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 3 – 29 April 2022

Outline

Pipelines

Streaming Processors

Memory-Mapped Bus

Pulse-width Modulation

High Resolution PWM

Outline

Pipelines

Streaming Processors

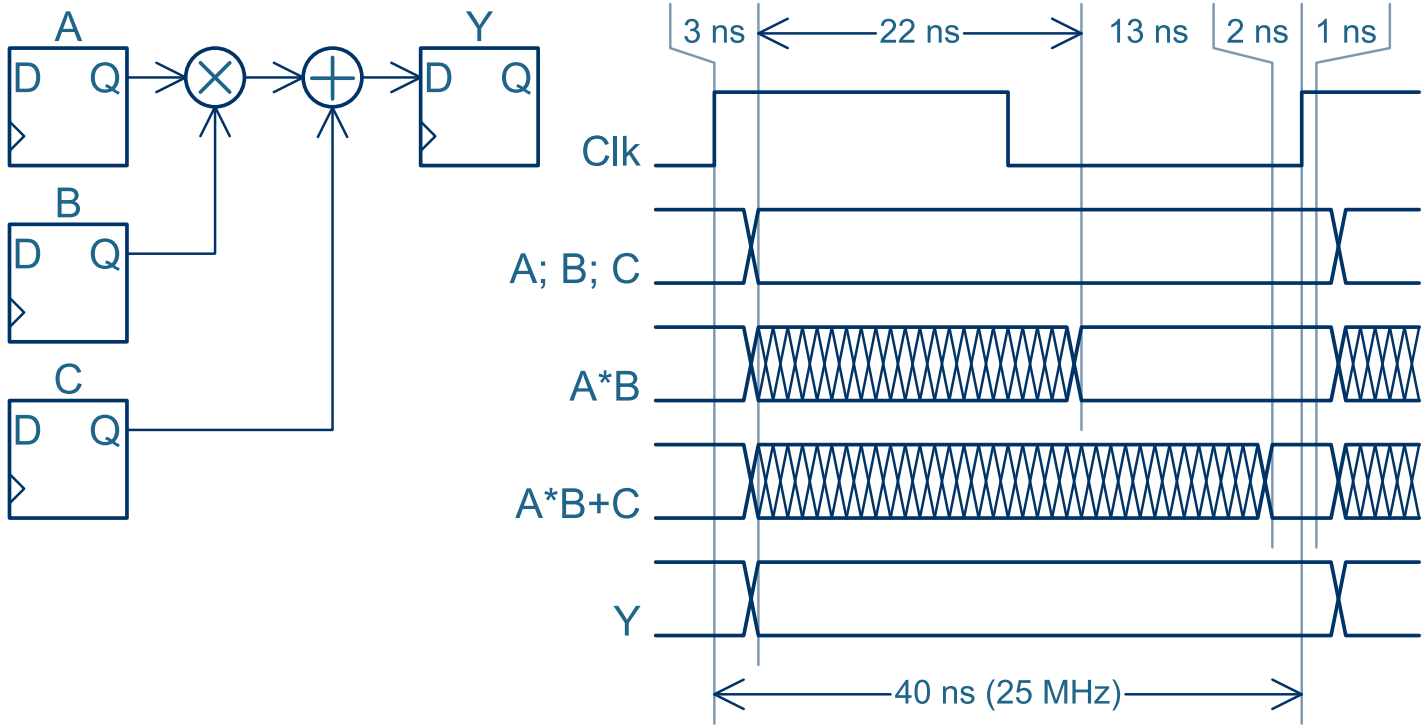
Memory-Mapped Bus

Pulse-width Modulation

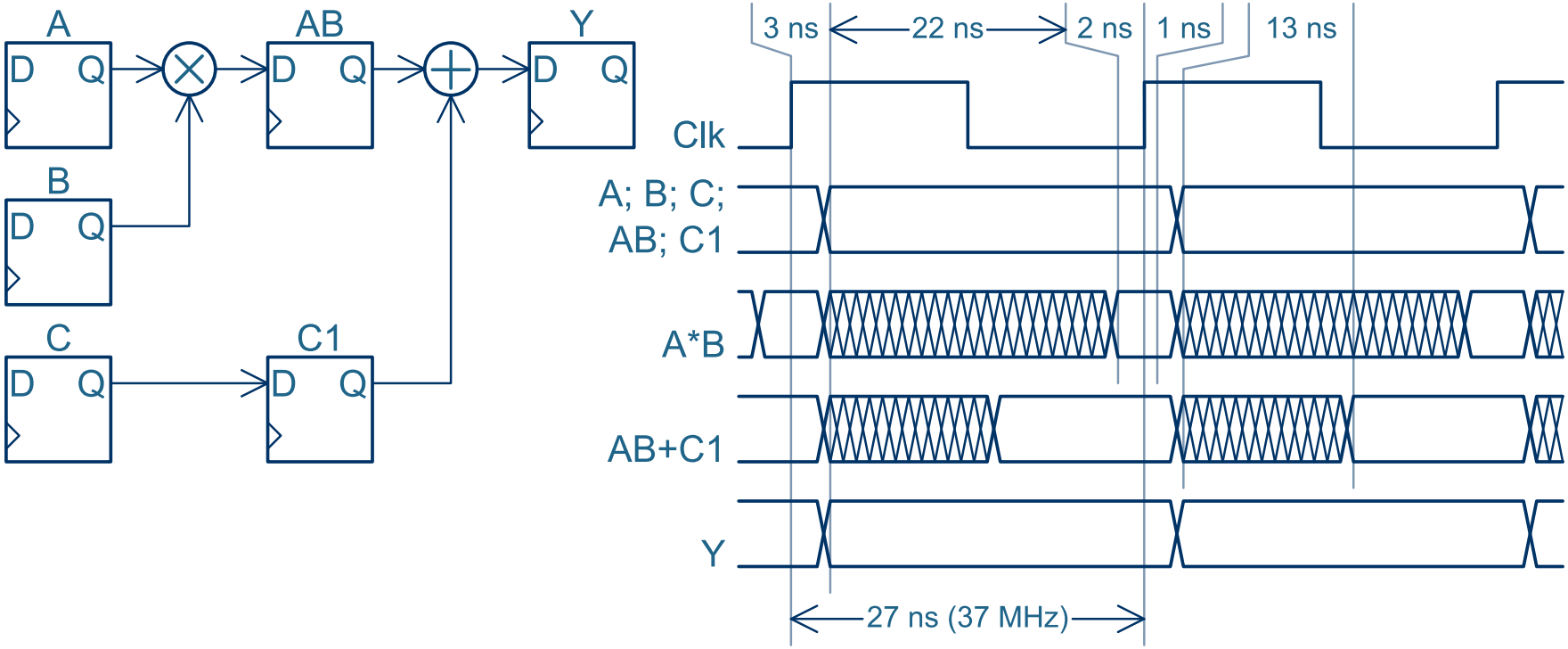
High Resolution PWM

THE RADAR MASTERS COURSE

Simple Pipelines



Simple Pipelines



Simple Pipelines

- Gain throughput at the cost of latency and resources
- Often easier to use arrays, especially with long chains
- Keep the index equal to the stage which assigns the value

```
always @(posedge ipClk) begin
  // Stage 1
  AB <= A*B;
  C1 <= C;

  // Stage 2
  Y <= AB + C1;
end
```

Simple Pipelines

