

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

Introduction

FPGA Internals

The Development Kit

Development Cycle

Verilog Basics

Simulation



Outline

Introduction

FPGA Internals

The Development Kit

Development Cycle

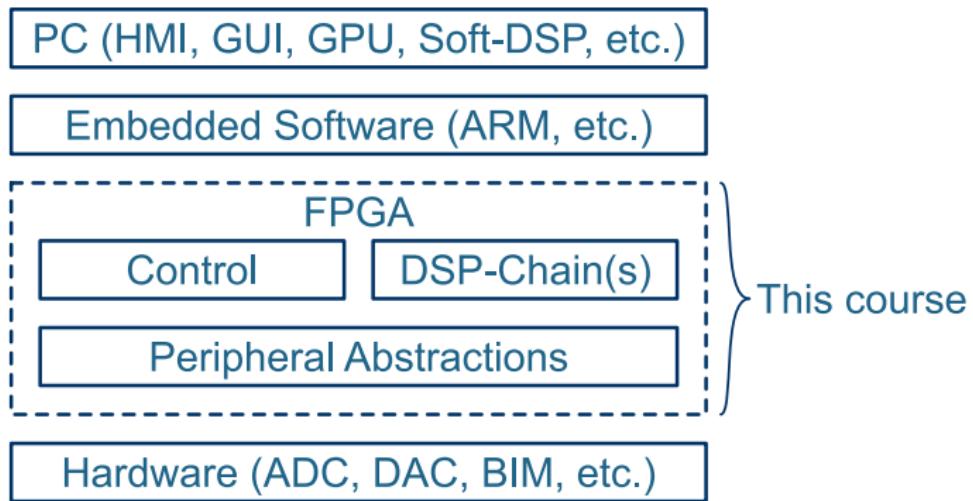
Verilog Basics

Simulation



Course Overview

2 of 49



- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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

- ▶ 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**

► 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 FPGA tool-set, including 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.

- ▶ 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 FPGA tool-set, including 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.



- ▶ 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 FPGA tool-set, including 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.

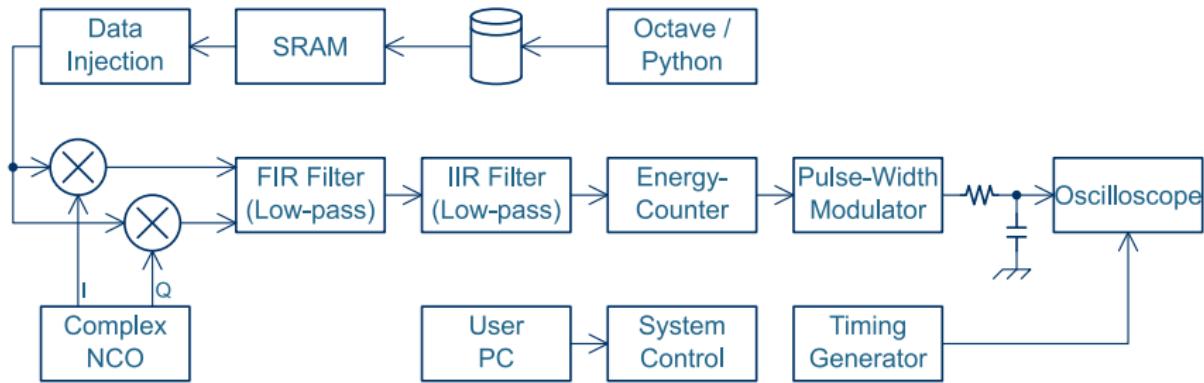


- ▶ EEE5117Z (Days 1 to 3)
 - ▶ Understand the underlying physical architecture of FPGAs
 - ▶ Understand the concept of timing constraints, clock domains and other timing-related issues
 - ▶ Use the FPGA tool-set, including JTAG debugging and the general Verilog-based compilation process
- ▶ EEE5118Z (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
 - ▶ Analyse timing closure issues and solve the problem such that the final design meets all timing requirements.



Practicals

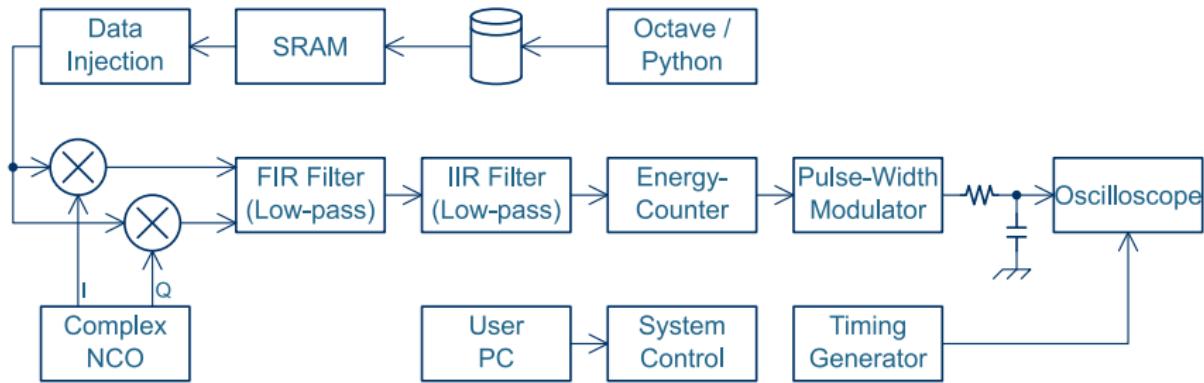
5 of 49



- ▶ UART interfacing to / from the computer
- ▶ 16-bit, 12.5 MSps data injection
- ▶ 2048-point FIR-filter with a 256 subsampling rate

Practicals

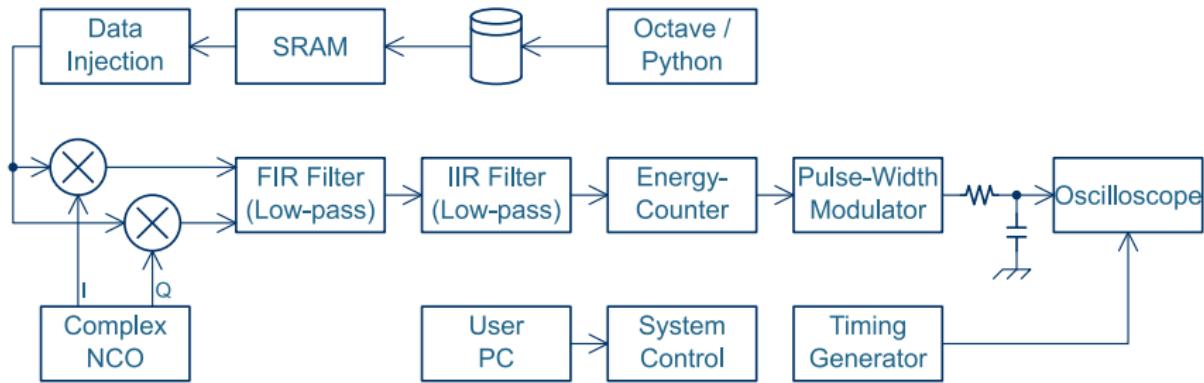
5 of 49



- ▶ UART interfacing to / from the computer
- ▶ 16-bit, 12.5 MSps data injection
- ▶ 2048-point FIR-filter with a 256 subsampling rate

Practicals

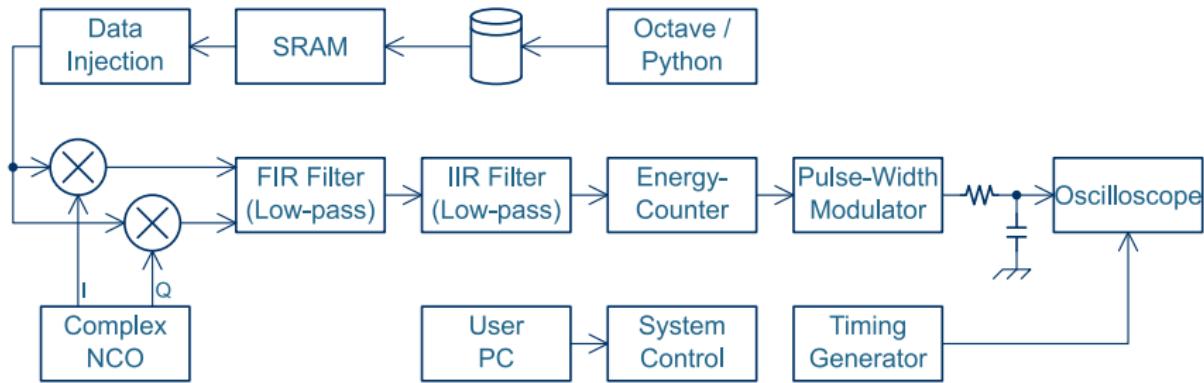
5 of 49



- ▶ UART interfacing to / from the computer
- ▶ 16-bit, 12.5 MSps data injection
- ▶ 2048-point FIR-filter with a 256 subsampling rate

