design FPGA firmware systems on a high level
  - Design FPGA firmware blocks on a low level (i.e. RTL representations of finite state machines and pipelines)
  - Implementing FPGA firmware systems
  - Debug an FPGA firmware implementation
  - Analise timing closure issues and solve the problem such that the final design meets all timing requirements.

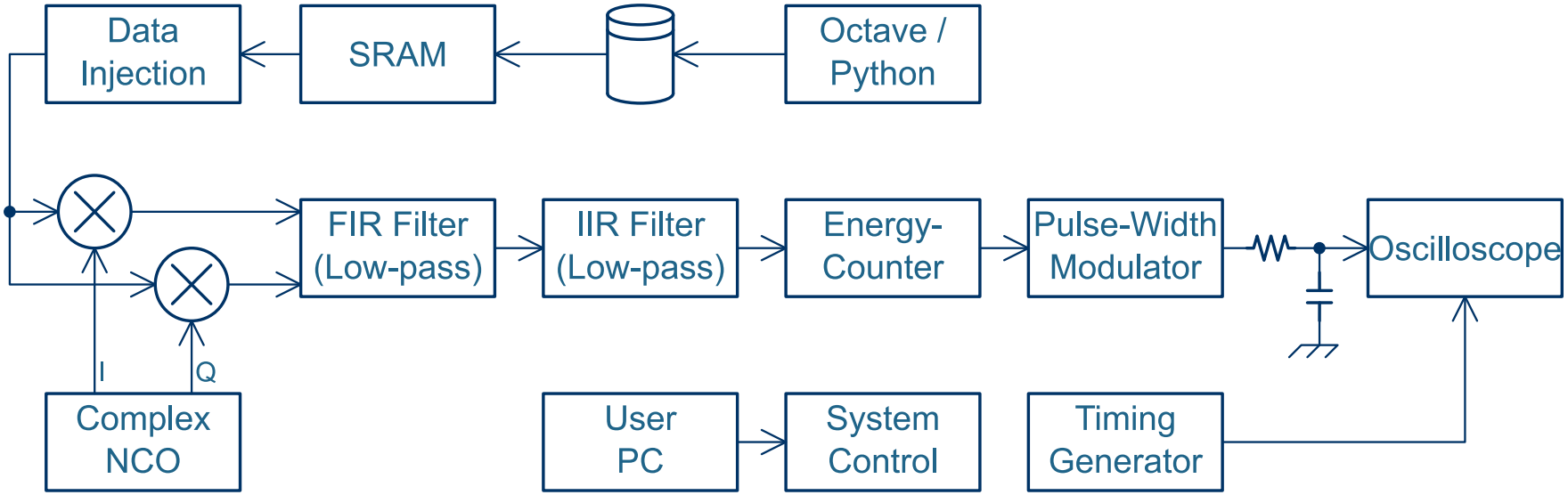
Course Overview

- EEE5117Z (Days 1 to 3)
  - Understand the underlying physical architecture of FPGAs
  - Understand the concept of timing constraints, clock domains and other timing-related issues
  - Use the Altera tool-set, including Qsys, JTAG debugging and the general Verilog-based compilation process
- EEE5118Z (Days 4 to 6)
  - Design FPGA firmware systems on a high level
  - Design FPGA firmware blocks on a low level (i.e. RTL representations of finite state machines and pipelines)
  - Implementing FPGA firmware systems
  - Debug an FPGA firmware implementation
  - Analise timing closure issues and solve the problem such that the final design meets all timing requirements.

Course Overview

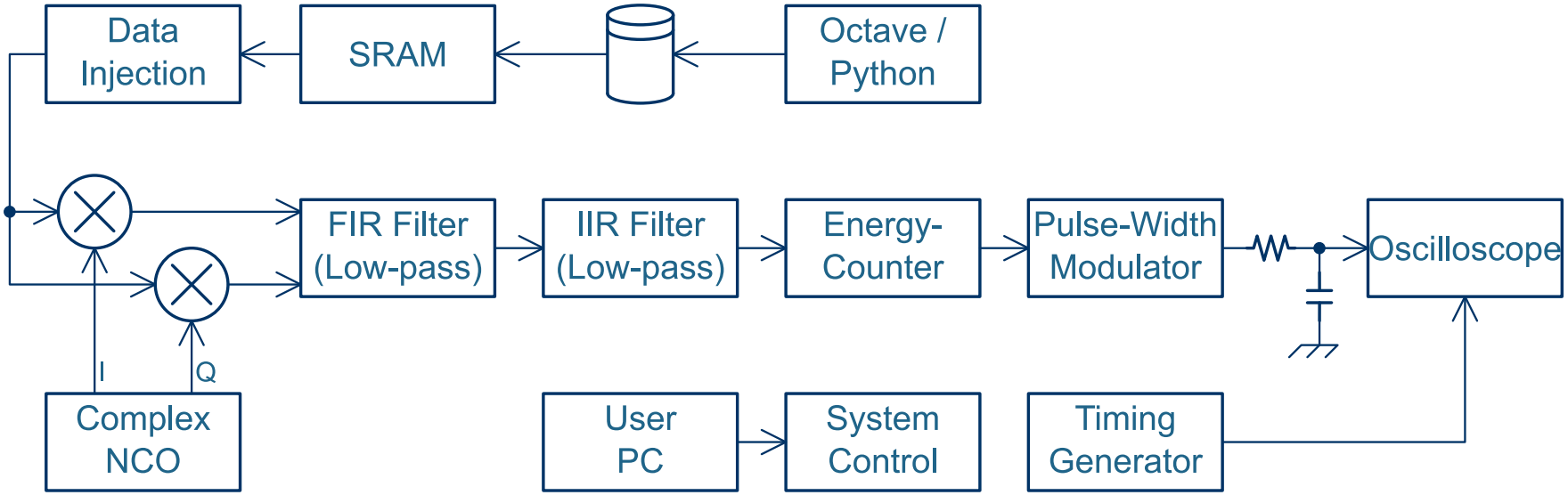
- EEE5117Z (Days 1 to 3)
  - Understand the underlying physical architecture of FPGAs
  - Understand the concept of timing constraints, clock domains and other timing-related issues
  - Use the Altera tool-set, including Qsys, JTAG debugging and the general Verilog-based compilation process
- EEE5118Z (Days 4 to 6)
  - Design FPGA firmware systems on a high level
  - Design FPGA firmware blocks on a low level (i.e. RTL representations of finite state machines and pipelines)
  - Implementing FPGA firmware systems
  - Debug an FPGA firmware implementation
  - Analise timing closure issues and solve the problem such that the final design meets all timing requirements.

Practicals



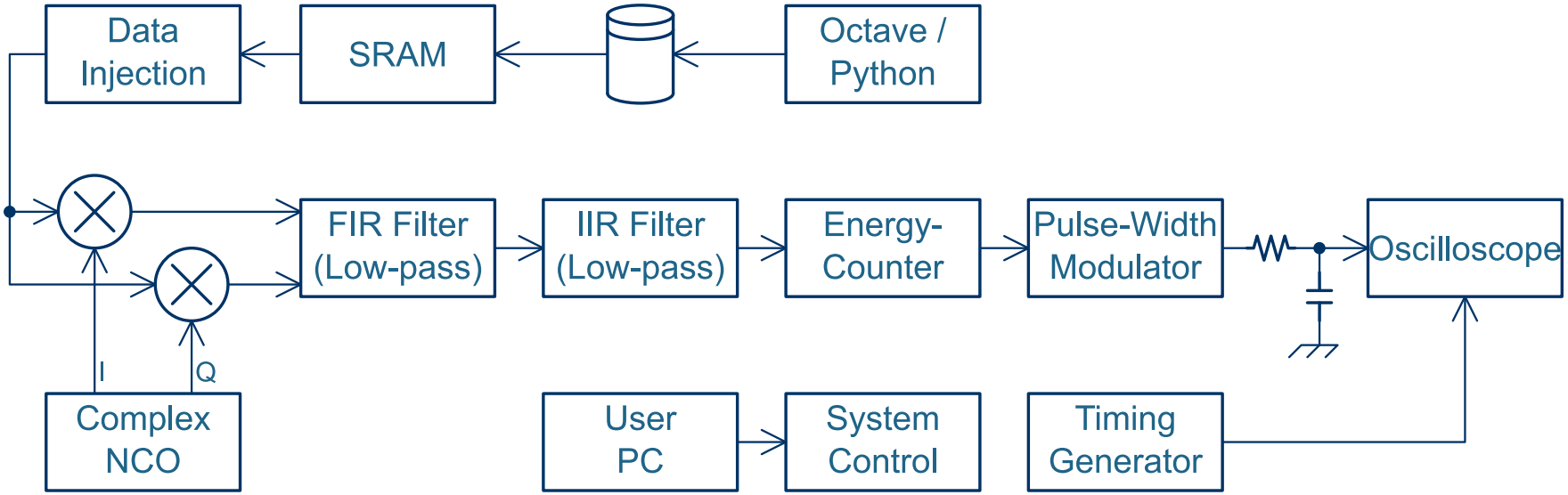
- JTAG interfacing to / from the computer (Source-and-Probes and TCL scripting)
- 8-bit, 100 MSps data injection
- 1024-point FIR-filter with 128× decimation

Practicals



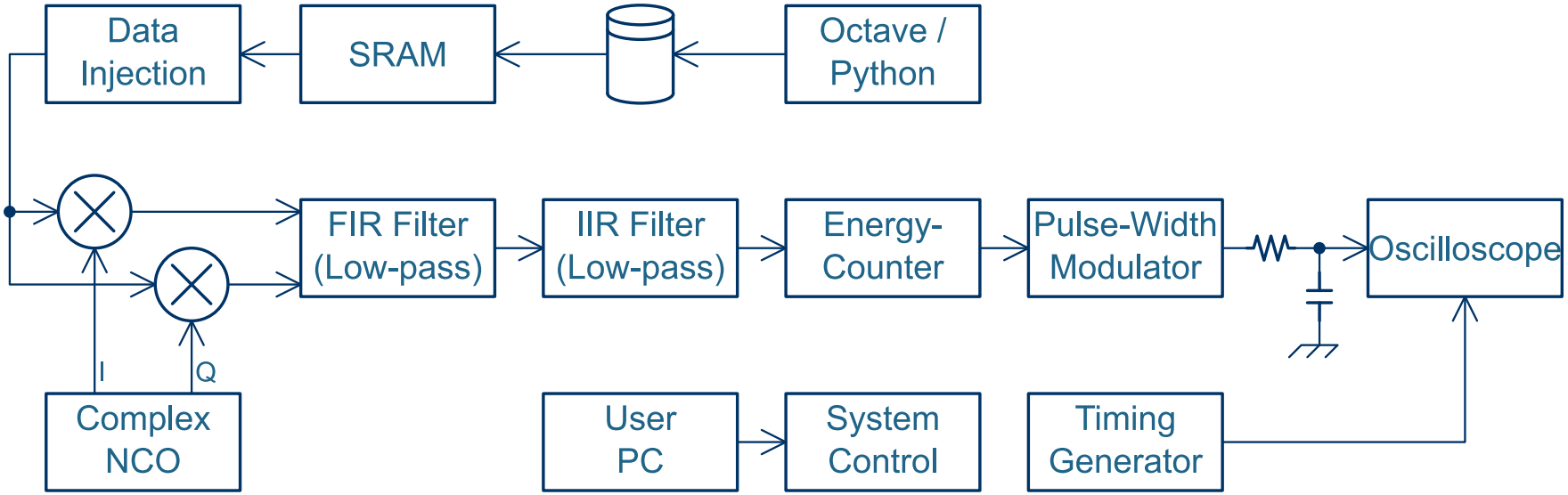
- JTAG interfacing to / from the computer (Source-and-Probes and TCL scripting)
- 8-bit, 100 MSps data injection
- 1024-point FIR-filter with 128× decimation

Practicals



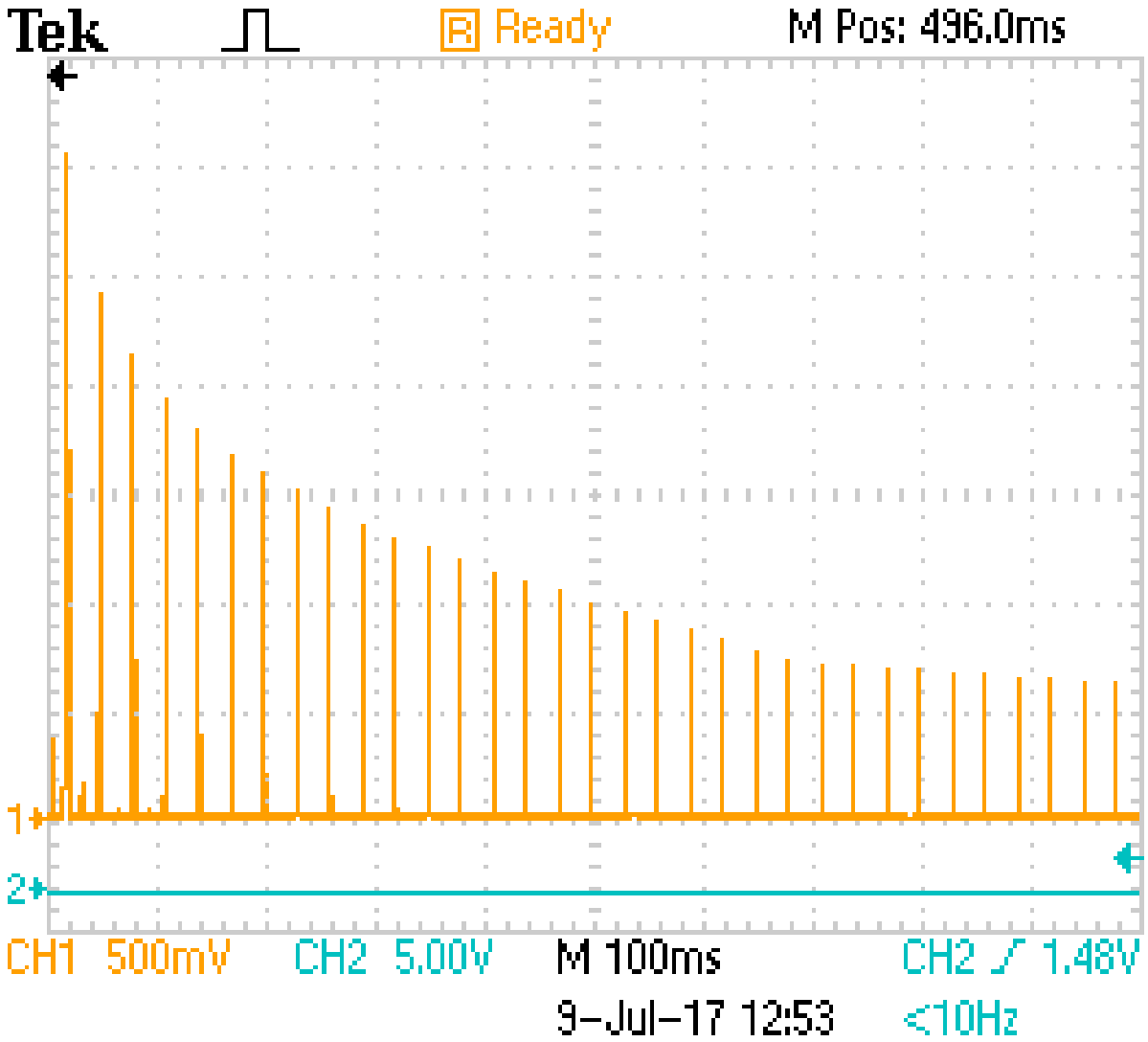
- JTAG interfacing to / from the computer (Source-and-Probes and TCL scripting)
- 8-bit, 100 MSps data injection
- 1024-point FIR-filter with 128× decimation

Practicals



- JTAG interfacing to / from the computer (Source-and-Probes and TCL scripting)
- 8-bit, 100 MSps data injection
- 1024-point FIR-filter with 128× decimation

Practicals


Tek
Ready
M Pos: 496.0ms
1
2
CH1 500mV
CH2 5.00V
M 100ms
CH2 ⨍ 1.48V
9-Jul-17 12:53
<10Hz

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

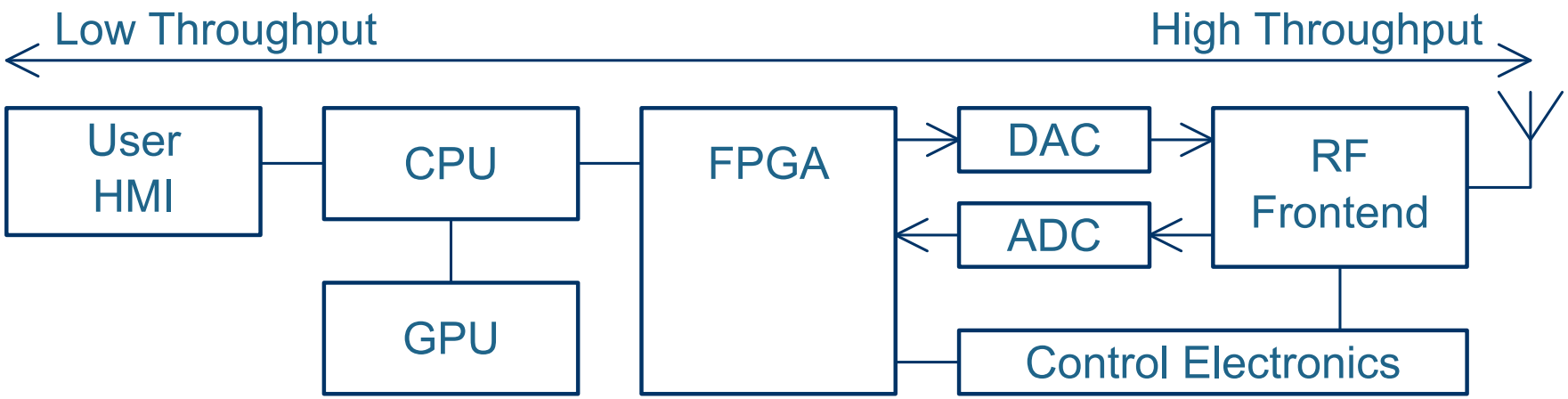
The Audience

- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

The Audience

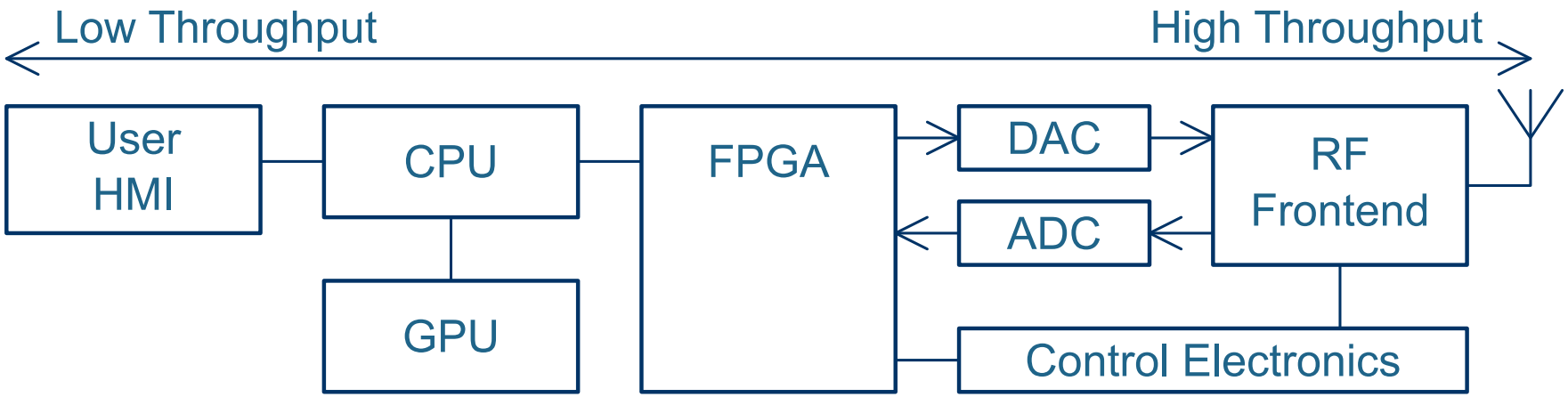
- Programming experience?
  - C / C++ / C#?
  - Embedded? PC?
  - GPU? OpenGL? OpenCL? DirectX? CUDA?
- FPGA Experience?
  - VHDL? Verilog? PyHDL? Migen? Graphical tools?
  - Synthesis? Simulation only?
  - What context / application?