- Gain throughput at the cost of latency and resources
- Often easier to use arrays, especially with long chains
- Keep the index equal to the stage which assigns the value

```
reg [7:0]C[1:0];

always @(posedge ipClk) begin
  // Stage 0
  C[0] <= C_Input;

  // Stage 1
  AB   <= A*B;
  C[1] <= C[0];

  // Stage 2
  Y <= AB + C[1];
end
```

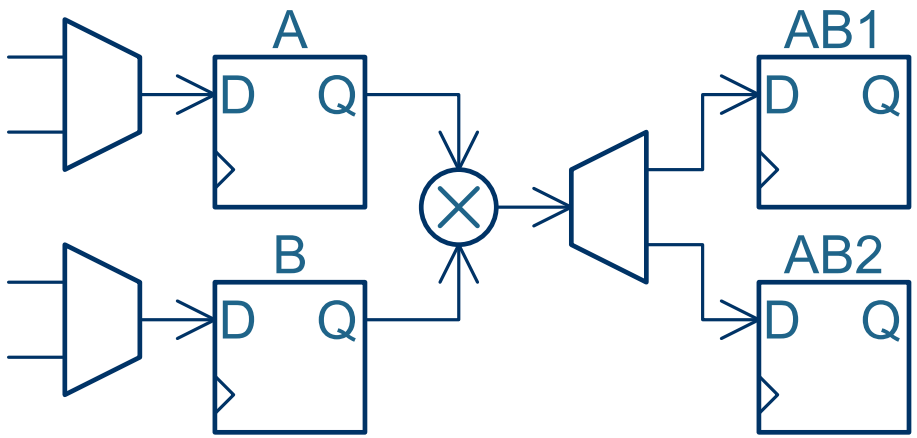
Simple Pipelines

- Gain throughput at the cost of latency and resources
- Often easier to use arrays, especially with long chains
- Keep the index equal to the stage which assigns the value

```
reg [7:0]C[1:0];

always @(posedge ipClk) begin
  // Stage 0
  C[0] <= C_Input;

  // Stage 1
  AB   <= A*B;
  C[1] <= C[0];

  // Stage 2
  Y <= AB + C[1];
end
```