Practicals

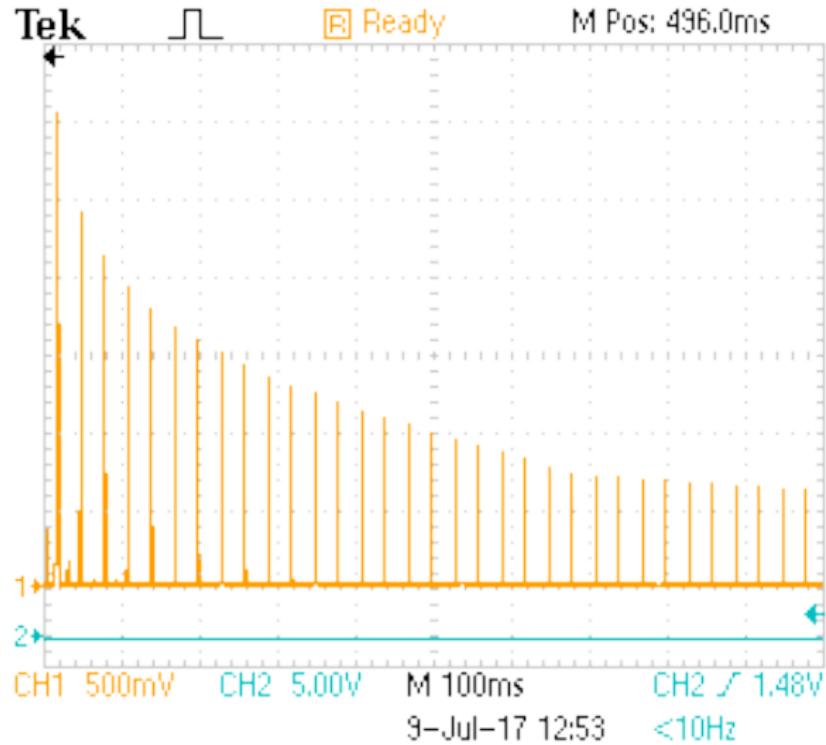
5 of 49



- ▶ UART interfacing to / from the computer
- ▶ 16-bit, 12.5 MSps data injection
- ▶ 2048-point FIR-filter with a 256 subsampling rate

Practicals

5 of 49



- ▶ 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?



- ▶ 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?



- ▶ 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?



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

- ▶ 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?

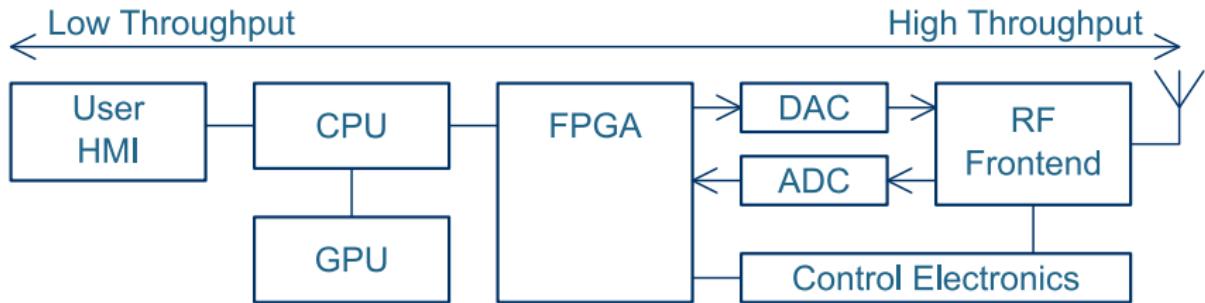
- ▶ Programming experience?
 - ▶ C / C++ / C#?
 - ▶ Embedded? PC?
 - ▶ GPU? OpenGL? OpenCL? DirectX? CUDA?
- ▶ FPGA Experience?
 - ▶ VHDL? Verilog? PyHDL? Migen? Graphical tools?
 - ▶ Synthesis? Simulation only?
 - ▶ What context / application?

- ▶ 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?

- ▶ 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

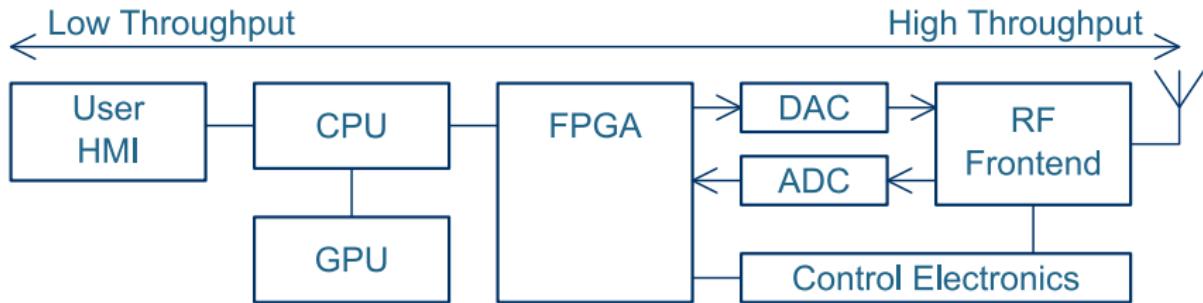
7 of 49



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

Software-Defined Radio

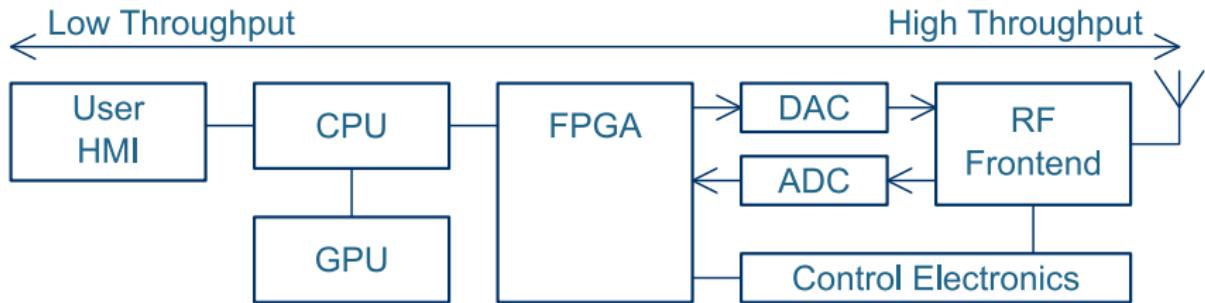
7 of 49



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

Software-Defined Radio

7 of 49



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

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



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

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

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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 coarse-grained parallel algorithms
 - ▶ Medium development cycle
 - ▶ Memory bandwidth problems
- ▶ FPGA:

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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:

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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:

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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:

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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



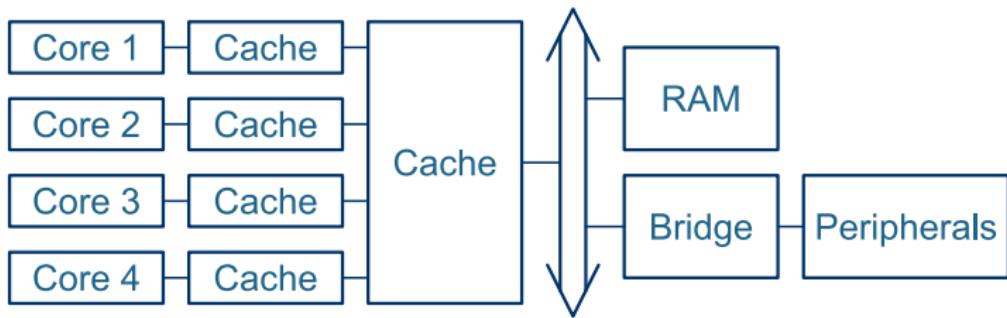
- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ Rich set of matured libraries and APIs
 - ▶ Does not handle parallel algorithms particularly well
- ▶ GPU:
 - ▶ Agile: can be reprogrammed in real-time
 - ▶ Extremely good at course-grained parallel algorithms
 - ▶ 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

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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

- ▶ CPU:
 - ▶ General-purpose; Easily modified; Short development cycle
 - ▶ Extensive history ⇒ 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