Software-Defined Radio



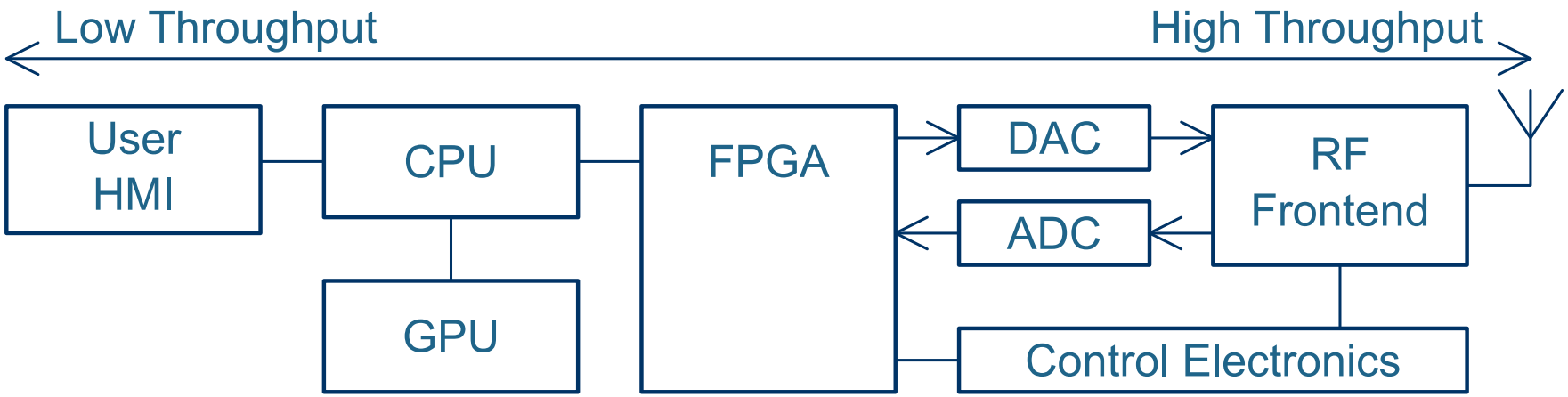
- In high-performance computing, the CPU, GPU and FPGA work together
- Each processing element is used for its strength

Software-Defined Radio



- In high-performance computing, the CPU, GPU and FPGA work together
- Each processing element is used for its strength

Software-Defined Radio



- In high-performance computing, the CPU, GPU and FPGA work together
- Each processing element is used for its strength

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time (kernels are referenced by a handle, or memory address)
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms (little to none inter-worker communication: matrix multiplication; FFT; FIR filters; etc.)
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle – debugging and optimising is a challenging process
  - Memory bandwidth problems
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems (the bottle-neck is generally in data transfer, but less so in modern GPUs with advanced parallel copy-engines)
- FPGA:

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:
  - Highly flexible, but with a relatively slow clock
  - Provides ASIC functionality at a small fraction of the cost
  - Excellent at fine-grained and time-critical problems
  - Long development cycle; challenging to debug and optimise
  - Not agile: takes a few seconds to switch functionality

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:
  - Highly flexible, but with a relatively slow clock
  - Provides ASIC functionality at a small fraction of the cost
  - Excellent at fine-grained and time-critical problems
  - Long development cycle; challenging to debug and optimise
  - Not agile: takes a few seconds to switch functionality

Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:
  - Highly flexible, but with a relatively slow clock
  - Provides ASIC functionality at a small fraction of the cost
  - Excellent at fine-grained and time-critical problems
  - Long development cycle; challenging to debug and optimise
  - Not agile: takes a few seconds to switch functionality

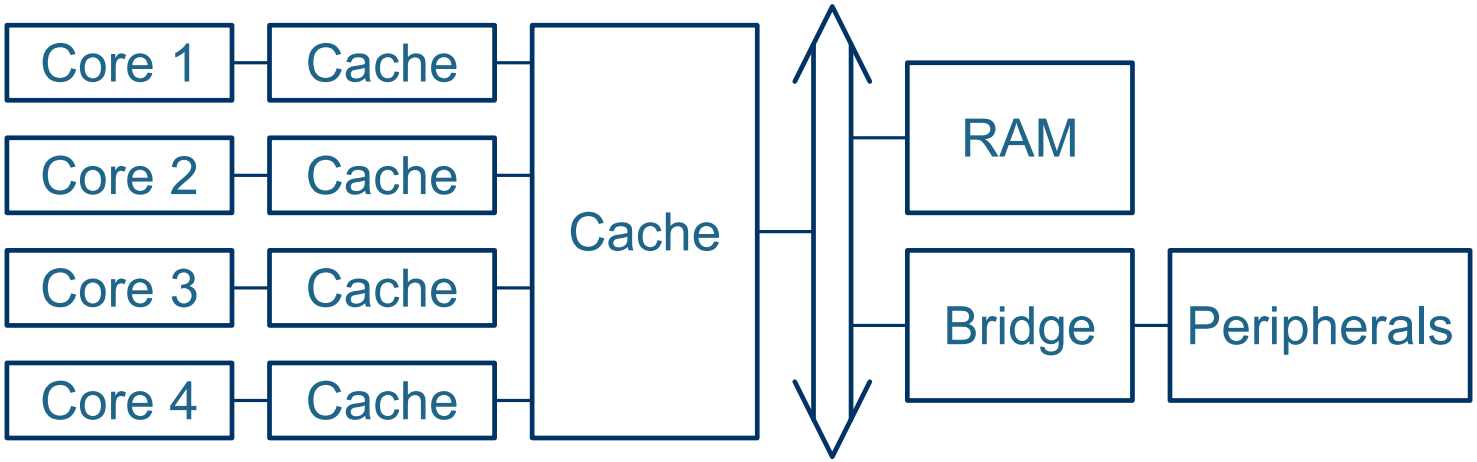
Processing Platforms

- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:
  - Highly flexible, but with a relatively slow clock
  - Provides ASIC functionality at a small fraction of the cost
  - Excellent at fine-grained and time-critical problems
  - Long development cycle; challenging to debug and optimise
  - Not agile: takes a few seconds to switch functionality

Processing Platforms

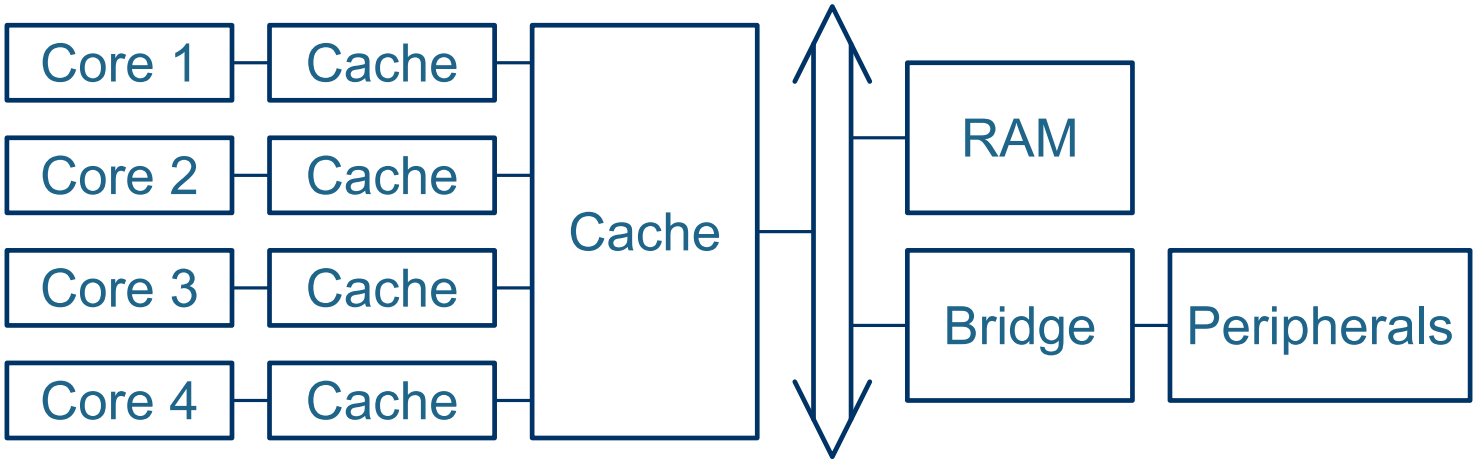
- CPU:
  - General-purpose; Easily modified; Short development cycle
  - Extensive history $\Rightarrow$ Rich set of matured libraries and APIs
  - Does not handle parallel algorithms particularly well
- GPU:
  - Agile: can be reprogrammed in real-time
  - Extremely good at course-grained parallel algorithms
  - Medium development cycle
  - Memory bandwidth problems
- FPGA:
  - Highly flexible, but with a relatively slow clock
  - Provides ASIC functionality at a small fraction of the cost
  - Excellent at fine-grained and time-critical problems
  - Long development cycle; challenging to debug and optimise
  - Not agile: takes a few seconds to switch functionality

Programming Model – CPU



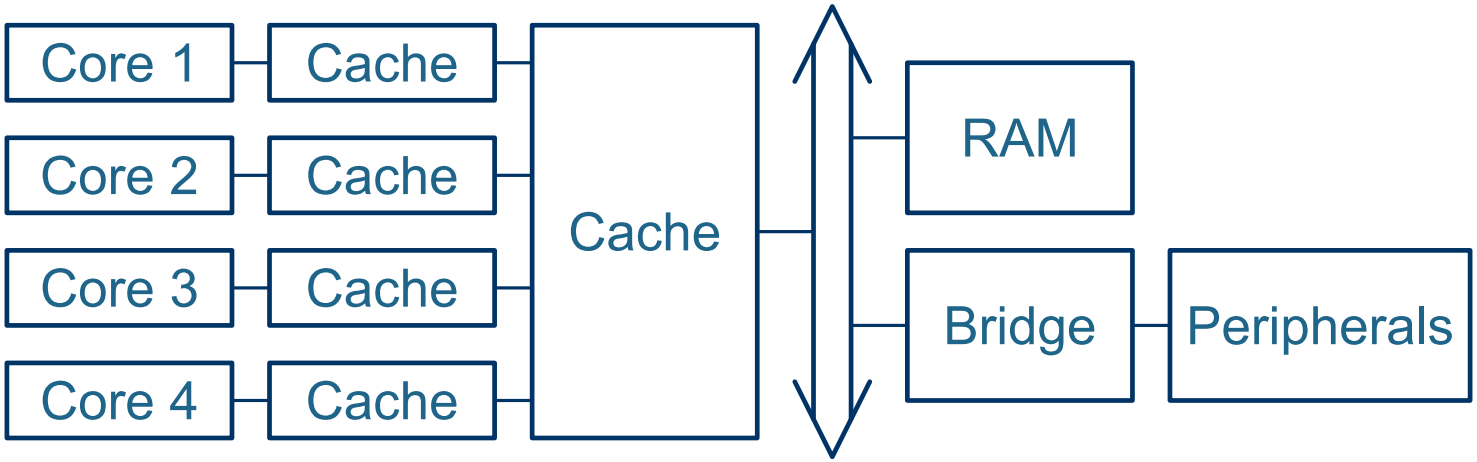
- Fetch-decode-execute cycle – serialised execution
- The RAM is divided into program, stack and heap areas
- All CPU cores share the same RAM, but is cache-assisted to reduce contention

Programming Model – CPU



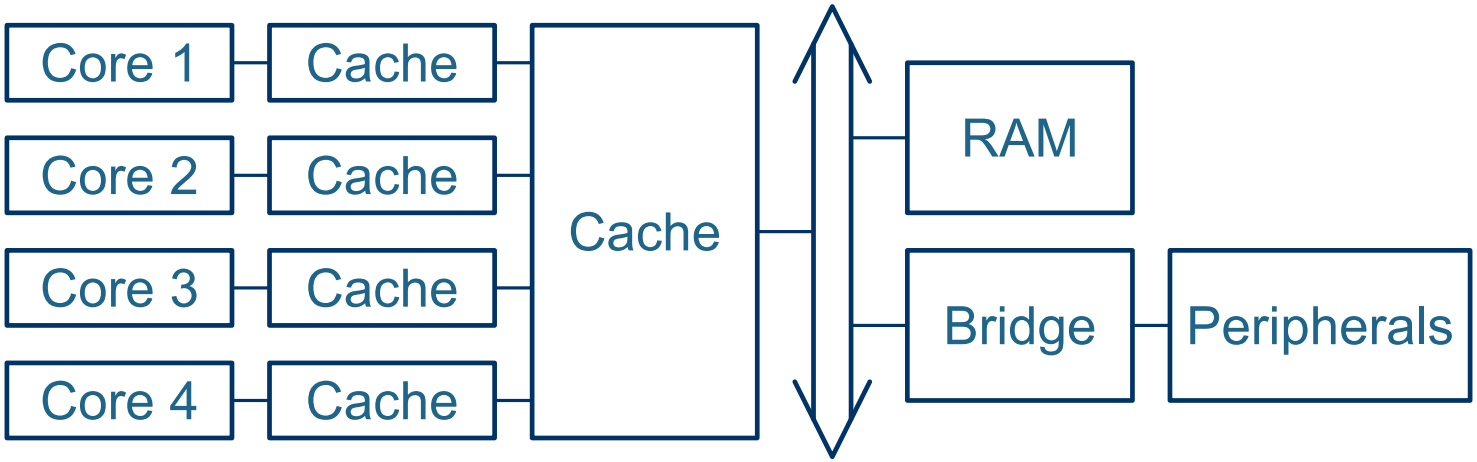
- Fetch-decode-execute cycle – serialised execution
- The RAM is divided into program, stack and heap areas
- All CPU cores share the same RAM, but is cache-assisted to reduce contention

Programming Model – CPU



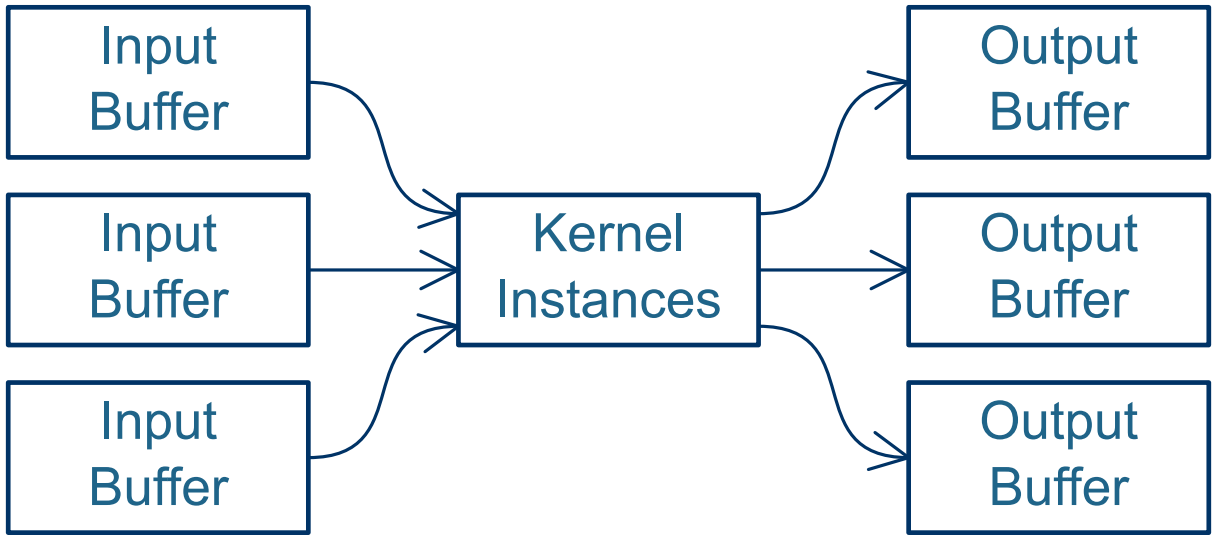
- Fetch-decode-execute cycle – serialised execution
- The RAM is divided into program, stack and heap areas
- All CPU cores share the same RAM, but is cache-assisted to reduce contention