Sharing Resources

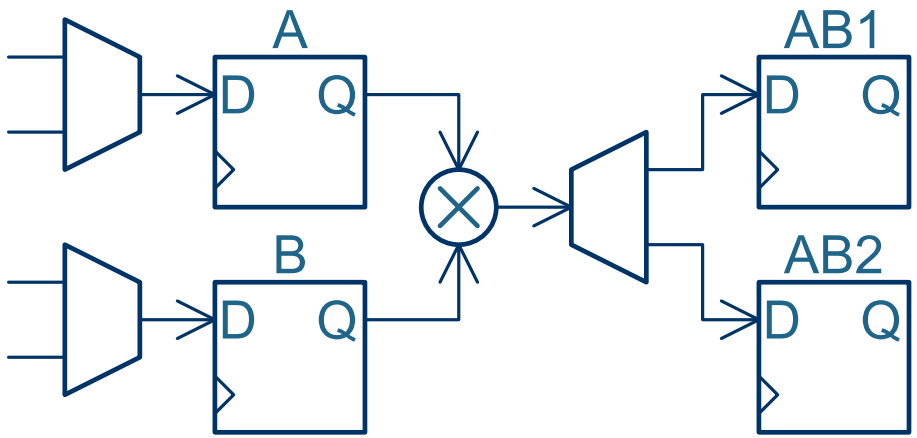

```
reg  [ 7:0]A, B;
wire [15:0]AB = A * B;

State1: begin
   AB2    <= AB;
   A      <= A1;
   B      <= B1;
   State  <= State2;
end

State2: begin
   AB1    <= AB;
   A      <= A2;
   B      <= B2;
   State  <= State1;
end
```



- Gain resource efficiency at the cost of throughput

Sharing Resources


```
reg  [ 7:0]A, B;
wire [15:0]AB = A * B;

State1: begin
   AB2    <= AB;
   A      <= A1;
   B      <= B1;
   State  <= State2;
end

State2: begin
   AB1    <= AB;
   A      <= A2;
   B      <= B2;
   State  <= State1;
end
```