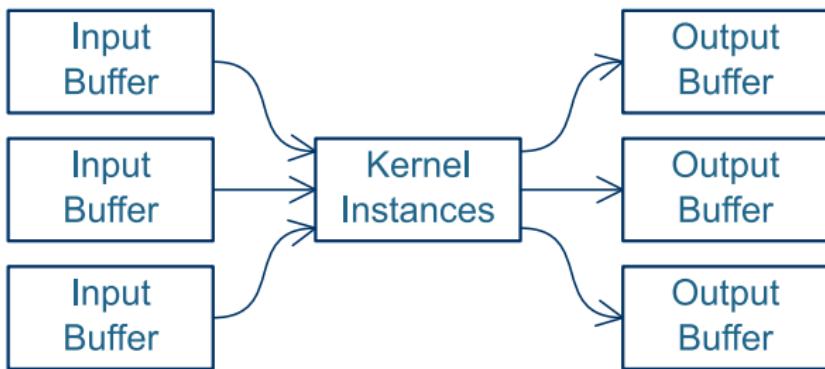
9 of 49



- ▶ 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

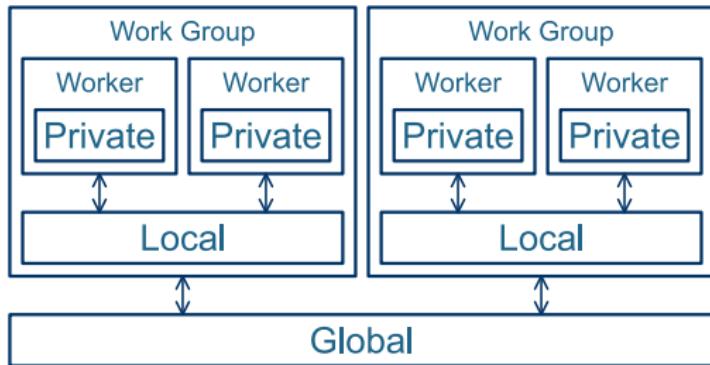
9 of 49



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

Programming Model – GPU

9 of 49

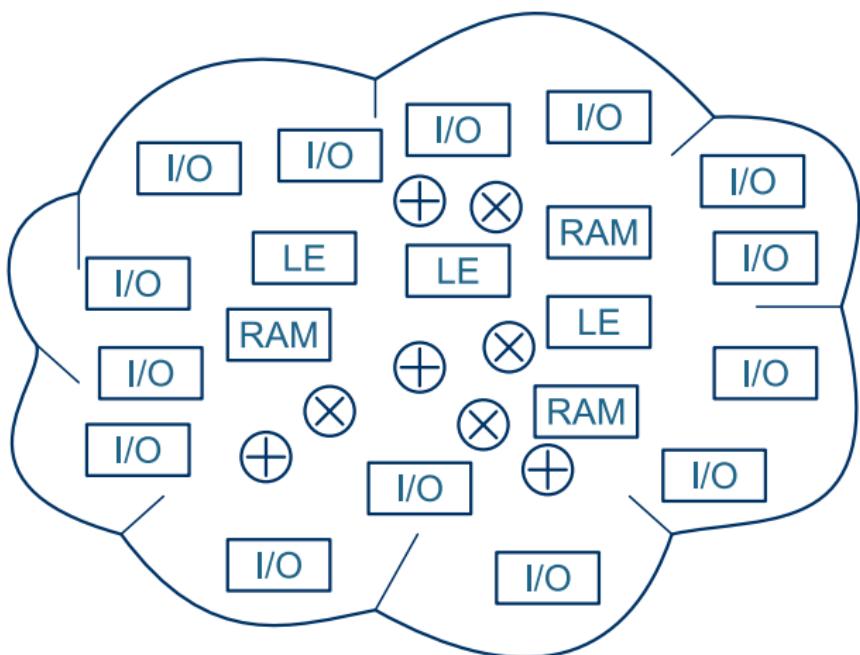


- ▶ The hardware is organised into groups of processors
- ▶ Within a group, execution is in lock-step
- ▶ Execution within one group is independent of other groups
- ▶ Three levels of memory

Programming Model – FPGA

9 of 49

- ▶ Any architecture you like...