Programming Model – CPU



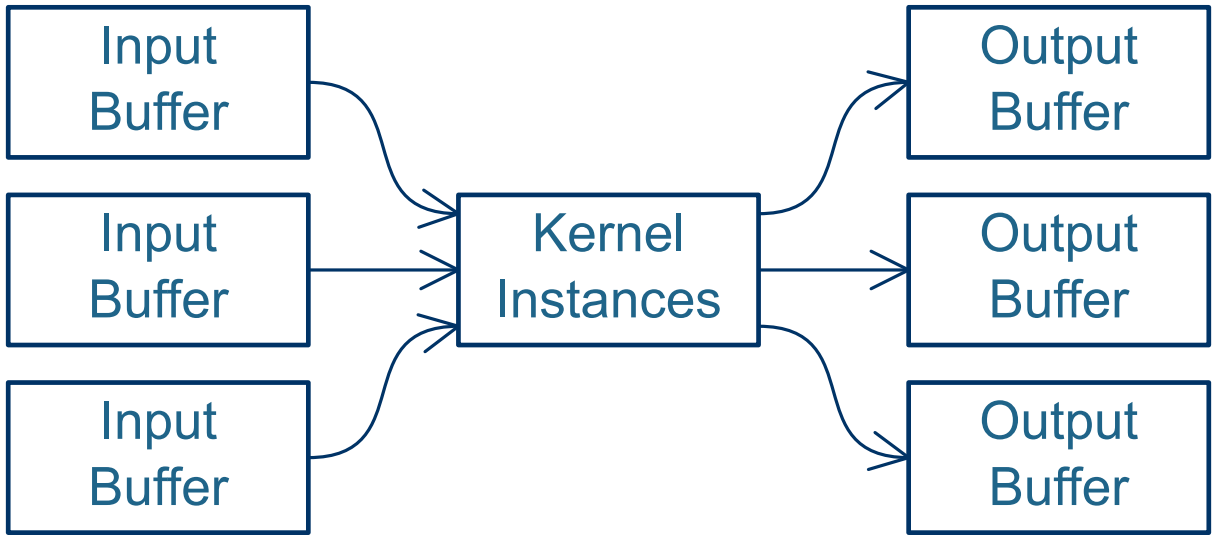
- Fetch-decode-execute cycle – serialised execution
- The RAM is divided into program, stack and heap areas
- All CPU cores share the same RAM, but is cache-assisted to reduce contention

Programming Model – GPU



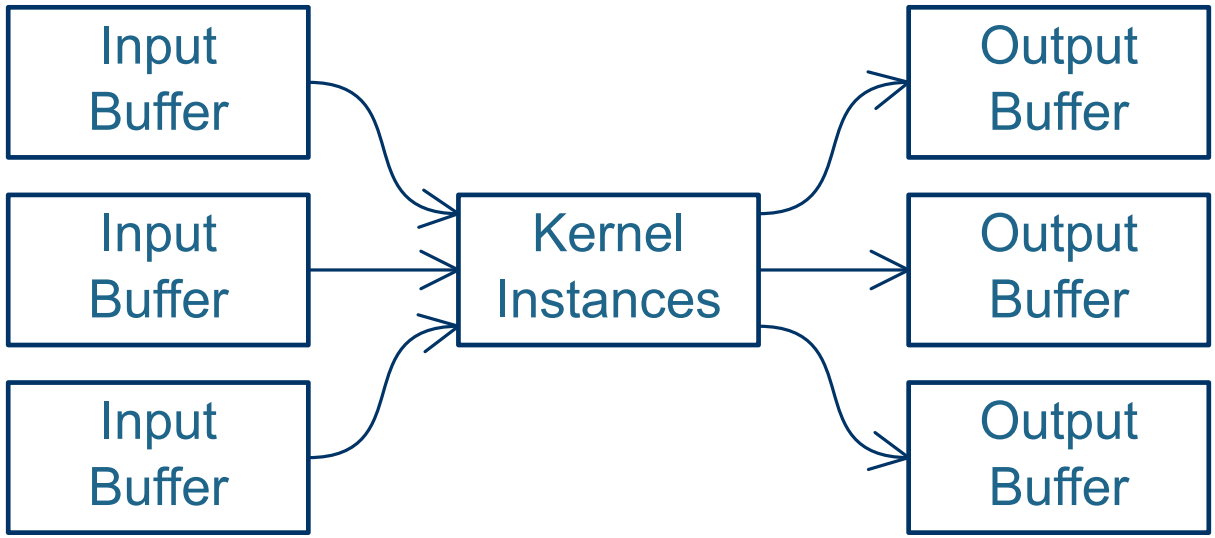
- Client-server interface with the CPU
- Runs a pipeline of kernels, in SIMD operation

Programming Model – GPU



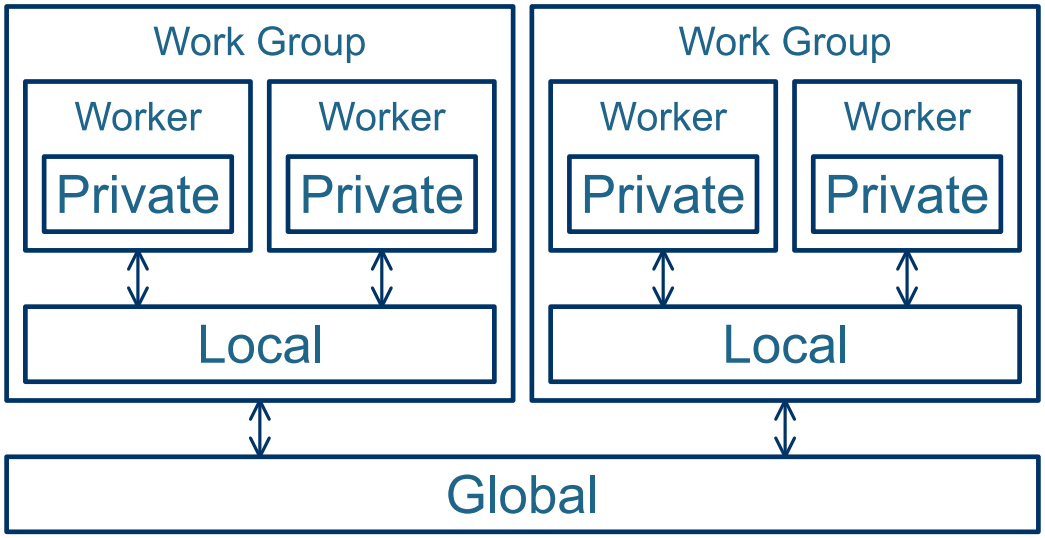
- Client-server interface with the CPU
- Runs a pipeline of kernels, in SIMD operation

Programming Model – GPU



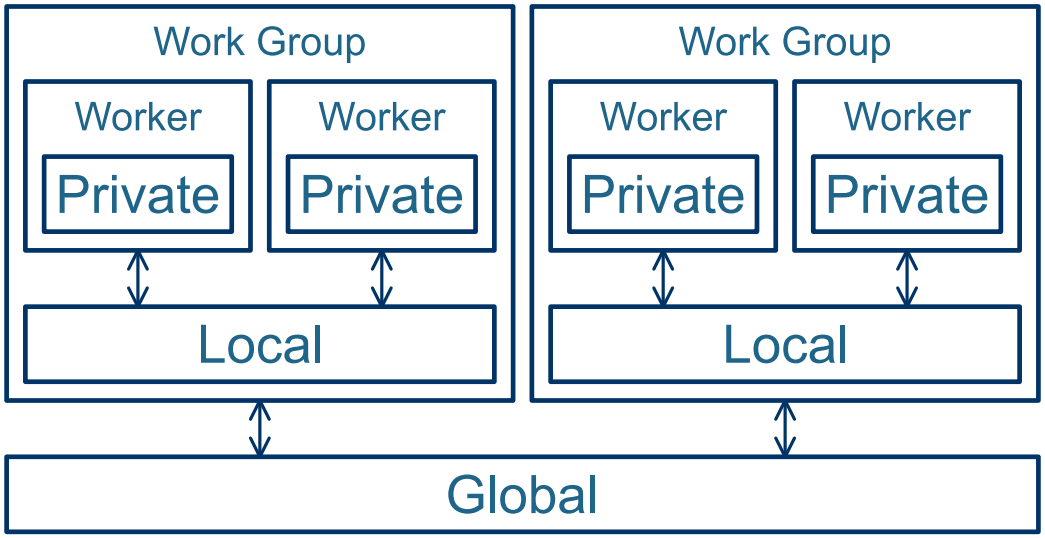
- Client-server interface with the CPU
- Runs a pipeline of kernels, in SIMD operation

Programming Model – GPU



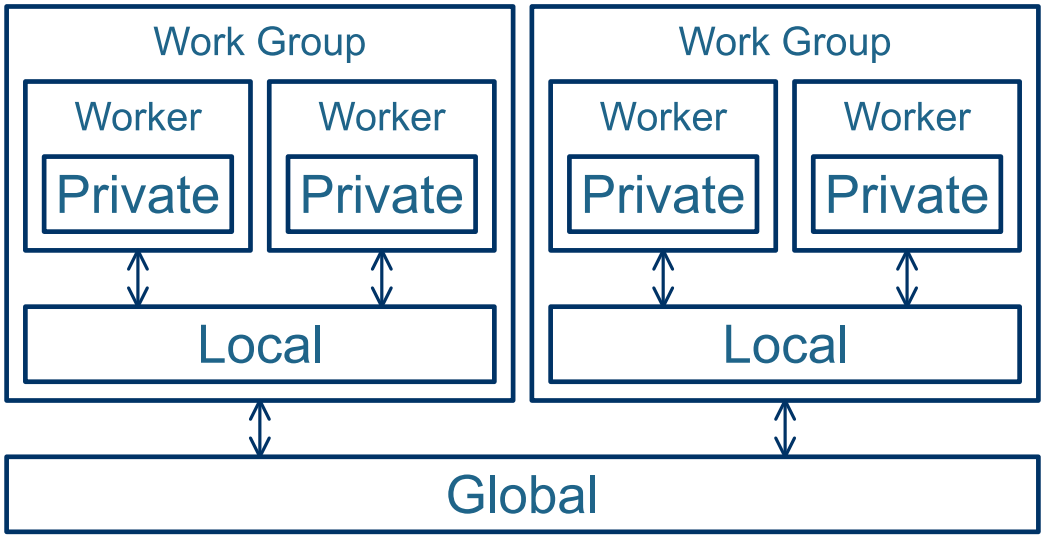
- The hardware is organised into groups of processors
- Within a group, execution is in lock-step
- Execution within one group is independent of other groups
- Three levels of memory

Programming Model – GPU



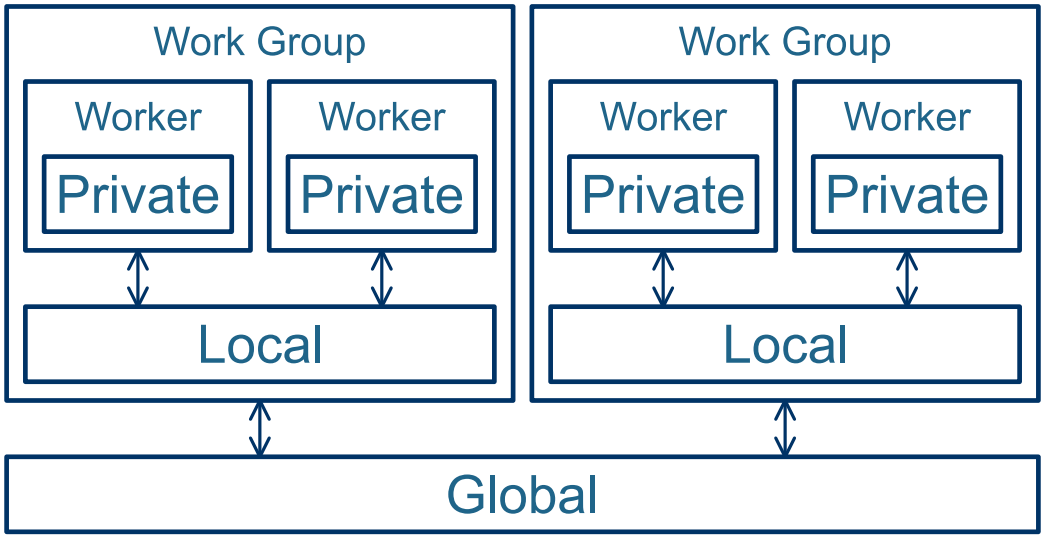
- The hardware is organised into groups of processors
- Within a group, execution is in lock-step
- Execution within one group is independent of other groups
- Three levels of memory

Programming Model – GPU



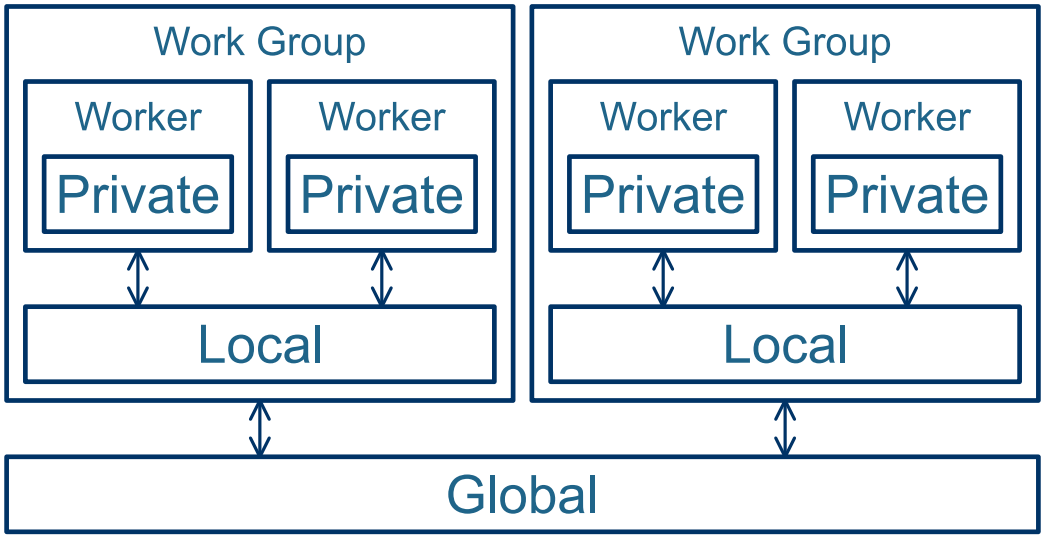
- The hardware is organised into groups of processors
- Within a group, execution is in lock-step
- Execution within one group is independent of other groups
- Three levels of memory

Programming Model – GPU



- The hardware is organised into groups of processors
- Within a group, execution is in lock-step
- Execution within one group is independent of other groups
- Three levels of memory

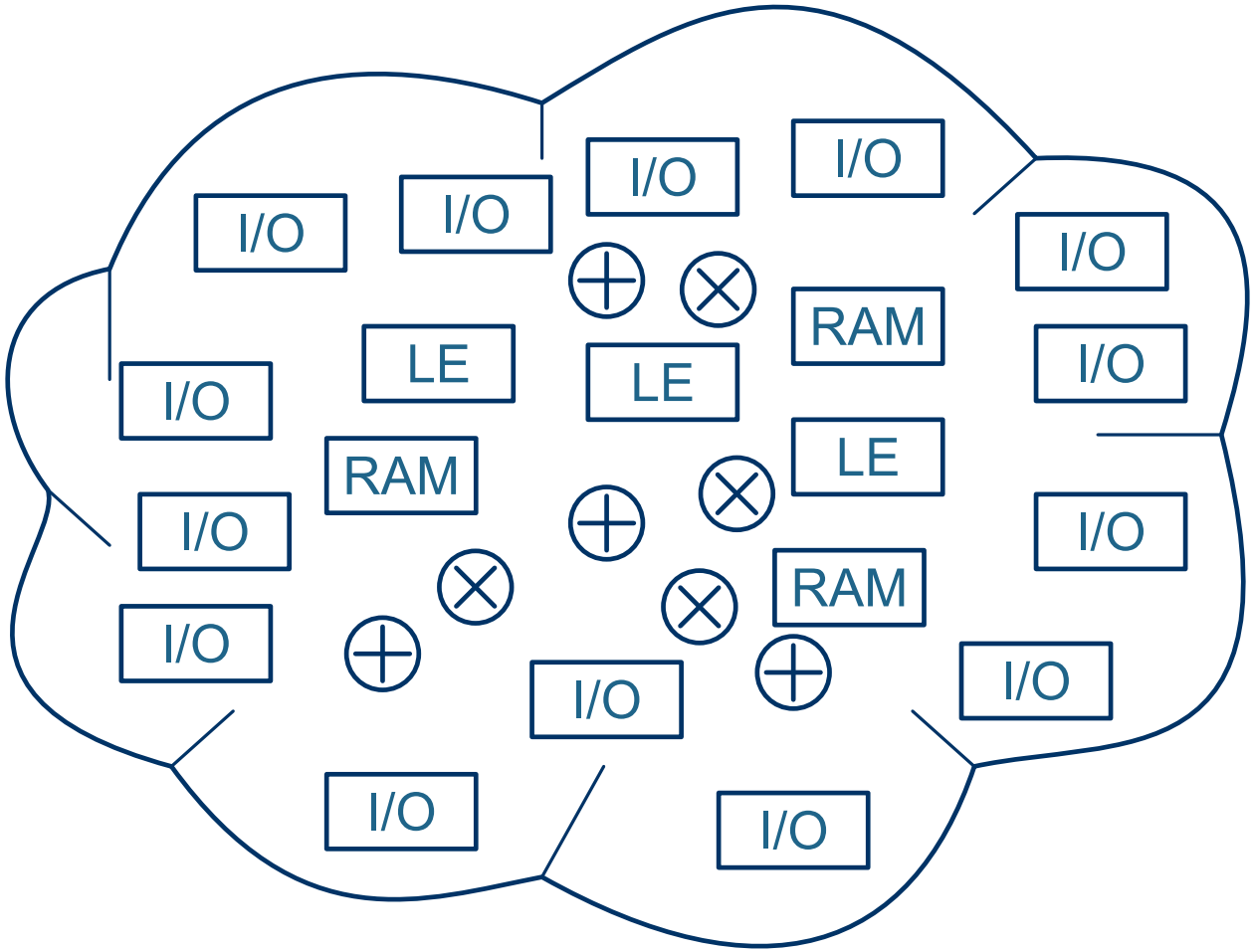
Programming Model – GPU



- The hardware is organised into groups of processors
- Within a group, execution is in lock-step
- Execution within one group is independent of other groups
- Three levels of memory

Programming Model – FPGA

- Any architecture you like...