- Gain resource efficiency at the cost of throughput

Outline

Pipelines

Streaming Processors

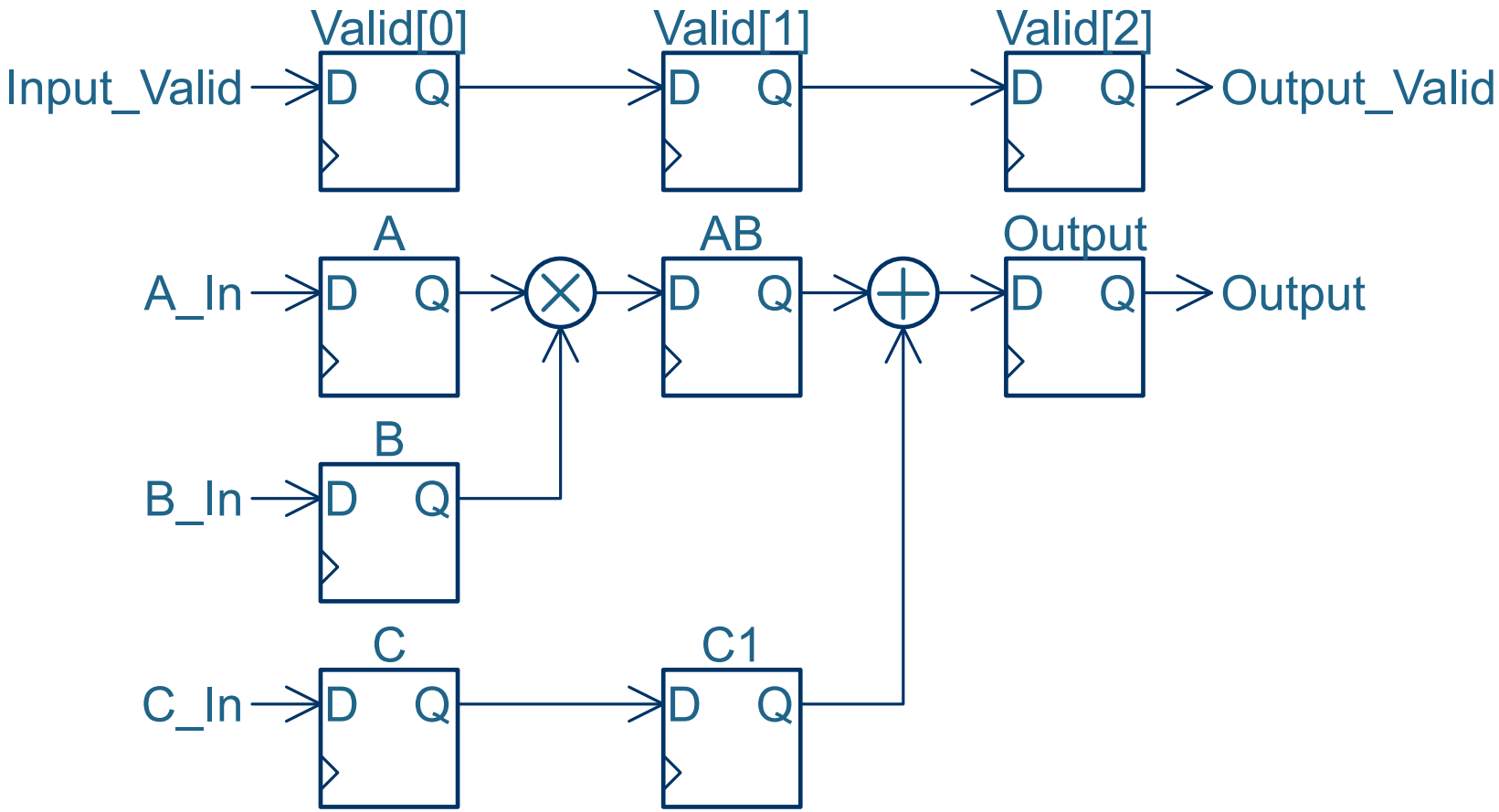
Memory-Mapped Bus

Pulse-width Modulation

High Resolution PWM

THE RADAR MASTERS COURSE

Stream Pipeline



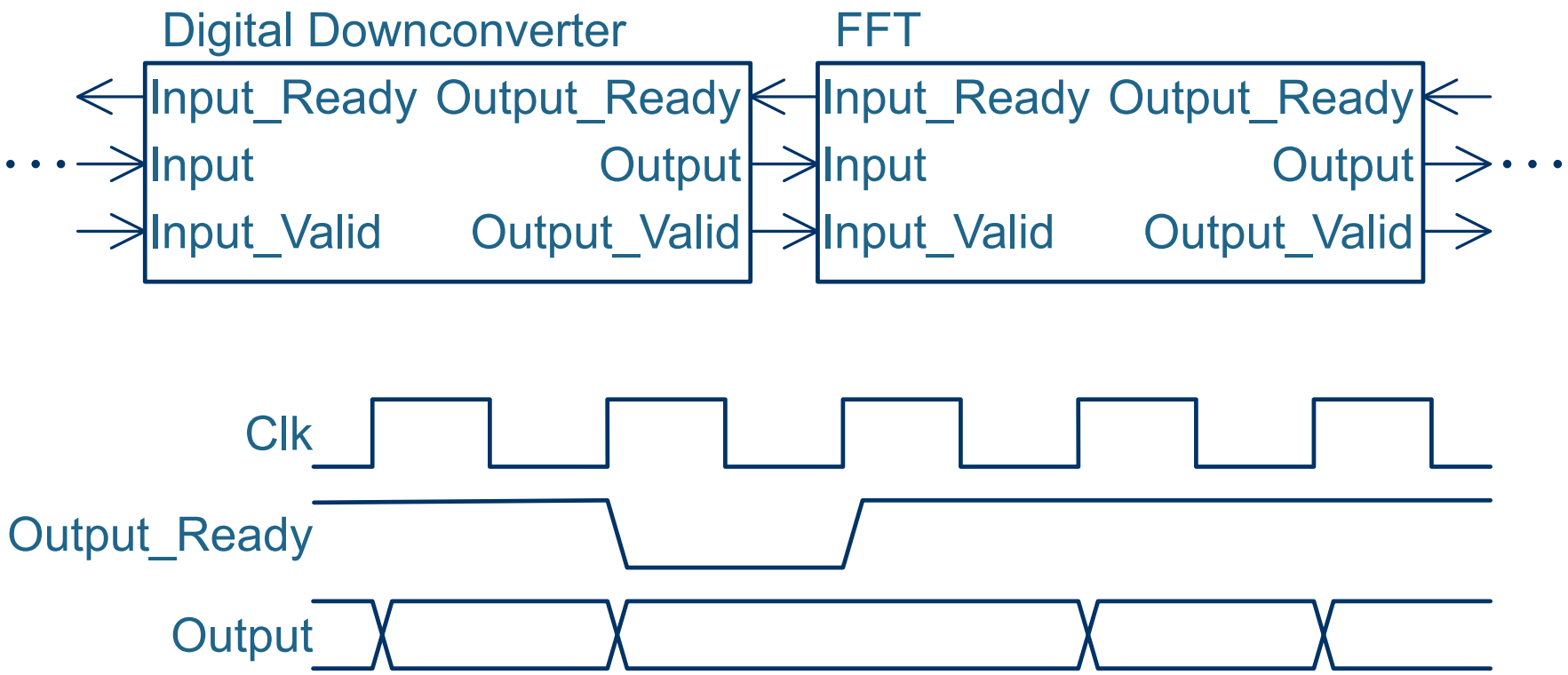
Stream Pipeline


```
reg [2:0]Valid;

always @(posedge ipClk) begin
  // Input
  A      <= A_In; B <= B_In; C <= C_In;
  Valid <= {Valid[1:0], Input_Valid};

  // Stage 1
  AB <= A*B;
  C1 <= C;

  // Stage 2
  Output <= AB + C1;
end

assign Output_Valid = Valid[2];
```