Outline

Introduction

FPGA Internals

The Development Kit

Development Cycle

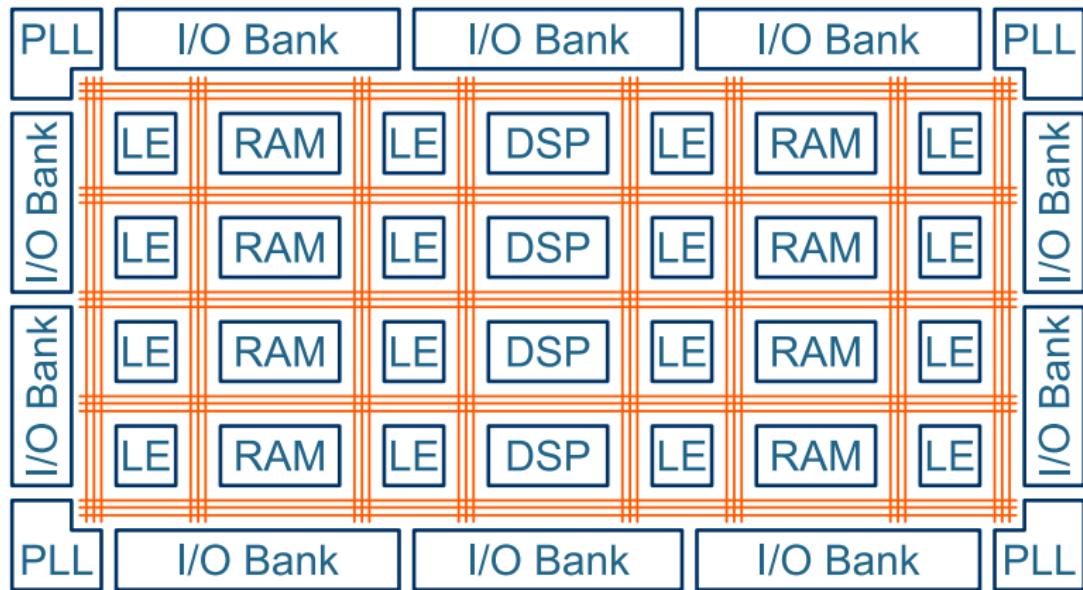
Verilog Basics

Simulation



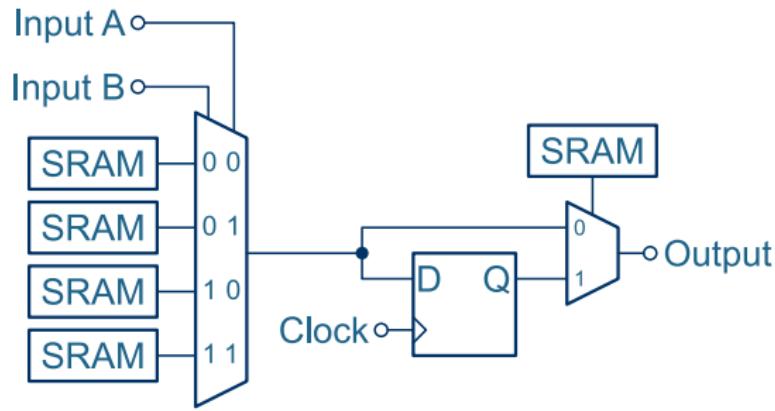
FPGA Architecture Overview

10 of 49



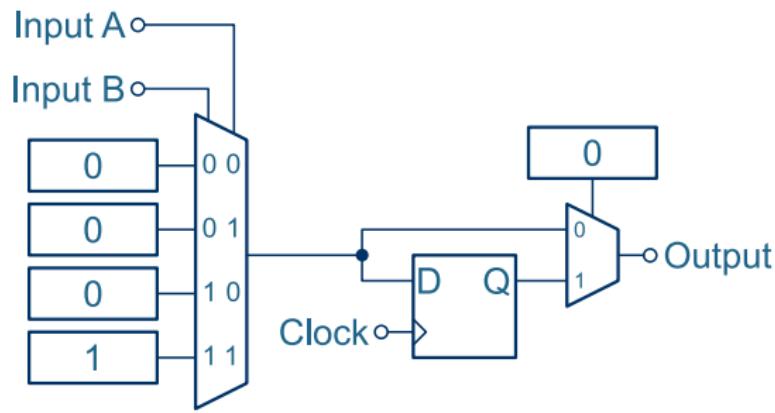
Logic Elements

11 of 49



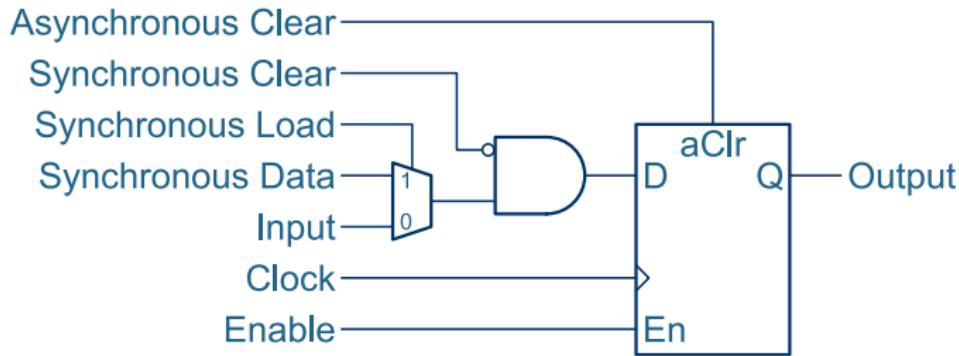
Logic Elements

11 of 49



Register Detail

12 of 49



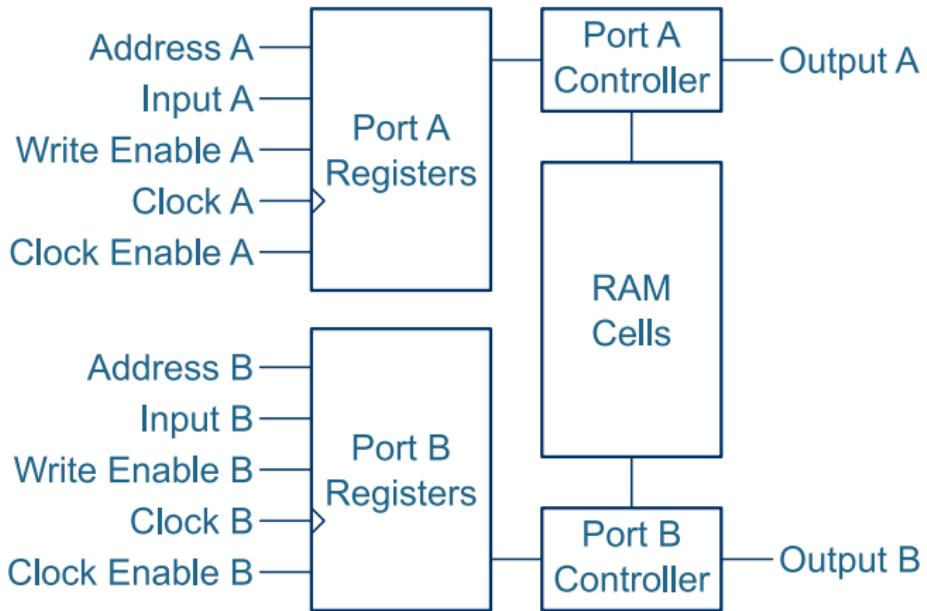
Register Detail

12 of 49

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

Internal RAM Blocks

13 of 49



Internal RAM Blocks

14 of 49

- ▶ 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



- ▶ 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



- ▶ 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

15 of 49

- ▶ 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
00 00 7E 81 A5 81 81 BD 99 81 81 7E 00 00 00 00 00
00 00 7E FF DB FF FF C3 E7 FF FF 7E 00 00 00 00 00
...
00 70 D8 30 60 C8 F8 00 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 00 00 00 FF FF 00
```



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

```
-- Header Comments
WIDTH=8;
DEPTH=4096;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT BEGIN
    [000..011] : 00;
    012        : 7E;
    013        : 81;
    ...
    [FFD..FFE] : FF;
    FFF        : 00;
END;
```

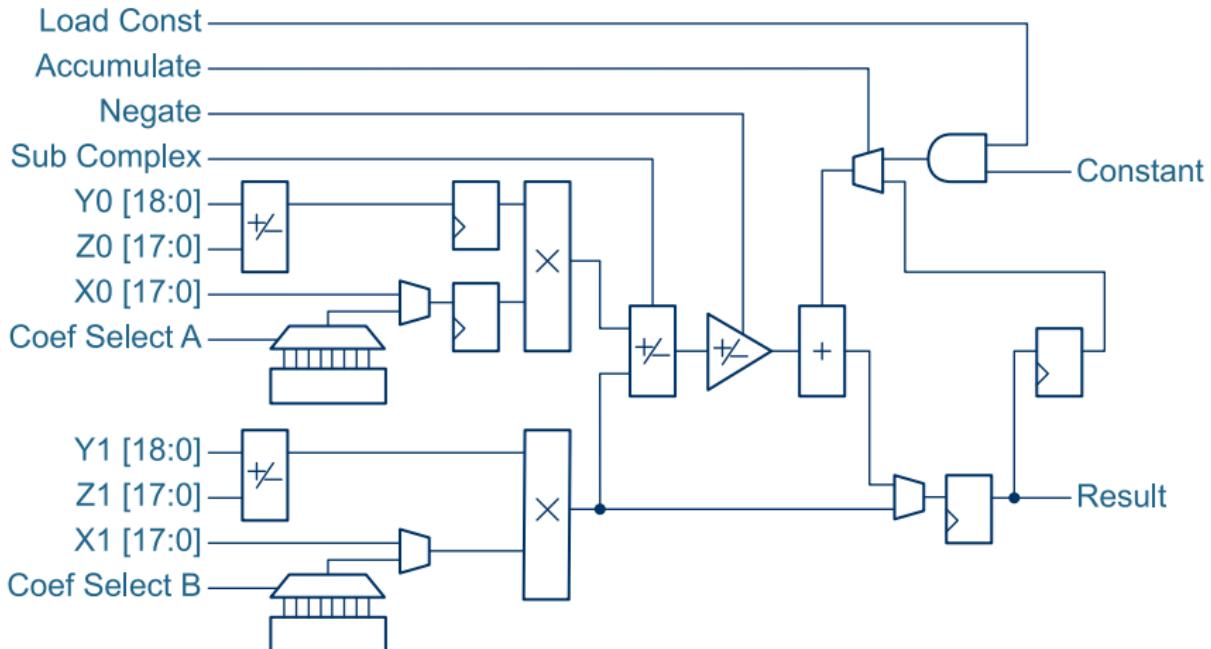


- ▶ Internal RAM can be initialised by means of a memory initialisation file (.mem)
- ▶ Altera (Intel) uses MIF files
- ▶ Xilinx (AMD) uses COE files