Outline

Introduction

FPGA Internals

The Development Kit

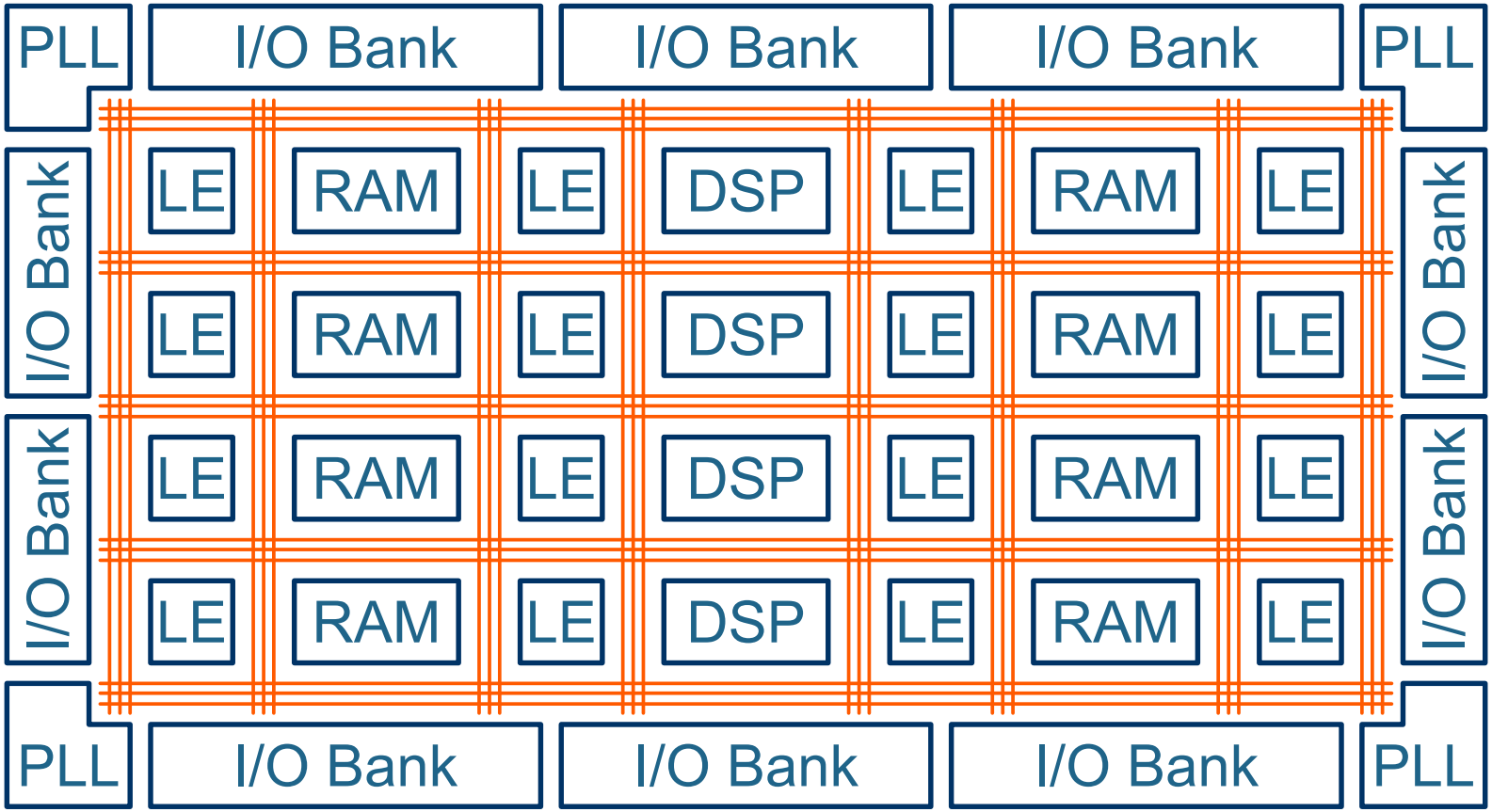
Development Cycle

Verilog Basics

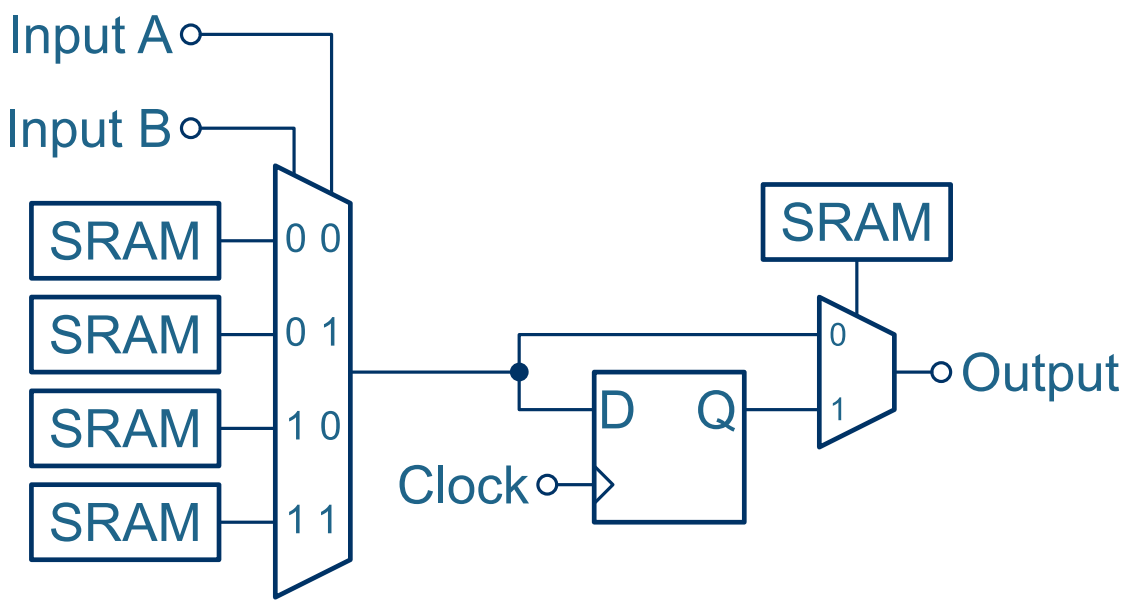
Simulation

THE RADAR MASTERS COURSE

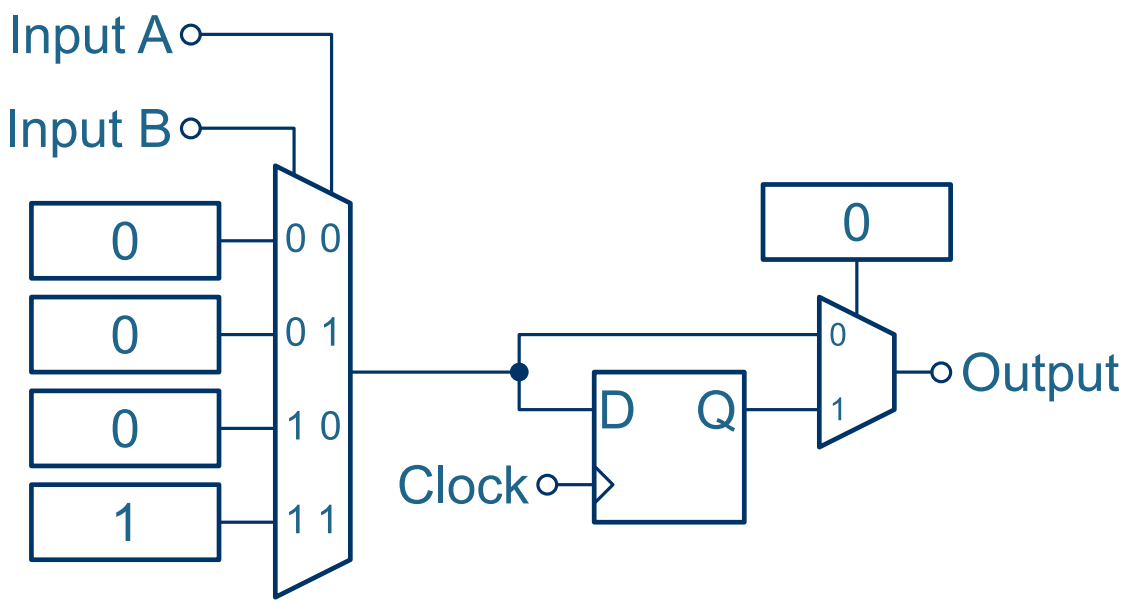
FPGA Architecture Overview



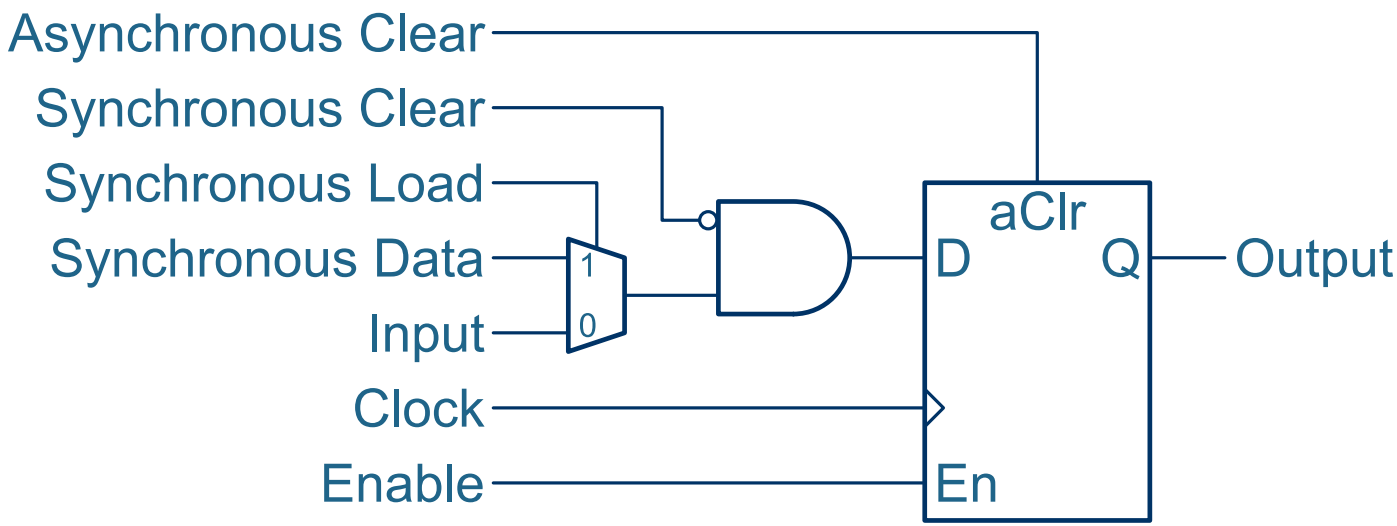
Logic Elements



Logic Elements



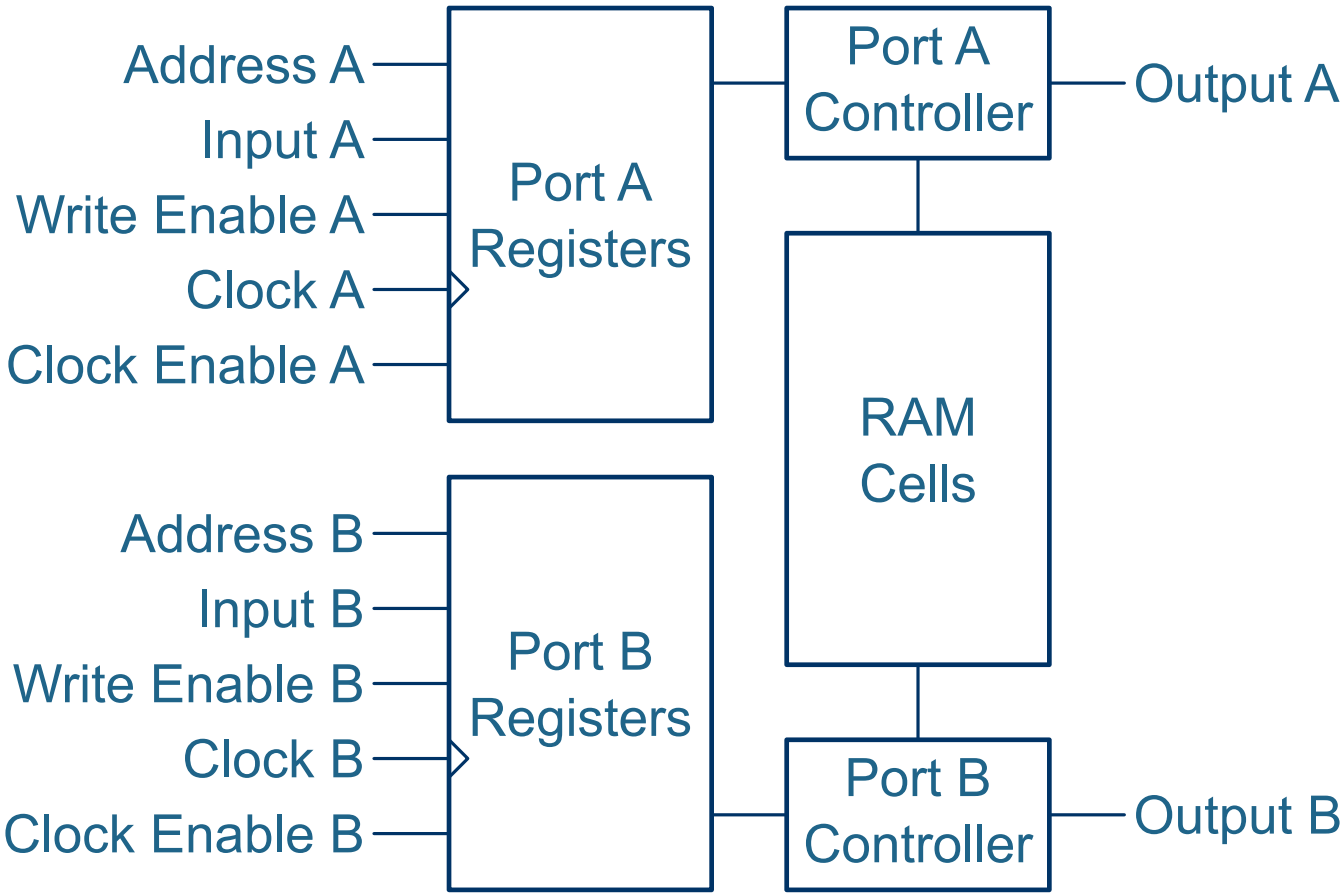
Register Detail



Register Detail

A-Clr	En	S-Clr	S-Ld	S-Dat	In	Clk	Output
1	×	×	×	×	×	×	0
0	1	1	×	×	×	↑	0
0	1	0	1	0	×	↑	0
0	1	0	1	1	×	↑	1
0	1	0	0	×	0	↑	0
0	1	0	0	×	1	↑	1
0	0	×	×	×	×	×	no-change

Internal RAM Blocks



Internal RAM Blocks

- Lattice embedded block RAM (EBR) blocks can be configured as:

  16384 × 1
  8192 × 2
  4096 × 4
  2048 × 9
  1024 × 18
  512 × 36 (Not for true dual-port)

- The two ports can have different configurations
- The two ports can have independent clocks

Internal RAM Blocks

- Lattice embedded block RAM (EBR) blocks can be configured as:

  16384 × 1
  8192 × 2
  4096 × 4
  2048 × 9
  1024 × 18
  512 × 36 (Not for true dual-port)

- The two ports can have different configurations
- The two ports can have independent clocks

Internal RAM Blocks

- Lattice embedded block RAM (EBR) blocks can be configured as:

  16384 × 1
  8192 × 2
  4096 × 4
  2048 × 9
  1024 × 18
  512 × 36 (Not for true dual-port)

- The two ports can have different configurations
- The two ports can have independent clocks

Internal RAM Blocks

- Internal RAM can be initialised by means of a memory initialisation file (`.mem`)

```
// Header Comments
#Format=Hex
#Depth=4096
#Width=8
#AddrRadix=3
#DataRadix=3
#Data
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 7E 81 A5 81 81 BD 99 81 81 7E 00 00 00 00
00 00 7E FF DB FF FF C3 E7 FF FF 7E 00 00 00 00
...
00 70 D8 30 60 C8 F8 00 00 00 00 00 00 00 00 00
00 00 00 00 7C 7C 7C 7C 7C 7C 7C 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 FF FF 00
```

Internal RAM Blocks

- Internal RAM can be initialised by means of a memory initialisation file (`.mem`)

```
// Header Comments
#Format=AddrHex
#Depth=4096
#Width=8
#AddrRadix=3
#DataRadix=3
#Data
012:7E 81 A5 81 81 BD 99 81 81 7E
022:7E FF DB FF FF C3 E7 FF FF 7E
034:6C FE FE FE FE 7C 38 10
...
FD1:70 D8 30 60 C8 F8
FE4:7C 7C 7C 7C 7C 7C 7C
FFD:FF FF
```

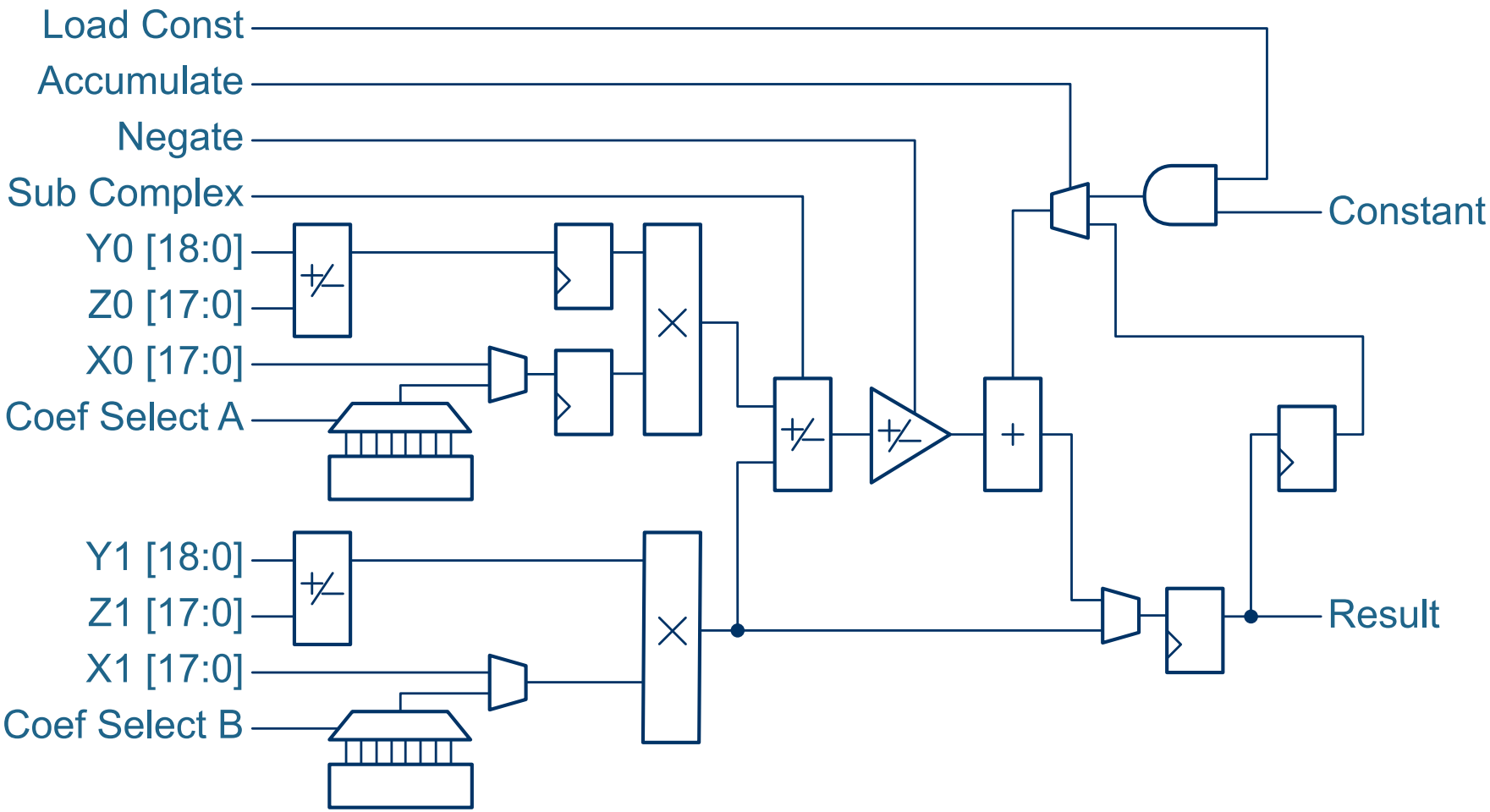
Internal RAM Blocks

- Internal RAM can be initialised by means of a memory initialisation file (`.mem`)
- Altera (Intel) uses MIF files

Internal RAM Blocks

- Internal RAM can be initialised by means of a memory initialisation file (`.mem`)
- Altera (Intel) uses MIF files
- Xilinx (AMD) uses COE files

DSP Blocks (Altera / Intel)



DSP Detail (Altera / Intel)

Each Cyclone V DSP block can be configured as:

- Three 9×9-bit multipliers
- Two 18×18-bit multipliers (unsigned)
- Two 18×19-bit multipliers (signed)
- One 18×25-bit multiplier
- One 20×24-bit multiplier
- One 27×27-bit multiplier
- One 18×19-bit multiply-accumulate
- One 18×18-bit multiply-accumulate with adder
- Half of a 18×19-bit complex multiplier

DSP Detail (Lattice)

- The Lattice XP2 have similar DSP blocks
- Each DSP block can be configured as
  two 9×9-bit multipliers or
  one 18×18-bit multiplier
- Each input can be configured as signed or unsigned
- Includes an accumulate function
- The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

DSP Detail (Lattice)

- The Lattice XP2 have similar DSP blocks
- Each DSP block can be configured as
  two 9×9-bit multipliers or
  one 18×18-bit multiplier
- Each input can be configured as signed or unsigned
- Includes an accumulate function
- The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

DSP Detail (Lattice)

- The Lattice XP2 have similar DSP blocks
- Each DSP block can be configured as
  two 9×9-bit multipliers or
  one 18×18-bit multiplier
- Each input can be configured as signed or unsigned
- Includes an accumulate function
- The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

DSP Detail (Lattice)

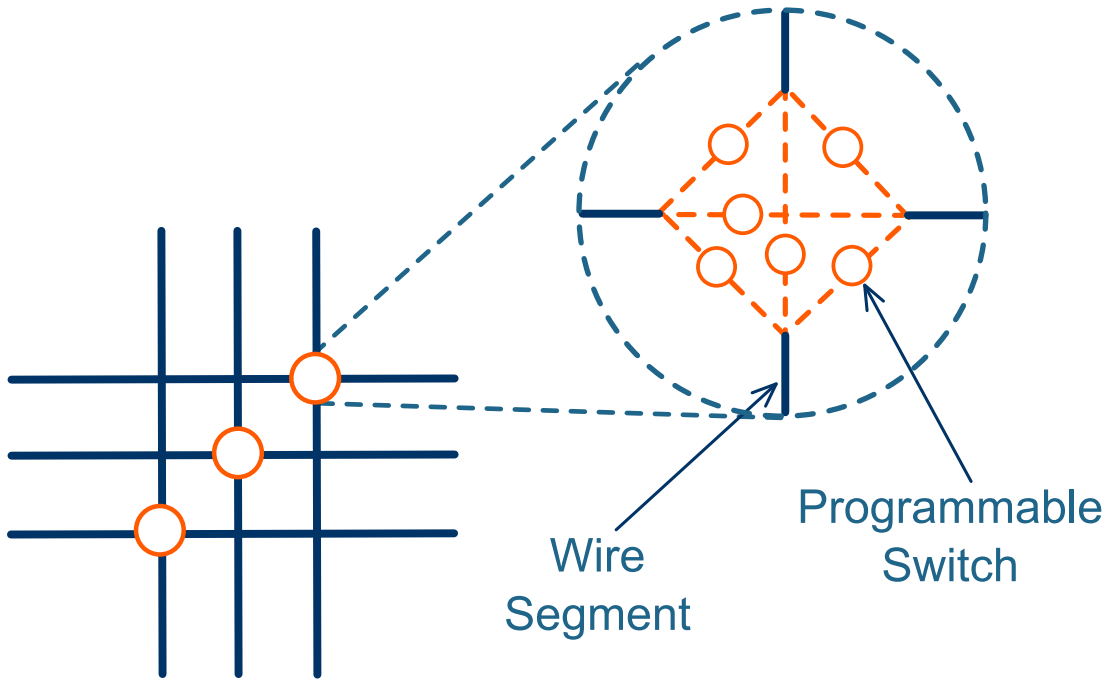
- The Lattice XP2 have similar DSP blocks
- Each DSP block can be configured as
  two 9×9-bit multipliers or
  one 18×18-bit multiplier
- Each input can be configured as signed or unsigned
- Includes an accumulate function
- The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

DSP Detail (Lattice)

- The Lattice XP2 have similar DSP blocks
- Each DSP block can be configured as
  two 9×9-bit multipliers or
  one 18×18-bit multiplier
- Each input can be configured as signed or unsigned
- Includes an accumulate function
- The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

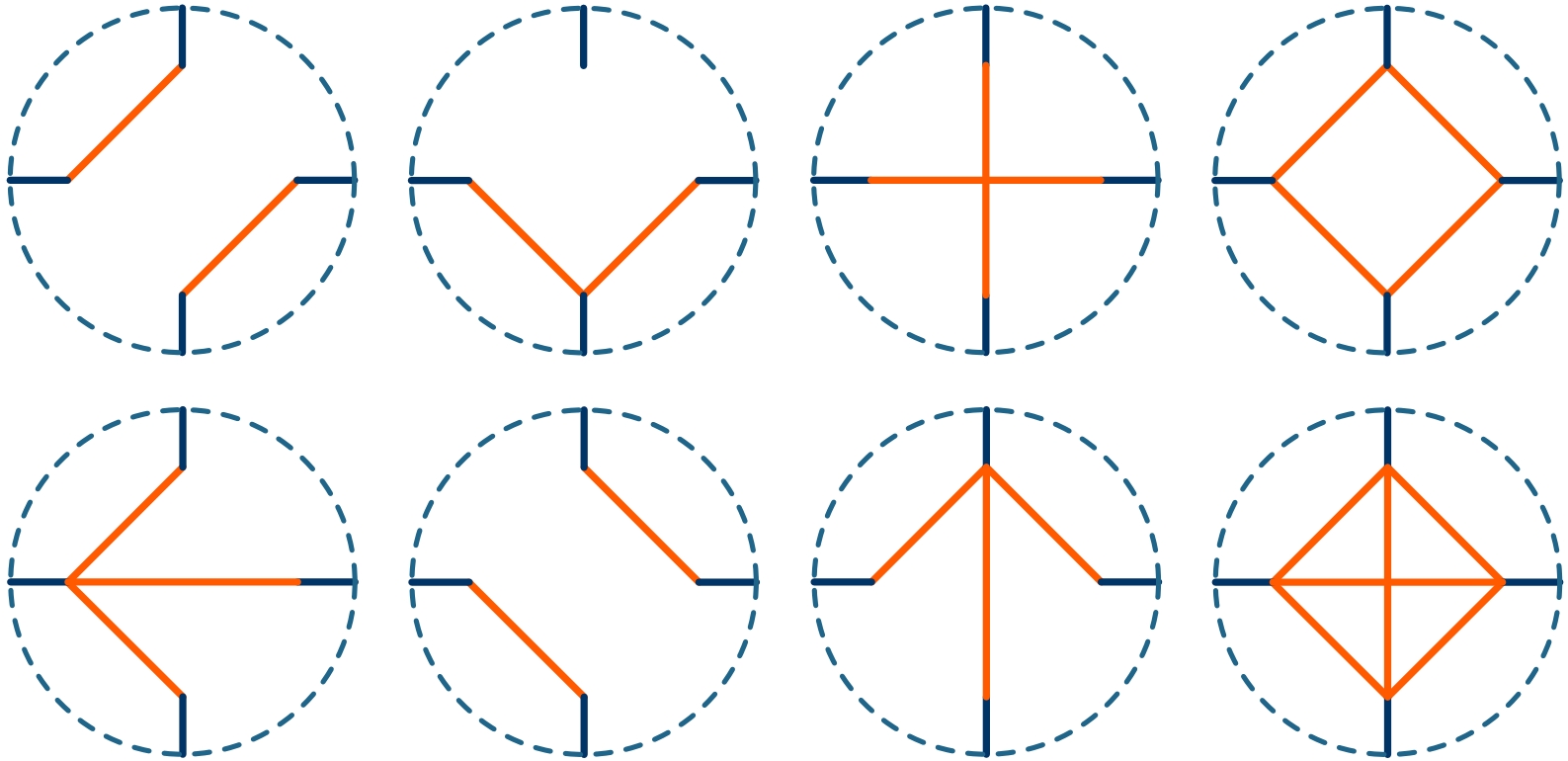
Interconnect

- Each interconnect crossing contains 6 switches
- These switches can be configured in various ways



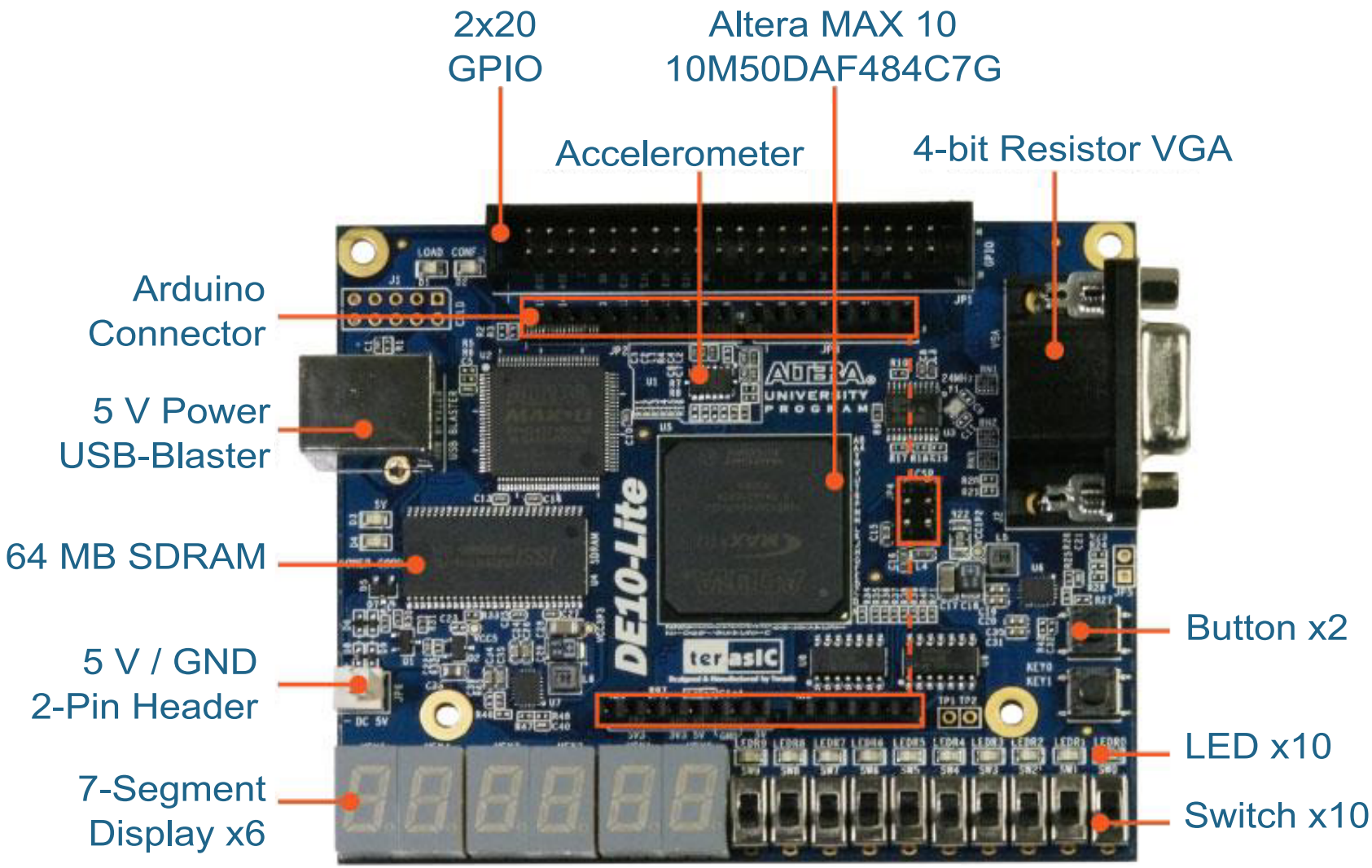
Interconnect

- Each interconnect crossing contains 6 switches
- These switches can be configured in various ways

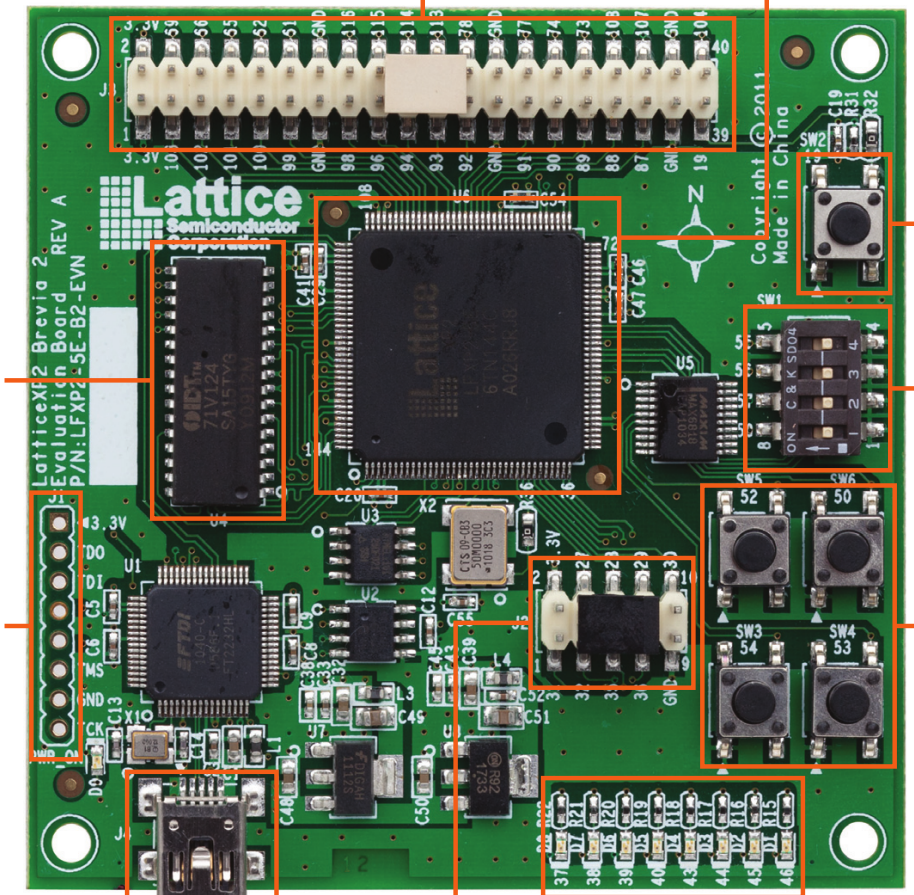


Outline

DE10 Lite Overview



LatticeXP2 Brevia2 Overview



FPGA Details

- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

FPGA Details

- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

FPGA Details

- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

FPGA Details

- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

FPGA Details

- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

FPGA Details

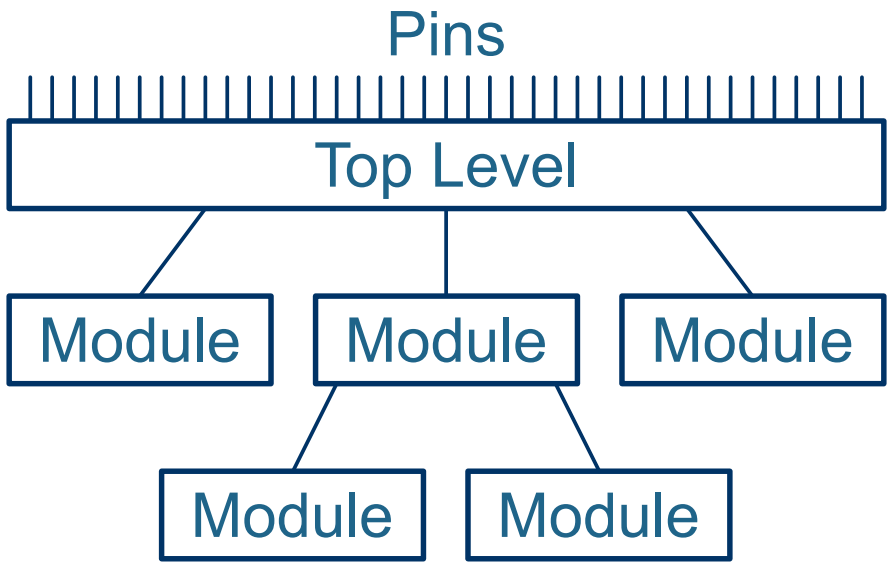
- LatticeXP2 series (LFXP2-5E-6TN144I)
- 4 752 logic elements
- 9 EBR memory blocks
- On-chip configuration flash memory
- 2 phase-locked loop blocks
- 100 I/O Pins

Coffee Break...