```
MEMORY_INITIALIZATION_RADIX=16;  
MEMORY_INITIALIZATION_VECTOR=  
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, 7E, FF, DB, FF, FF, C3, E7, FF, FF, 7E, 00, 00, 00,  
...  
00, 70, D8, 30, 60, C8, F8, 00, 00, 00, 00, 00, 00, 00, 00,  
00, 00, 00, 00, 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;
```

DSP Blocks (Altera / Intel)

16 of 49



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



- ▶ The Lattice XP2 have similar DSP blocks
- ▶ Each DSP block can be configured as
 - ▶ Eight 9×9 -bit multipliers
 - ▶ Four 18×18 -bit multipliers
 - ▶ One 36×36 -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

- ▶ The Lattice XP2 have similar DSP blocks
- ▶ Each DSP block can be configured as
 - ▶ Eight 9×9 -bit multipliers
 - ▶ Four 18×18 -bit multipliers
 - ▶ One 36×36 -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

- ▶ The Lattice XP2 have similar DSP blocks
- ▶ Each DSP block can be configured as
 - ▶ Eight 9×9 -bit multipliers
 - ▶ Four 18×18 -bit multipliers
 - ▶ One 36×36 -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

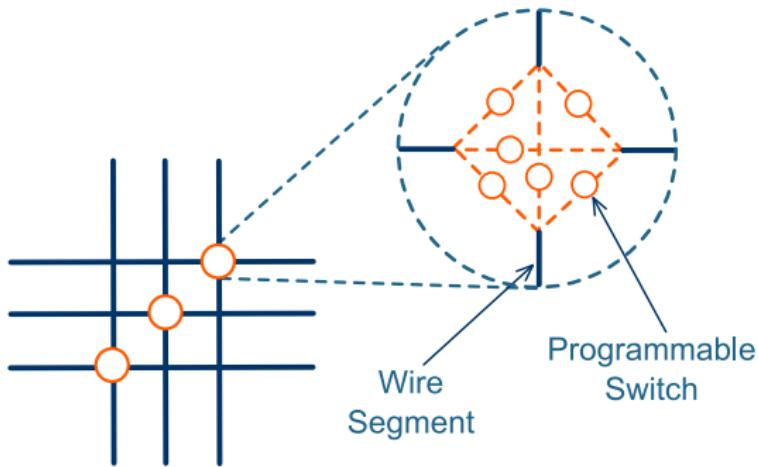
- ▶ The Lattice XP2 have similar DSP blocks
- ▶ Each DSP block can be configured as
 - ▶ Eight 9×9 -bit multipliers
 - ▶ Four 18×18 -bit multipliers
 - ▶ One 36×36 -bit multiplier
- ▶ Each input can be configured as signed or unsigned
- ▶ Includes an accumulate function
- ▶ The inputs and outputs can optionally be registered inside the multiplier block, which improves timing

- ▶ The Lattice XP2 have similar DSP blocks
- ▶ Each DSP block can be configured as
 - ▶ Eight 9×9 -bit multipliers
 - ▶ Four 18×18 -bit multipliers
 - ▶ One 36×36 -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

18 of 49

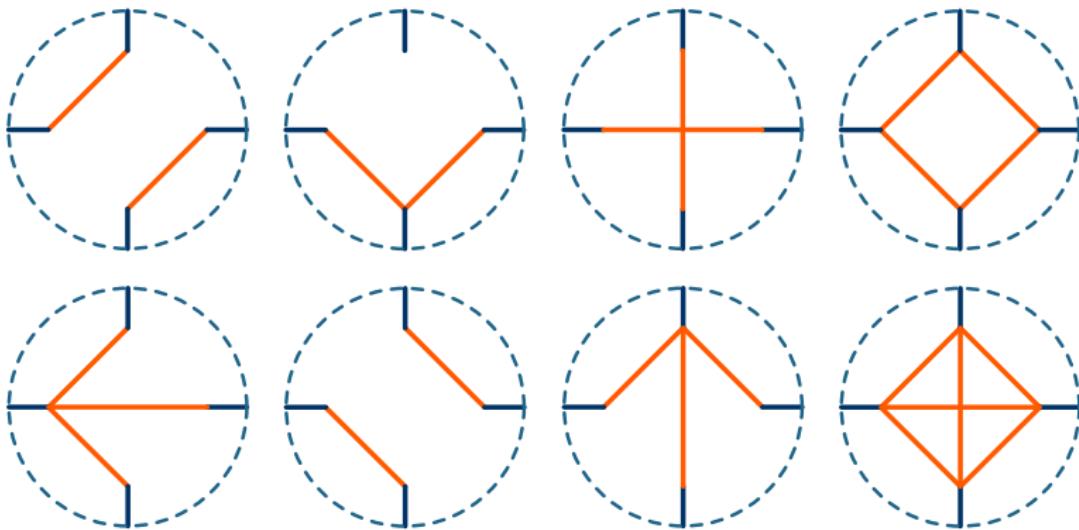
- ▶ Each interconnect crossing contains 6 switches
- ▶ These switches can be configured in various ways



Interconnect

18 of 49

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



Outline

Introduction

FPGA Internals

The Development Kit

Development Cycle

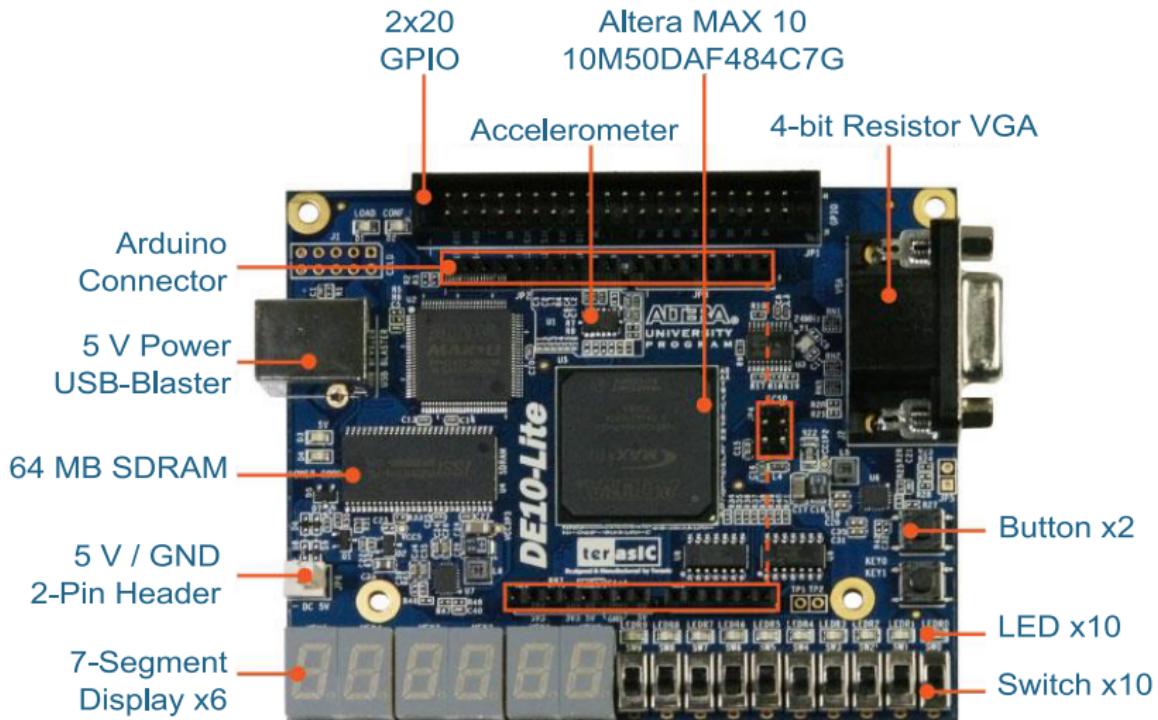
Verilog Basics

Simulation



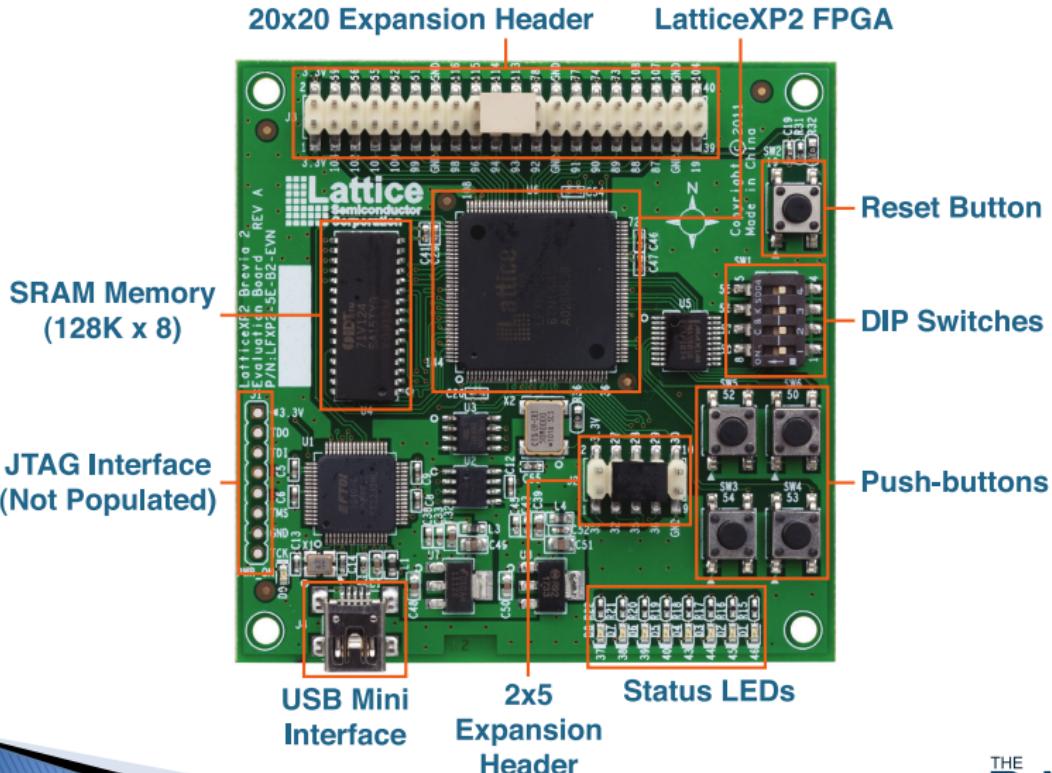
DE10 Lite Overview

19 of 49



LatticeXP2 Brevia2 Overview

20 of 49



- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18-bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18-bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18-bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18 -bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18 -bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18 -bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

- ▶ LatticeXP2 series (LFXP2-5E-6TN144I)
- ▶ 4 752 logic elements
- ▶ 9 EBR memory blocks (i.e. 18 432 kiB)
- ▶ 3 DSP blocks (i.e. twelve 18×18 -bit multipliers)
- ▶ On-chip configuration flash memory
- ▶ 2 phase-locked loop blocks
- ▶ 100 I/O Pins (40 on headers)

Coffee Break...

22 of 49



Outline

Introduction

FPGA Internals

The Development Kit

Development Cycle

Verilog Basics

Simulation



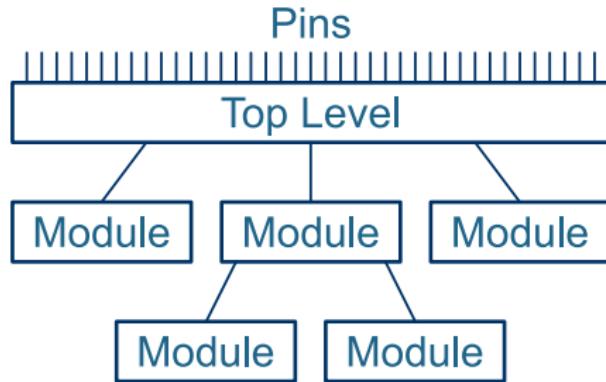
Development Cycle

23 of 49



Design Entry – Modular Design

24 of 49



Design Entry – HDL

25 of 49

- ▶ 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
```