Outline

Development Cycle

Design Entry – Modular Design



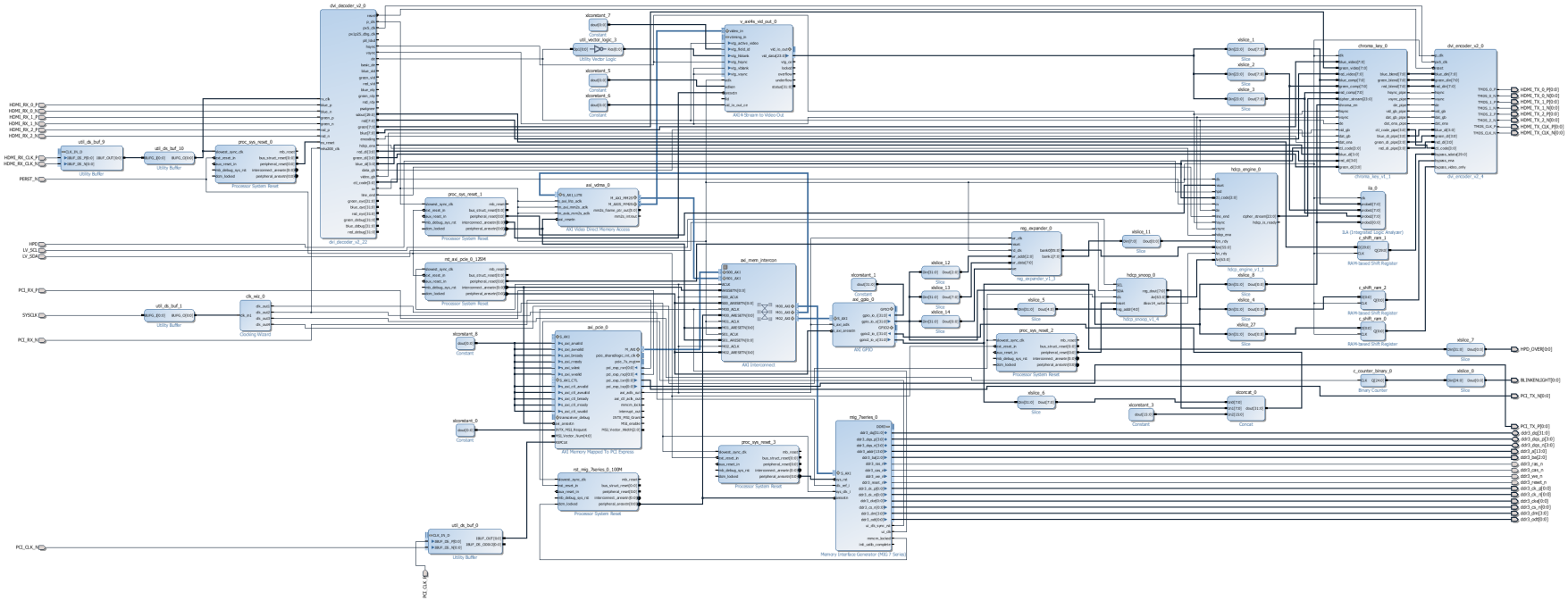
Design Entry – HDL

- Verilog / VHDL / PyHDL / Migen / etc.

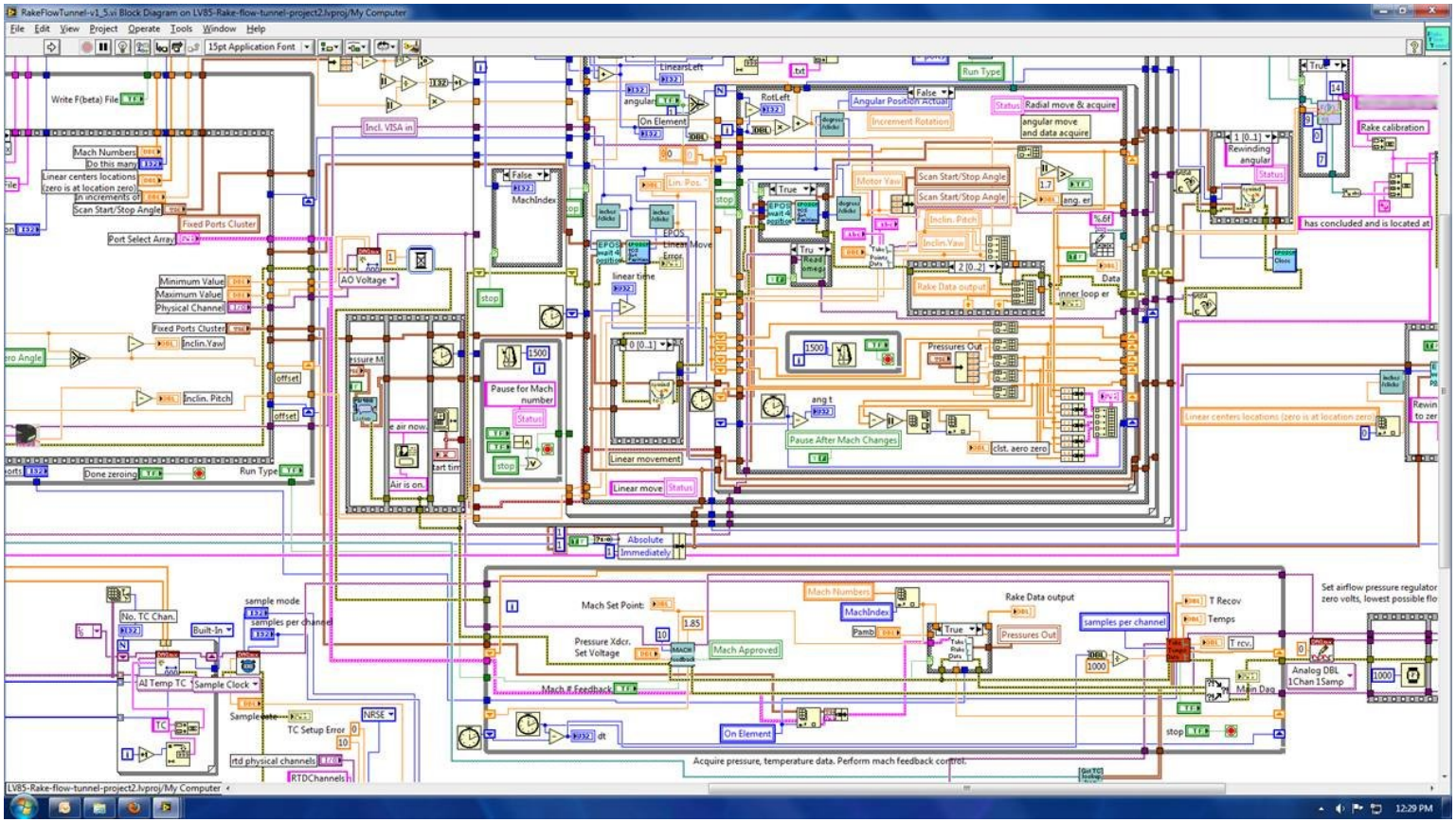
```
module Top_Level(input ipClk, input ipButton, output opLED);
  wire Debounced_Button;

  Debouncer Debouncer_Inst(
    ipClk, ipButton, Debounced_Button
  );

  LED_Driver LED_Driver_Inst(
    ipClk, Debounced_Button, opLED
  );
endmodule
```

Design Entry – Schematic

Design Entry – Schematic

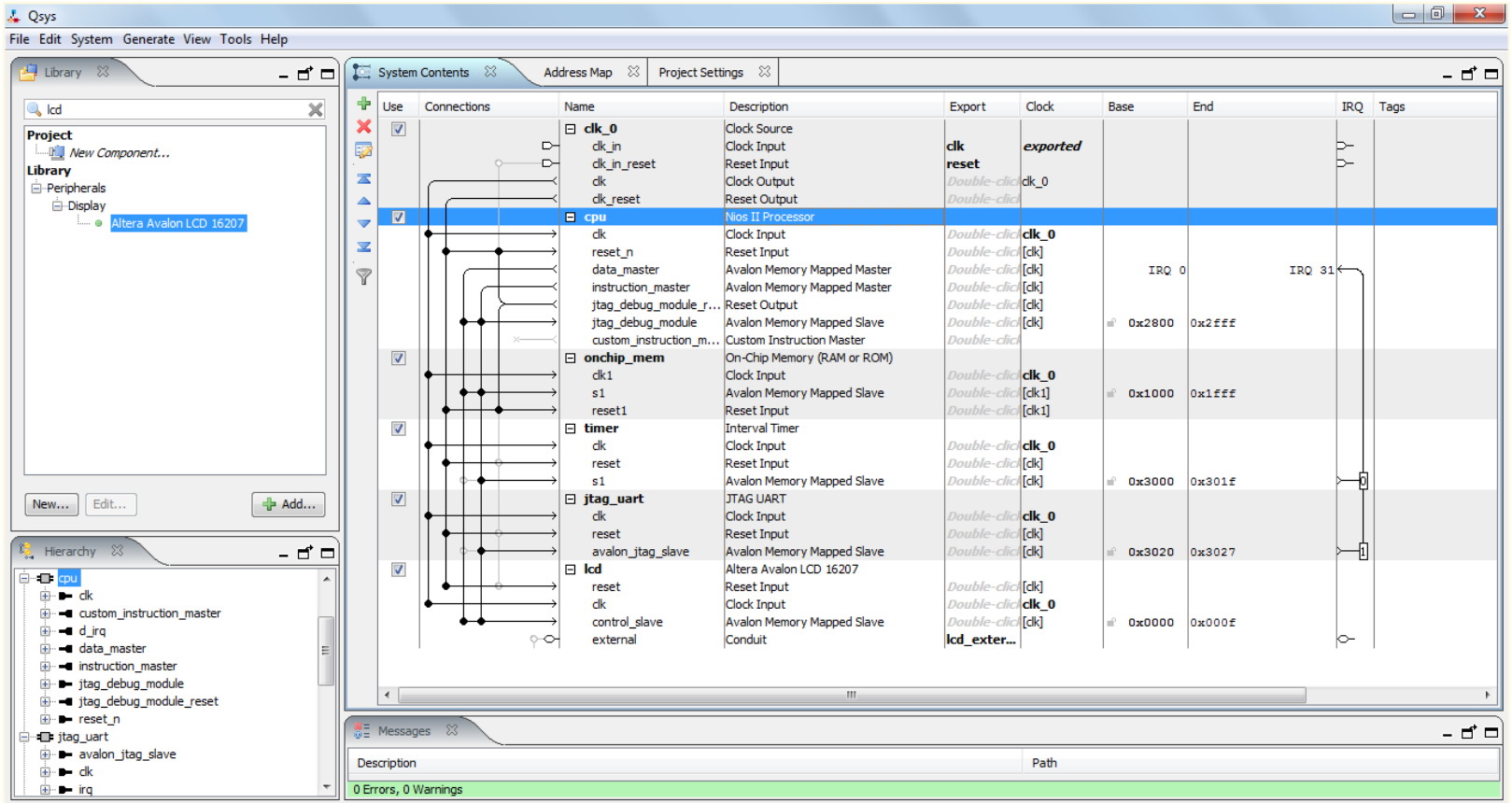


Design Entry – Schematic

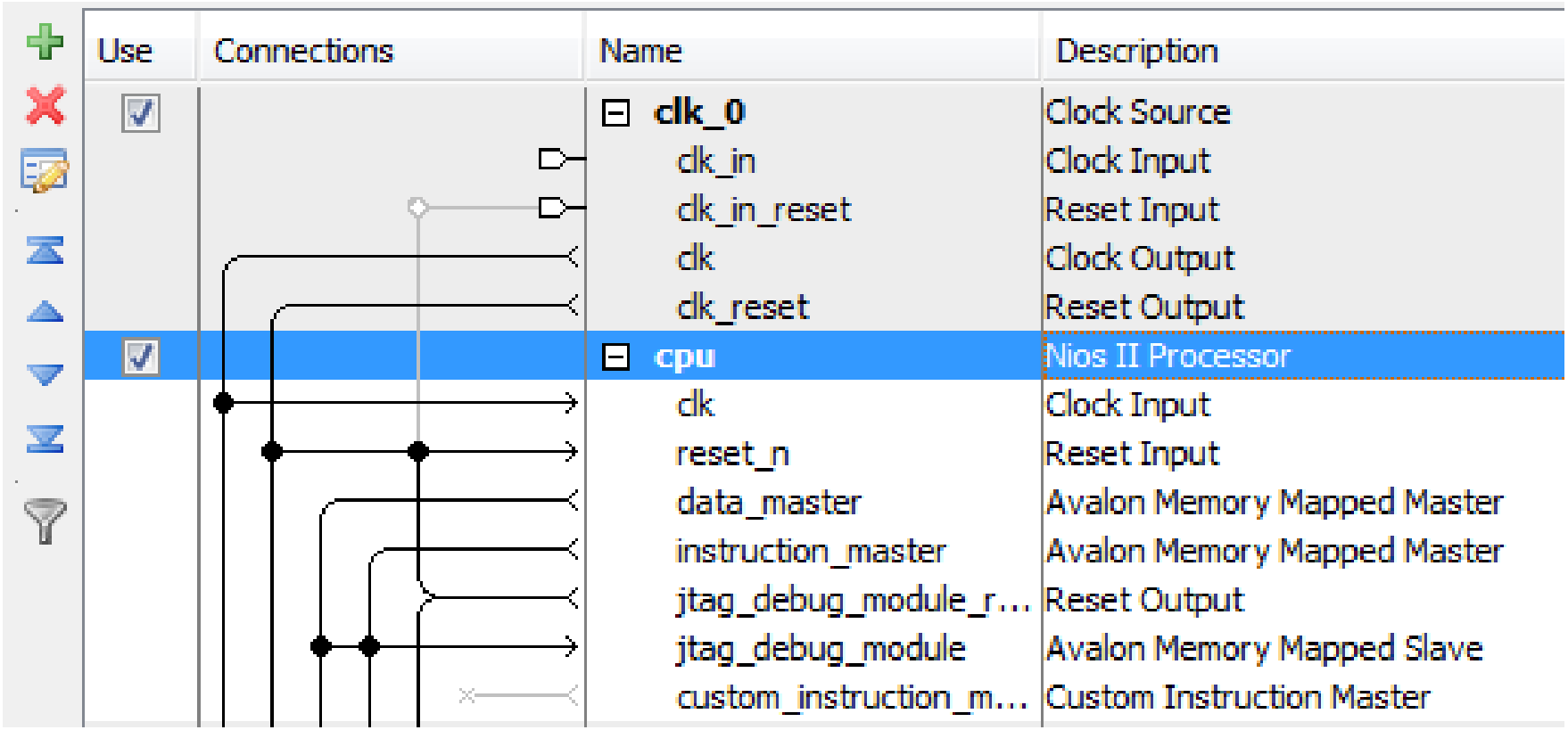


THE RADAR MASTERS COURSE

Design Entry – Qsys



Design Entry – Qsys



Design Entry – HLS


```
void paralleltest(
  bool _doWrite, int _writeAddr, int  _writeData,
  bool _doRead,  int _readAddr,  int* _readData
){
  #pragma HLS INTERFACE ap_ctrl_none port=return
  #pragma HLS PIPELINE II=1
  #pragma HLS DEPENDENCE variable=buffer inter WAR false
  #pragma HLS RESOURCE   variable=buffer core=RAM_2P_BRAM

  static const int BufferSize = 1024;
  static       int buffer[BufferSize];

  if (_doWrite){buffer[_writeAddr % BufferSize] = _writeData;}
  if (_doRead) {*_readData = buffer[_readAddr % BufferSize];}
}
```

Analysis and Synthesis

- Analyse code and perform optimisations (trim dead paths, check connectivity, etc.)
- Synthesise logic tables from expressions
- Match the logic tables and data flow graphs to the target hardware architecture
- Synthesise connection graphs

Note: Verilog automatically generates a 1-bit wire for any net that is used without definition, which often happens on typo's and spelling errors. Consult the synthesis warnings for "not declared" nets.

Analysis and Synthesis

- Analyse code and perform optimisations (trim dead paths, check connectivity, etc.)
- Synthesise logic tables from expressions
- Match the logic tables and data flow graphs to the target hardware architecture
- Synthesise connection graphs

Note: Verilog automatically generates a 1-bit wire for any net that is used without definition, which often happens on typo's and spelling errors. Consult the synthesis warnings for "not declared" nets.

Analysis and Synthesis

- Analyse code and perform optimisations (trim dead paths, check connectivity, etc.)
- Synthesise logic tables from expressions
- Match the logic tables and data flow graphs to the target hardware architecture
- Synthesise connection graphs

Note: Verilog automatically generates a 1-bit wire for any net that is used without definition, which often happens on typo's and spelling errors. Consult the synthesis warnings for "not declared" nets.

Analysis and Synthesis

- Analyse code and perform optimisations (trim dead paths, check connectivity, etc.)
- Synthesise logic tables from expressions
- Match the logic tables and data flow graphs to the target hardware architecture
- Synthesise connection graphs

Note: Verilog automatically generates a 1-bit wire for any net that is used without definition, which often happens on typo's and spelling errors. Consult the synthesis warnings for "not declared" nets.

Analysis and Synthesis

- Analyse code and perform optimisations (trim dead paths, check connectivity, etc.)
- Synthesise logic tables from expressions
- Match the logic tables and data flow graphs to the target hardware architecture
- Synthesise connection graphs

Note: Verilog automatically generates a 1-bit wire for any net that is used without definition, which often happens on typo's and spelling errors. Consult the synthesis warnings for "not declared" nets.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient … … … … … …
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

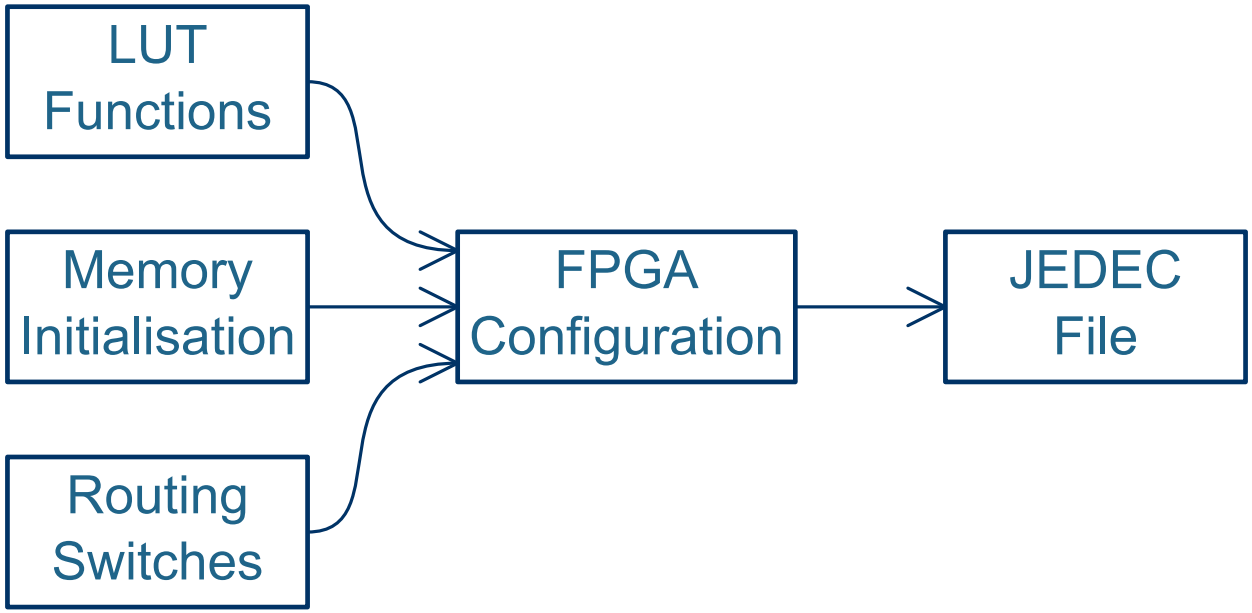
Fitter

- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Fitter

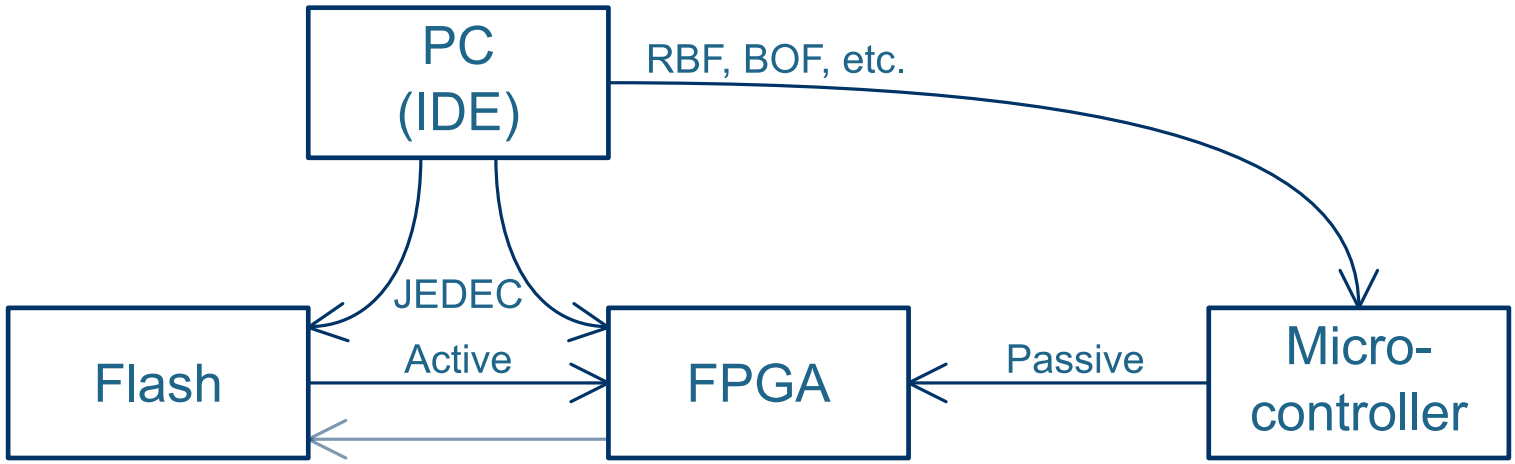
- The fitter performs a place and route function (similar to a PCB auto-router)
- This is a computationally-intensive process, so be patient ... ...
- Uses design constraints:
  - Clock rate
  - Combinational logic time delay
  - Multi-cycle timing requirements
  - False-path specifications
  - Routing delay
  - External timing requirements
  - User-defined placement
  - etc.

Assembler

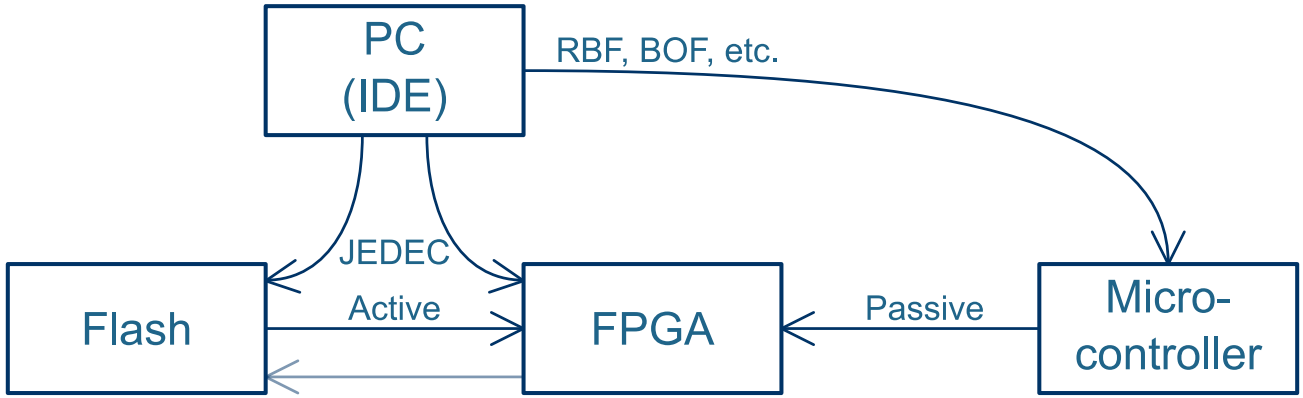

LUT
Functions
Memory
Initialisation
Routing
Switches
FPGA
Configuration
JEDEC
File


Analysis &
Synthesis
Fitter
Assembler
Programmer
FPGA

Programming Architectures



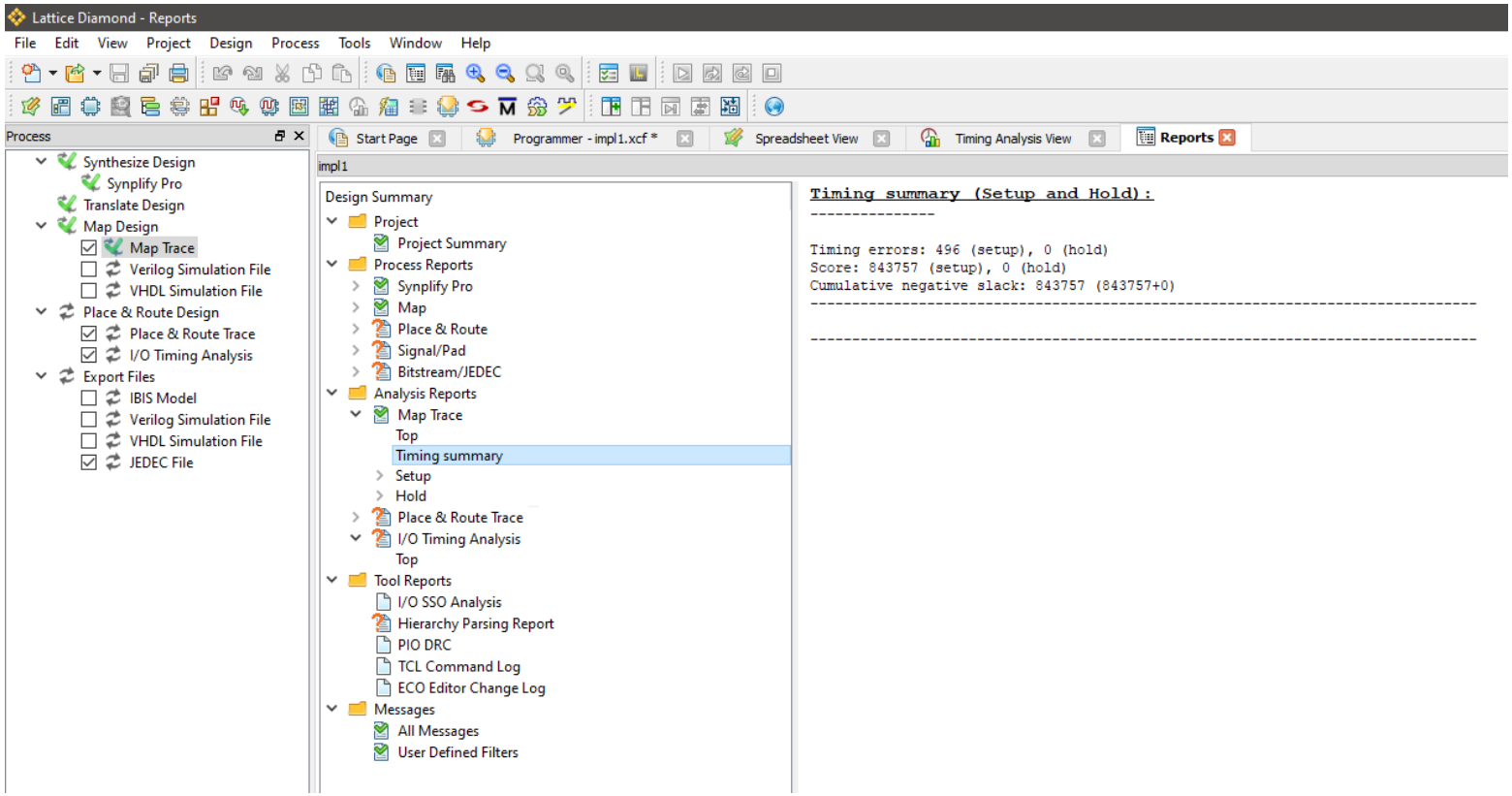
Programming Architectures



- JEDEC ⇒ Joint Electron Device Engineering Council
- RBF ⇒ Raw Binary File
- BOF ⇒ Borph Object File

Lattice Diamond

Practical `01 - Getting Started` introduces the Lattice Diamond IDE.



Outline

Module Definition


```
/* Use comments to describe the function...
   Multiple lines are possible */

module MyModule(
  input         ipClk,   // Try to be consistent
  input         ipReset, // on port placement

  input  [ 7:0]ipInput,  // Note that Verilog is
  output [ 9:0]opOutput, // case sensitive
  inout  [12:0]bpBidirectional
);

// The module body goes here...

endmodule
```

Wires

- Wires are internal connections
- See them as wires on a bread-board...

```
wire      A;       // A single-bit wire
wire [7:0]B;       // An 8-bit wire
wire [7:0]C[5:0]; // A 6-element array of 8-bit wires

wire [7:0]X = B + C[3]; // Wire definition and
                        // assignment in one
```

Wires

- Wires are internal connections
- See them as wires on a bread-board...

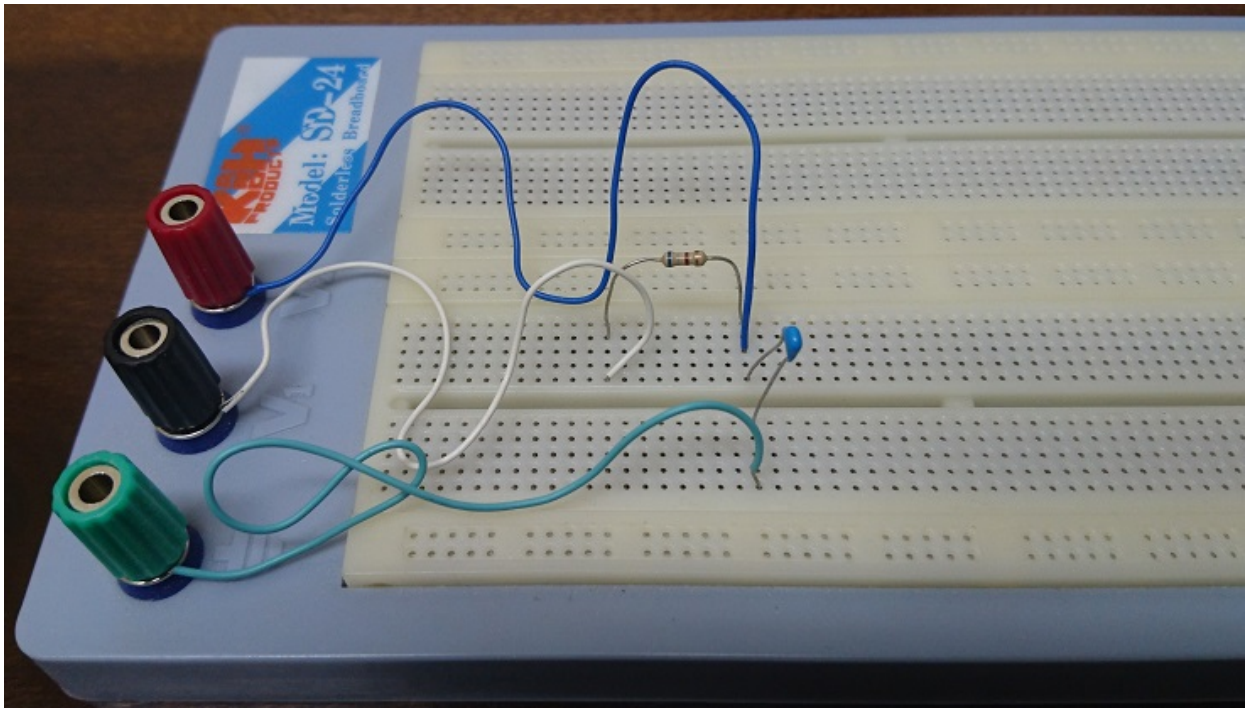
```
wire      A;       // A single-bit wire
wire [7:0]B;       // An 8-bit wire
wire [7:0]C[5:0]; // A 6-element array of 8-bit wires

wire [7:0]X = B + C[3]; // Wire definition and
                        // assignment in one
```

Wires

- Wires are internal connections
- See them as wires on a bread-board...