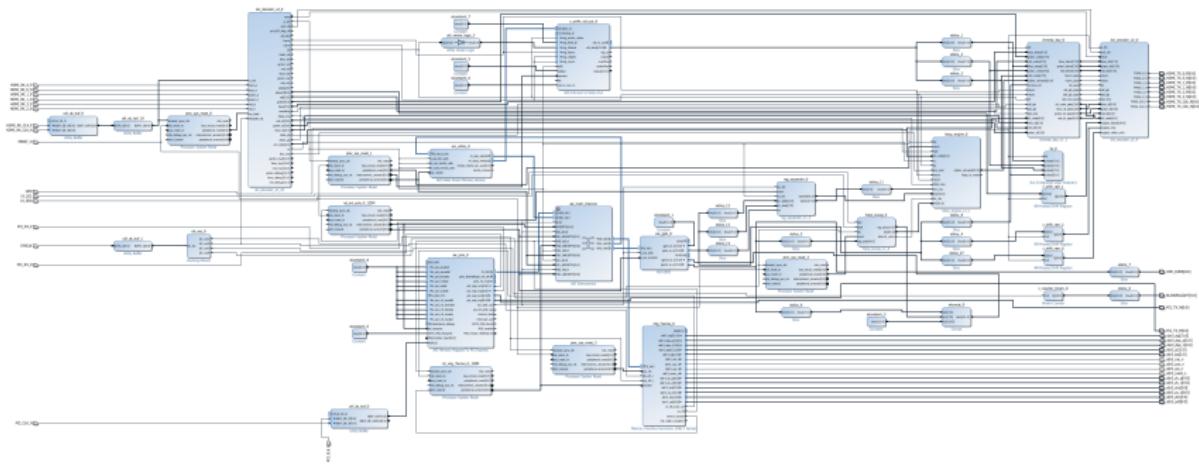
Design Entry – Schematic

26 of 49



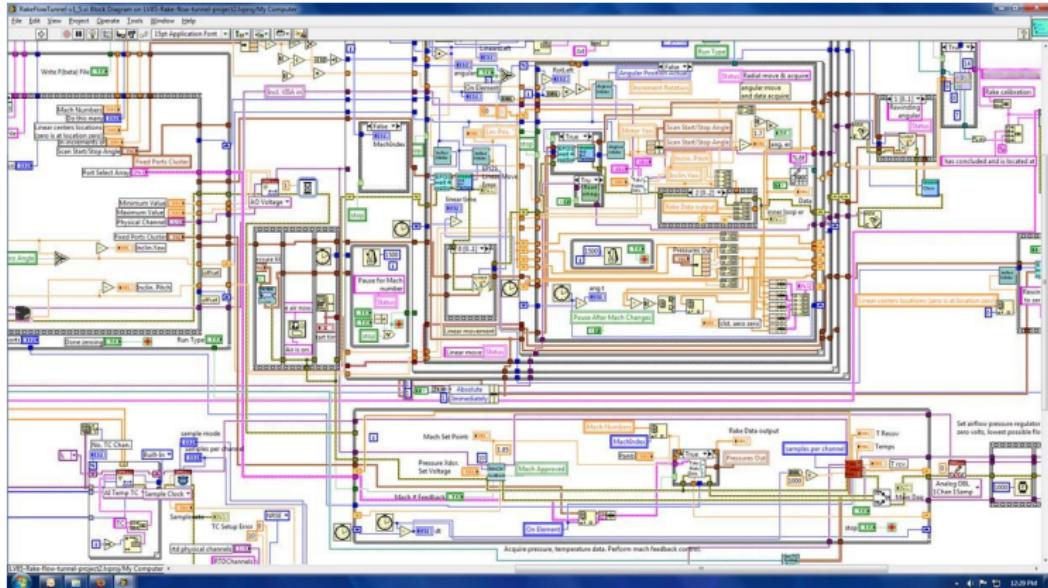
Design Entry – Schematic

26 of 49



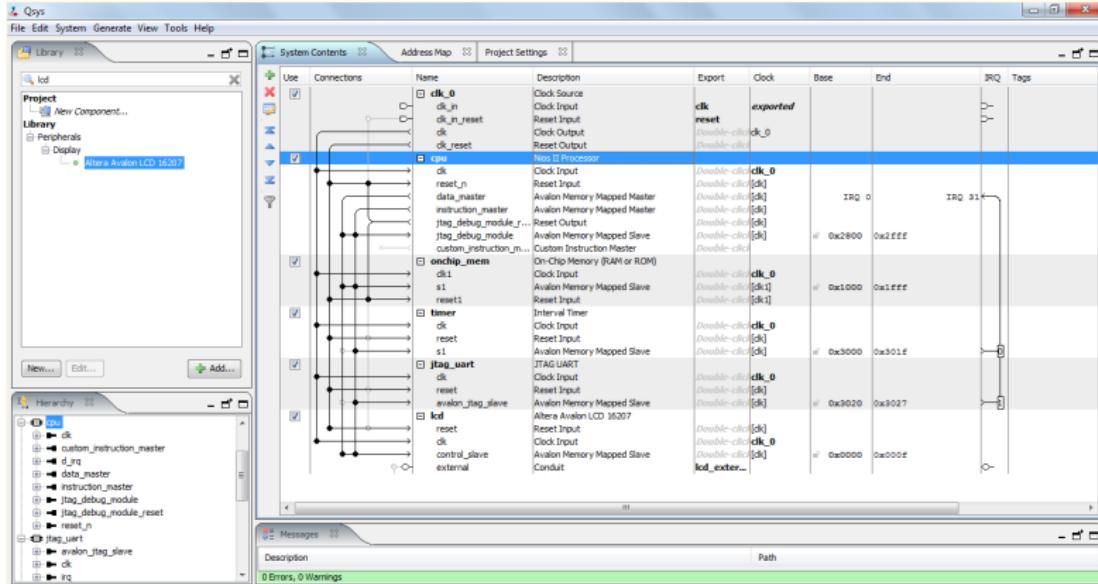
Design Entry – Schematic

26 of 49



Design Entry – Qsys

27 of 49



Design Entry – Qsys

27 of 49

The screenshot shows the Qsys interface with a toolbar on the left containing icons for add (+), delete (X), edit (pencil), and other functions. The main area has two panes: a connection diagram on the left and a table on the right.

Connections Diagram: Shows the physical connections between components. A 'clk_0' component is connected to a 'cpu' component. The 'clk_0' component has outputs 'clk_in' and 'clk_in_reset' which connect to the 'clk' and 'reset_n' inputs of the 'cpu'. The 'cpu' component has an output 'clk' which connects back to the 'clk_in' input of the 'clk_0' component. The 'cpu' component also has an output 'clk_reset' which connects to the 'clk_in_reset' input of the 'clk_0' component.

Table:

Use	Connections	Name	Description
<input checked="" type="checkbox"/>		clk_0	Clock Source
		clk_in	Clock Input
		clk_in_reset	Reset Input
		clk	Clock Output
		clk_reset	Reset Output
<input checked="" type="checkbox"/>		cpu	Nios II Processor
		clk	Clock Input
		reset_n	Reset Input
		data_master	Avalon Memory Mapped Master
		instruction_master	Avalon Memory Mapped Master
		jtag_debug_module_r...	Reset Output
		jtag_debug_module	Avalon Memory Mapped Slave
		custom_instruction_m...	Custom Instruction Master



Design Entry – HLS

28 of 49

```
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

29 of 49

- ▶ 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

29 of 49

- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.



- ▶ 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.

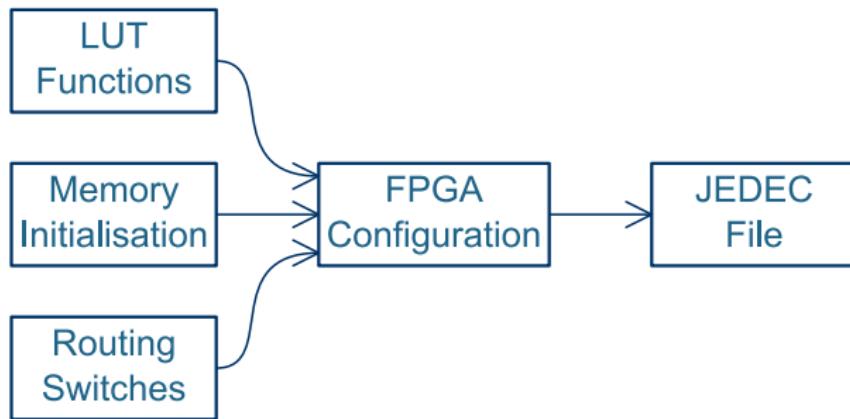


- ▶ The fitter performs a place and route function (similar to a PCB auto-router)
- ▶ This is a computationally-intensive process, so be patient
-
- ▶ Uses design constraints:
 - ▶ Clock rate
 - ▶ Combinational logic time delay
 - ▶ Multi-cycle timing requirements
 - ▶ False-path specifications
 - ▶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



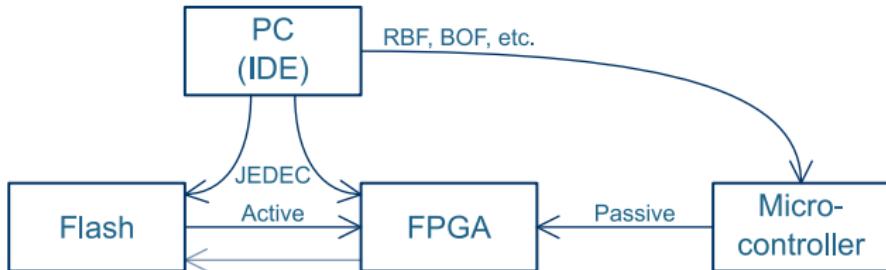
- ▶ The fitter performs a place and route function (similar to a PCB auto-router)
- ▶ This is a computationally-intensive process, so be patient
-
- ▶ Uses design constraints:
 - ▶ Clock rate
 - ▶ Combinational logic time delay
 - ▶ Multi-cycle timing requirements
 - ▶ False-path specifications
 - ▶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





Programming Architectures

32 of 49



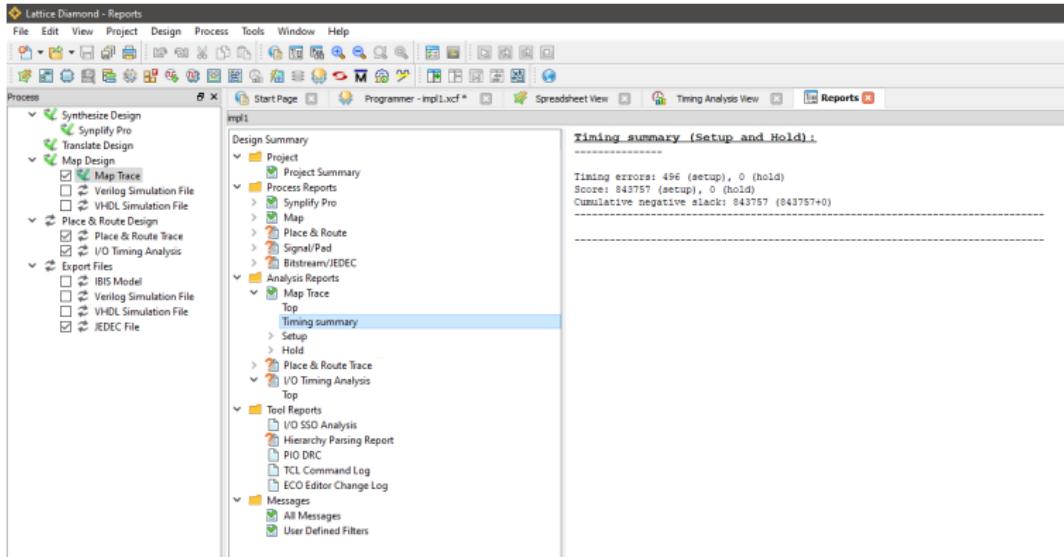
- ▶ JEDEC ⇒ Joint Electron Device Engineering Council
- ▶ RBF ⇒ Raw Binary File
- ▶ BOF ⇒ Borph Object File



Lattice Diamond

33 of 49

Practical 01 – Getting Started introduces the Lattice Diamond IDE.



Outline

Introduction

FPGA Internals

The Development Kit

Development Cycle

Verilog Basics

Simulation



Module Definition

34 of 49

```
/* 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 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 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
```

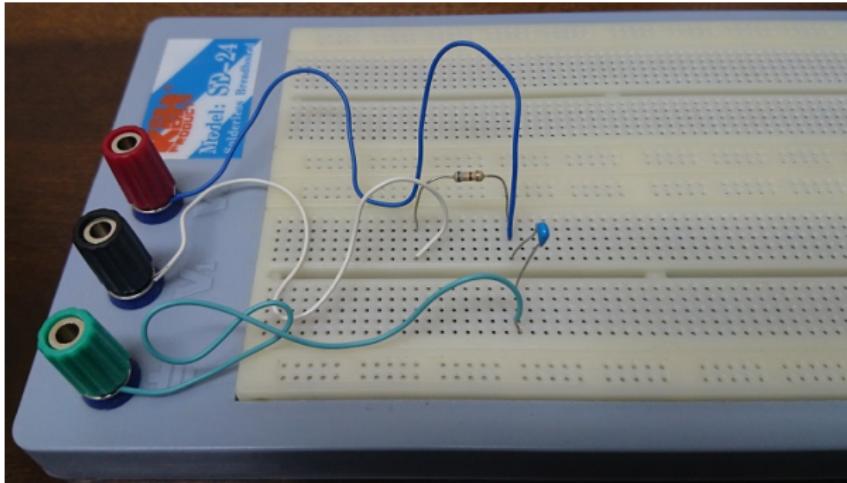


- ▶ Wires are internal connections
- ▶ See them as wires on a bread-board...

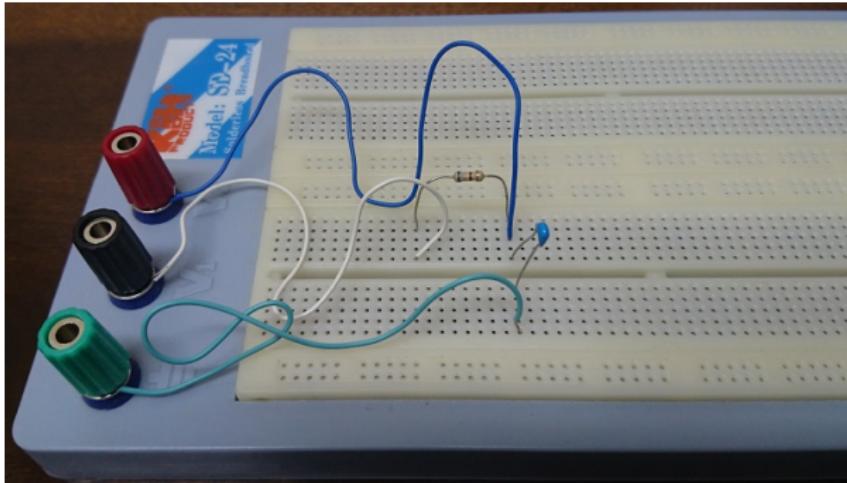
```
wire      A;          // A single-bit wire
wire [7:0]B;        // An 8-bit wire
wire [7:0]C[5:0]; // A 6-element array of 8-bit wires

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





```
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
```



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

Operators – Reduction

36 of 49

```
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

37 of 49

```
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

38 of 49

```
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

39 of 49

```
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

40 of 49

```
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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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

41 of 49

- ▶ 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);
```