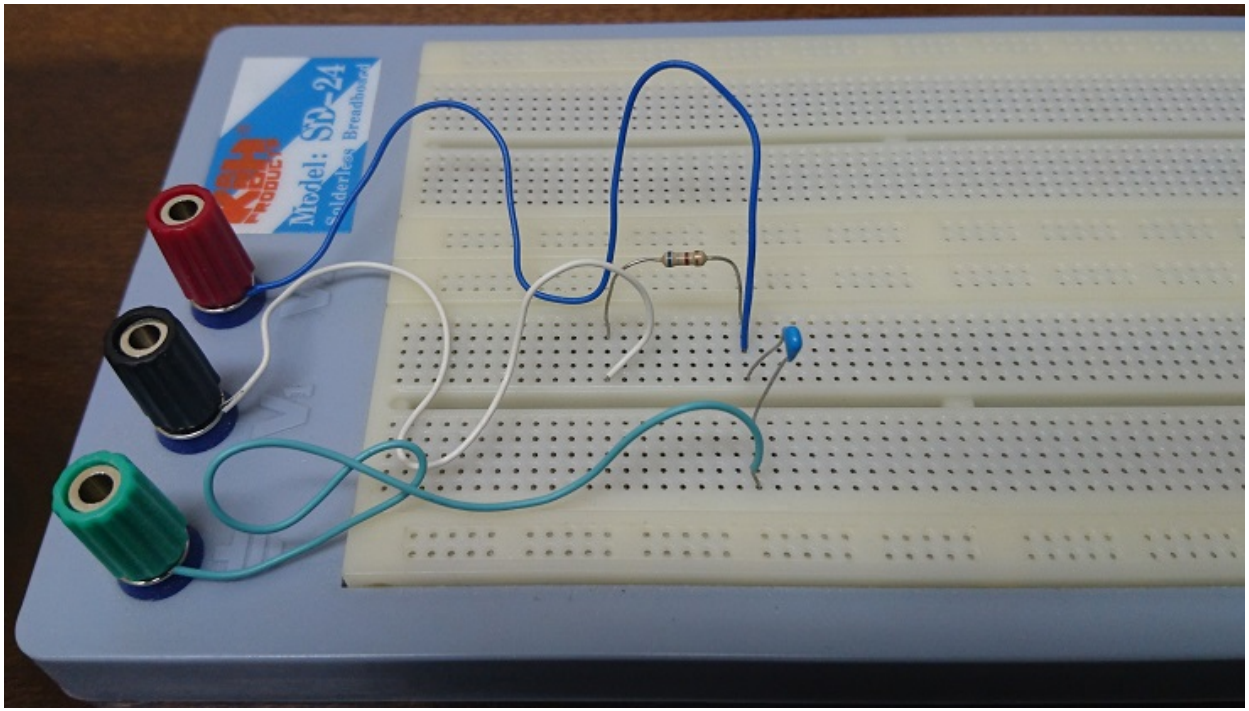
```
wire      A;       // A single-bit wire
wire [7:0]B;       // An 8-bit wire
wire [7:0]C[5:0]; // A 6-element array of 8-bit wires

wire [7:0]X = B + C[3]; // Wire definition and
                        // assignment in one
```

Wires




```
module PWM_Filter(input V1, output V2, input GND);
  wire Blue; wire White = V1; wire Cyan = GND;
  Resistor  #(  680) R1(White, Blue);
  Capacitor #(10000) C1(Blue , Cyan);
  assign V2 = Blue;
endmodule
```

Wires




```
module PWM_Filter(input V1, output V2, input GND);
  Resistor  #(  680) R1(V1, V2 );
  Capacitor #(10000) C1(V2, GND);
endmodule
```

Operators – Reduction


```
wire [7:0] X, Y, Z; // Defines 3x 8-bit wires
wire       A, B, C; // Defines 3x 1-bit wires

assign A = &X; // AND-reduce X and assign to A
assign B = |Y; //  OR-reduce Y and assign to B
assign C = ^Z; // XOR-reduce Z and assign to C
assign B = !Y; // Logical NOT: equivalent to B = ~|Y
```

Operators – Bitwise


```
wire [7:0] X, Y, Z; // Defines 3x 8-bit wires

assign Z = -X;     // 2's complement X and assign to Z
assign Z = ~Y;     // Bitwise NOT Y and assign to Z
assign Z = X | Y; // Bitwise OR  X with Y and assign to Z
assign Z = X & Y; // Bitwise AND X with Y and assign to Z
assign Z = X ^ Y; // Bitwise XOR X with Y and assign to Z
```

Operators – Concatenation


```
wire [7:0] A;
wire [3:0] B, C;

assign A = {B, C}; // Concatenates B--C and assign to A
assign {B, C} = A; // Assign A to the concatenation B--C
assign A = {2{B}}; // Replicate B 2 times and assign to A
```

Operators – Arithmetic


```
wire [ 7:0] A, B, C;
wire [15:0] X;

assign A = B + C; // Add C to B and assign to A
assign A = B - C; // Subtract C from B and assign to A
assign X = B * C; // Multiply B by C and assign to X

assign A = B << 5; // Left-shift B and assign to A
assign A = B >> 6; // Right-shift B and assign to A
```

Operators – Logical


```
wire [ 7:0]A, B, C;
wire       X;

assign X = A >  B; // A greater than B?          Result to X
assign X = A <  B; // A less than B?             Result to X
assign X = A >= B; // A greater or equal to B? Result to X
assign X = A <= B; // A less or equal to B?     Result to X
assign X = A == B; // A equal to B?  A          Result to X
assign X = A != B; // A not equal to B?         Result to X

assign X = A && B; // Equivalent to X = (|A) & (|B)
assign X = A || B; // Equivalent to X = (|A) | (|B)

assign C = X ? A : B // If X is 1, assign A to C,
                     // otherwise assign B to C
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the 's' modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the 's' modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the 's' modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the ‘s’ modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the ‘s’ modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the ‘s’ modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire        [ 7:0]A; // Unsigned vector
wire signed [ 7:0]B; //   Signed vector
wire signed [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- All operations are unsigned, unless specified otherwise
- The following always yield unsigned results:
  - Any operation with at least one unsigned operand
  - A literal without the ‘s’ modifier
  - Bit-select (eg. `A[5]`)
  - Part-select (eg. `A[5:3]`)
  - Concatenations

```
wire          [ 7:0]A; // Unsigned vector
wire signed   [ 7:0]B; //   Signed vector
wire signed   [15:0]X; //   Signed vector

assign X = $signed(A) * B; // A must be cast
```

Operators – Signed Operations

- In general, most operations should be done using standard unsigned arithmetic
- In the rare cases where the sign makes a difference (multiplication and relational statements), cast explicitly

```
wire [ 7:0]A;
wire [ 7:0]B;
wire [15:0]X;

assign X = $signed(A) * $signed(B);

PWM PWM1(Clk, {~X[15], X[14:0]}, PWM_Output);
```

Operators – Signed Operations

- In general, most operations should be done using standard unsigned arithmetic
- In the rare cases where the sign makes a difference (multiplication and relational statements), cast explicitly

```
wire [ 7:0]A;
wire [ 7:0]B;
wire [15:0]X;

assign X = $signed(A) * $signed(B);

PWM PWM1(Clk, {~X[15], X[14:0]}, PWM_Output);
```

Vector Lengths

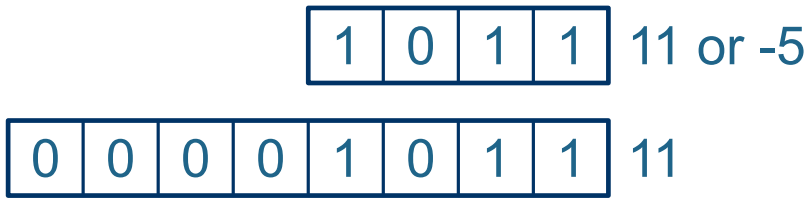
Note: Verilog does not enforce vector length matching:

- If the left-hand-side is longer than the right-hand-side, the most-significant side is padded with zeros
- If the left-hand-side is shorter than the right-hand-side, the most-significant side is truncated
- **This is done without warning!**
- Consult the compile warnings to check for unintended size-mismatches

Vector Lengths

Note: Verilog does not enforce vector length matching:

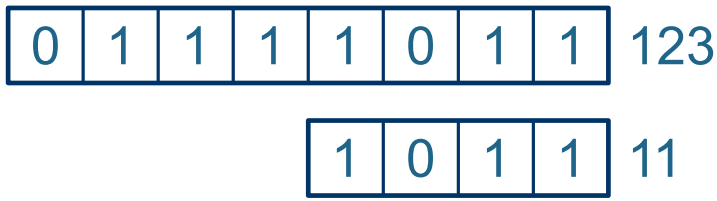
- If the left-hand-side is longer than the right-hand-side, the most-significant side is padded with zeros
- If the left-hand-side is shorter than the right-hand-side, the most-significant side is truncated
- **This is done without warning!**
- Consult the compile warnings to check for unintended size-mismatches



Vector Lengths

Note: Verilog does not enforce vector length matching:

- If the left-hand-side is longer than the right-hand-side, the most-significant side is padded with zeros
- If the left-hand-side is shorter than the right-hand-side, the most-significant side is truncated
- **This is done without warning!**
- Consult the compile warnings to check for unintended size-mismatches



Vector Lengths

Note: Verilog does not enforce vector length matching:

- If the left-hand-side is longer than the right-hand-side, the most-significant side is padded with zeros
- If the left-hand-side is shorter than the right-hand-side, the most-significant side is truncated
- **This is done without warning!**
- Consult the compile warnings to check for unintended size-mismatches

Vector Lengths

Note: Verilog does not enforce vector length matching:

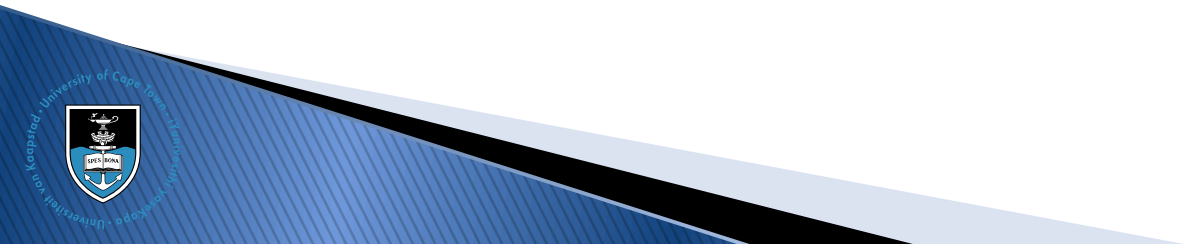
- If the left-hand-side is longer than the right-hand-side, the most-significant side is padded with zeros
- If the left-hand-side is shorter than the right-hand-side, the most-significant side is truncated
- **This is done without warning!**
- Consult the compile warnings to check for unintended size-mismatches

Literals


```
// All underscores "_" are ignored
 5'b_1_0110  //  5-bit binary
11'h_5_CE    // 11-bit hexadecimal
13'o_1_7642  // 13-bit octal
17'd_123_456 // 17-bit decimal

"H" // 8-bit ASCII constant

// Left bits (most significant) are padded with zeros,
// unless the most significant specified is 'Z' or 'X'
78'bZ // 78-bit high-impedance

// Use the 's' specifier for "signed" literals
-8'sd125 // 2's complement of the positive
         // signed decimal constant 125
```

Module Instances


```
module My_Submodule(input Clk, input A, input B, output X);
  // Module body...
endmodule

module Top_Level(
  input  Clock_50MHz,
  input  Input1, Input2,
  output Output
);

// Positional port mapping:
My_Submodule Instance_1(
  Clock_50MHz,
  Input1, Input2,
  Output
);
```

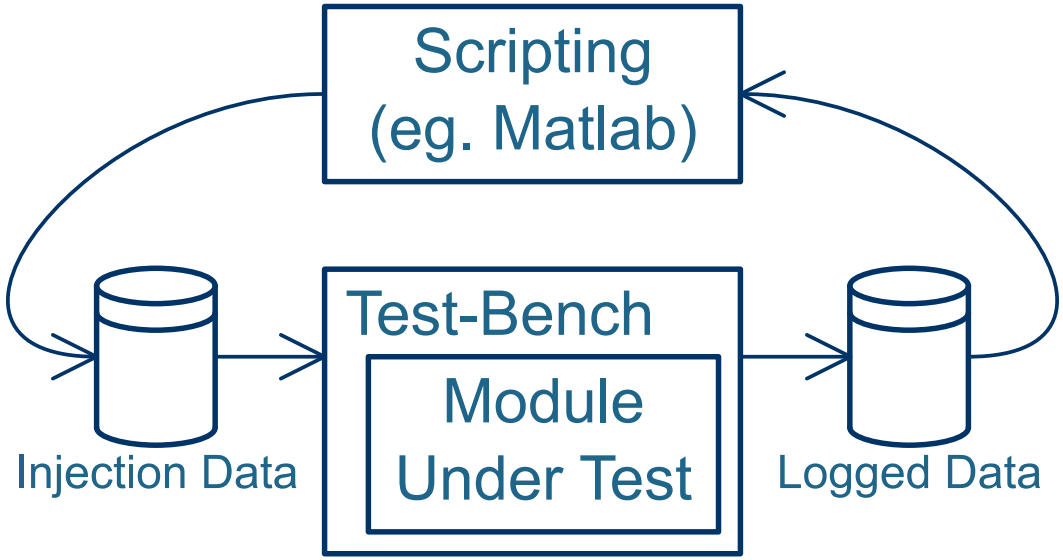


Module Instances


```
module My_Submodule(input Clk, input A, input B, output X);
  // Module body...
endmodule

module Top_Level(
  input  Clock_50MHz,
  input  Input1, Input2,
  output Output
);

// Named port mapping:
My_Submodule Instance_2(
  .X   (Output     ),
  .A   (Input1     ),
  .B   (Input2     ),
  .Clk(Clock_50MHz)
);
```

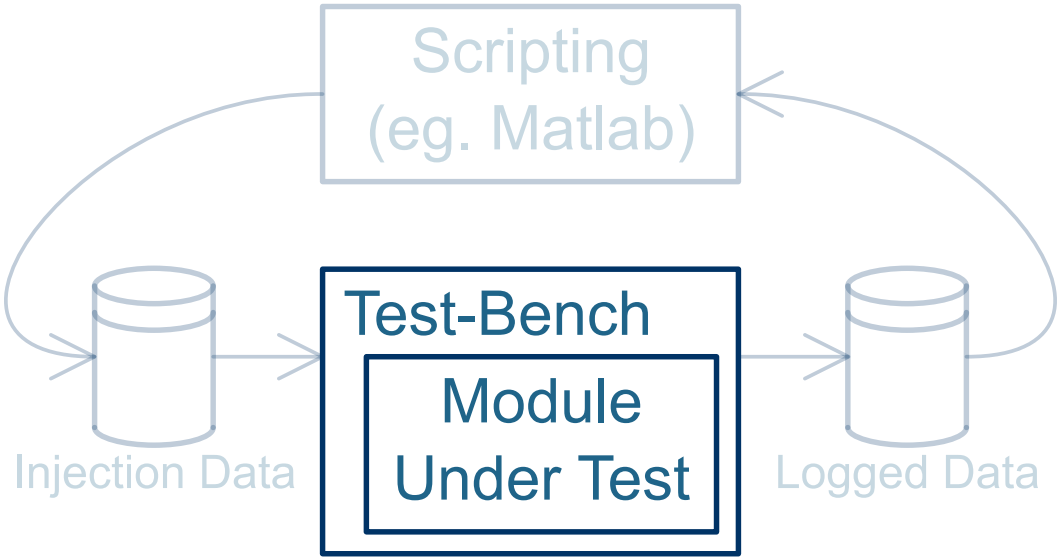
Outline

Simulation Basics



Today we only focus on the test-bench and simulation tool

Simulation Basics



Today we only focus on the test-bench and simulation tool

Test Bench Basics


```
`timescale 1ns/1ps
module ADXL345_TB;

// Clock
reg Clk_100M = 0;
always #5 Clk_100M <= ~Clk_100M;

// Reset
reg Reset = 1;
initial #20 Reset <= 0;
```

Test Bench Basics


```
// DUT
wire [15:0]X, Y, Z;
wire nCS, SClk, SDI;
reg  SDO = 0;

ADXL345 #(10) Accelerometer( // Set parameter for 100 MHz
  Clk_100M, Reset,
  X, Y, Z,
  nCS, SClk, SDI, SDO
);
```

Test Bench Basics


```
reg [ 7:0]DataIn;
reg [15:0]DataOut = 0;

integer n;
always begin
  @(negedge nCS);

  // Instruction word
  for(n = 7; n >= 0; n--) begin
    @(negedge SClk);
    DataIn[n] = SDI;
  end
```

Test Bench Basics

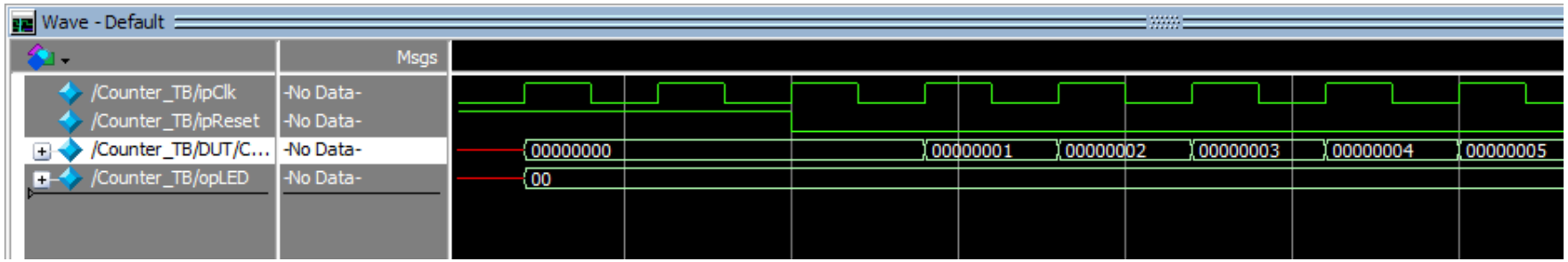

```
// The first data word
for(n = 7; n >= 0; n--) begin
  @(negedge SClk); #40 // Output delay;
  SDO <= DataOut[n];
end

// The optional second data word
if(DataIn[6]) begin // More bits
  for(n = 15; n >= 8; n--) begin
    @(negedge SClk); #40 // Output delay;
    SDO <= DataOut[n];
  end
end
```

Test Bench Basics


```
    @(posedge nCS);

    DataOut = DataOut + 1;
end

endmodule
```

Modelsim

Practical `02 - Simulation` introduces the Modelsim simulation tool.



Select References

Stephen Brown and Zvonko Vranesic
Fundamentals of Digital Logic with Verilog Design, 2nd Edition
ISBN 978-0-07-721164-6

Merrill L Skolnik
Introduction to RADAR Systems
ISBN 978-0-07-288138-7

Mark A. Richards and James A. Scheer
Principles of Modern Radar: Basic Principles
ISBN 978-1-89-112152-4

Deepak Kumar Tala
World of ASIC
http://www.asic-world.com/

Jean P. Nicolle
FPGA 4 Fun
http://www.fpga4fun.com/

FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Dr Stephen Paine

Day 1 – 27 April 2022