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 an obvious warning!**
- ▶ Consult the compile warnings to check for unintended size-mismatches

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 an obvious warning!**
- ▶ Consult the compile warnings to check for unintended size-mismatches

1	0	1	1	11 or -5				
0	0	0	0	1	0	1	1	11

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 an obvious warning!
- ▶ Consult the compile warnings to check for unintended size-mismatches

0	1	1	1	1	0	1	1	123
1	0	1	1	1	1	1	1	11

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 an obvious warning!**
- ▶ Consult the compile warnings to check for unintended size-mismatches

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 an obvious warning!**
- ▶ Consult the compile warnings to check for unintended size-mismatches

```
// All underscores "_" are ignored
// (except in Lattice: leading "_" gives an error)
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

44 of 49

```
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

44 of 49

```
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

Introduction

FPGA Internals

The Development Kit

Development Cycle

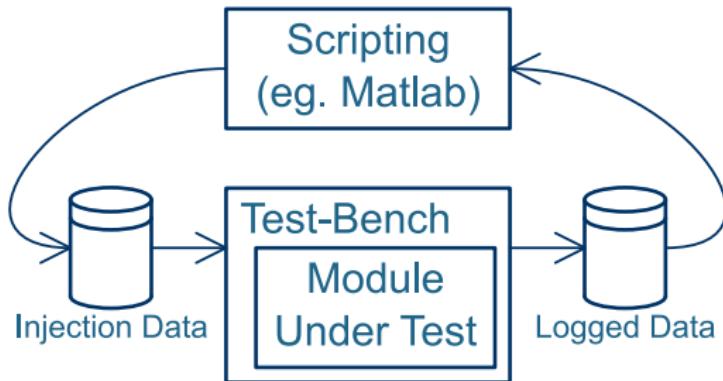
Verilog Basics

Simulation



Simulation Basics

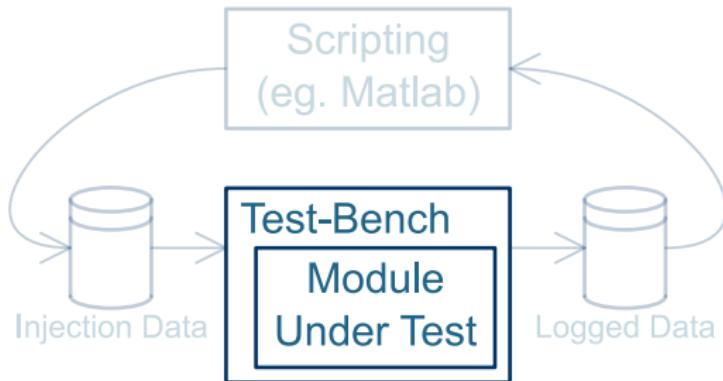
45 of 49



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

Simulation Basics

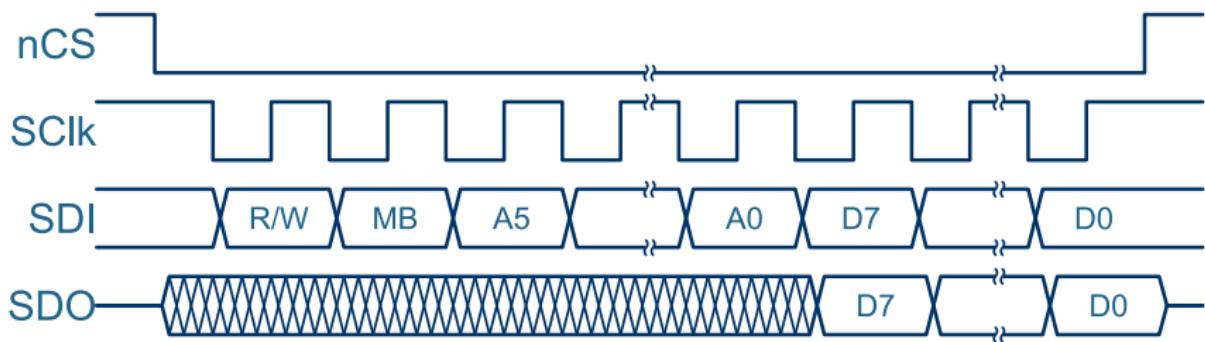
45 of 49



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

ADXL345 SPI Interface

46 of 49



Test Bench Basics

47 of 49

```
`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

47 of 49

```
// 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

47 of 49

```
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
end
```



Test Bench Basics

47 of 49

```
// 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
```



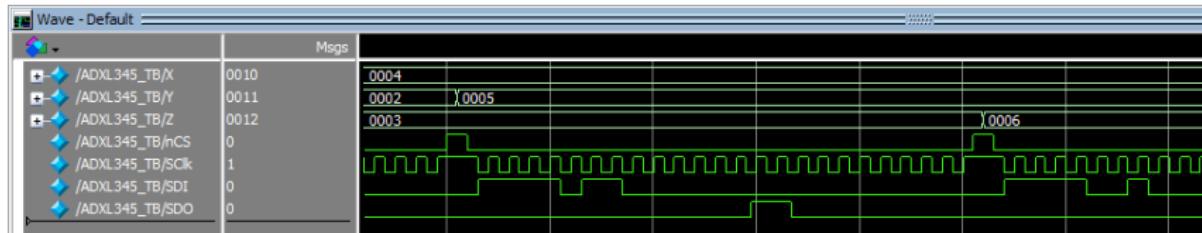
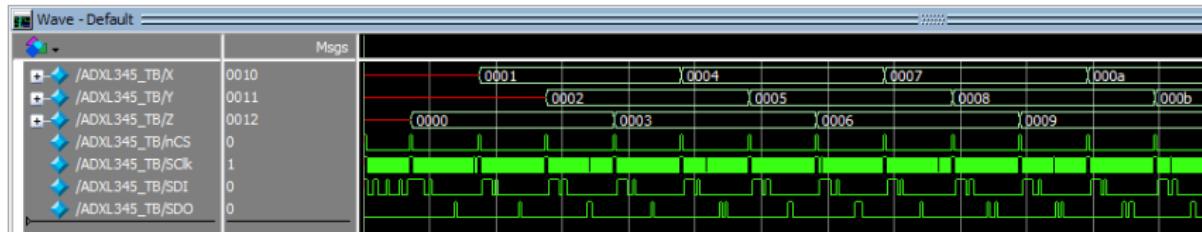
Test Bench Basics

47 of 49

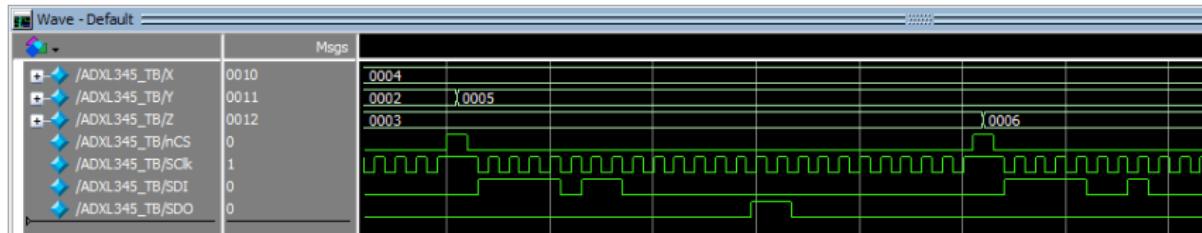
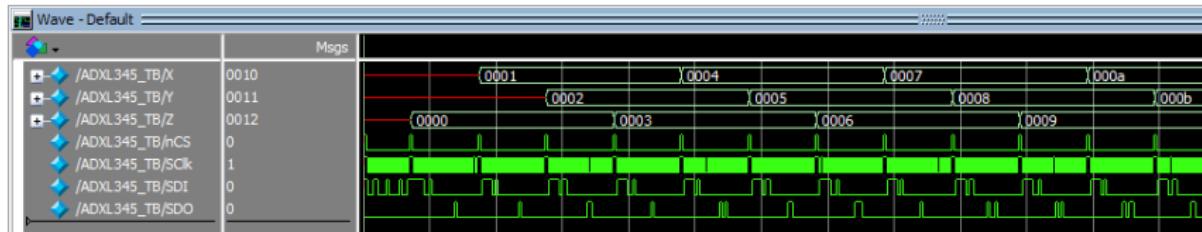
```
@ (posedge nCS) ;  
  
    DataOut = DataOut + 1;  
end  
  
endmodule
```



The resulting Modelsim waveforms:



The resulting Modelsim waveforms:



Practical 02 – Simulation introduces
the Modelsim simulation tool.

Select References

49 of 49

-  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