Back-pressure


```
reg [2:0]Valid;

always @(posedge ipClk) begin
  if(Output_Ready) begin
    // Input
    ...

    // Stage 1
    ...

    // Stage 2
    Output <= AB + C1;
  end
end

assign Input_Ready  = Output_Ready;
assign Output_Valid = Valid[2];
```

Streaming Processor



Handshaking Strategies

Different buses have different handshaking strategies:

- An Avalon source is allowed to wait for the sink to be ready before asserting the `Valid`.
- An AXI sink is allowed to wait for the source to be valid before asserting its `Ready`.
- A Wishbone slave must wait for valid data from the master before asserting its `Ack`.

Handshaking Strategies

Different buses have different handshaking strategies:

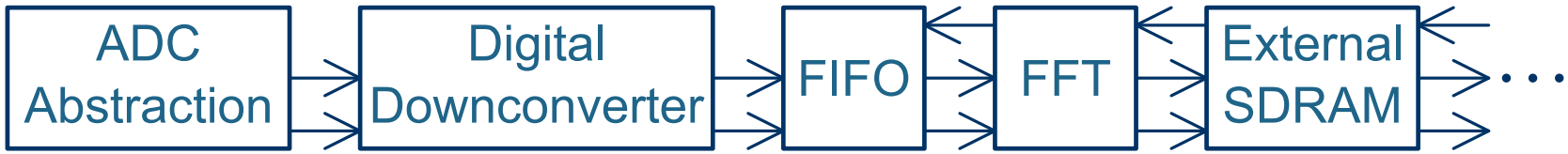
- An Avalon source is allowed to wait for the sink to be ready before asserting the `Valid`.
- An AXI sink is allowed to wait for the source to be valid before asserting its `Ready`.
- A Wishbone slave must wait for valid data from the master before asserting its `Ack`.

Handshaking Strategies

Different buses have different handshaking strategies:

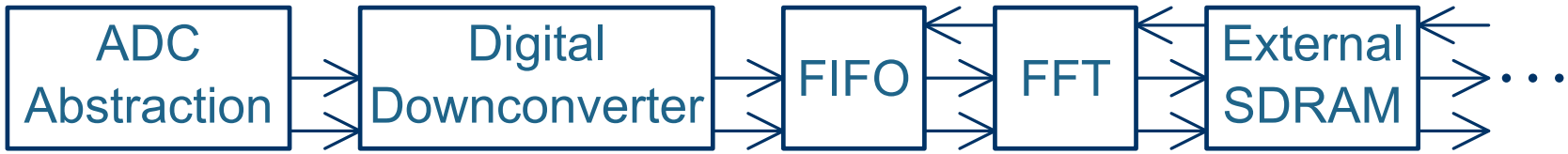
- An Avalon source is allowed to wait for the sink to be ready before asserting the `Valid`.
- An AXI sink is allowed to wait for the source to be valid before asserting its `Ready`.
- A Wishbone slave must wait for valid data from the master before asserting its `Ack`.

FIFO Queues



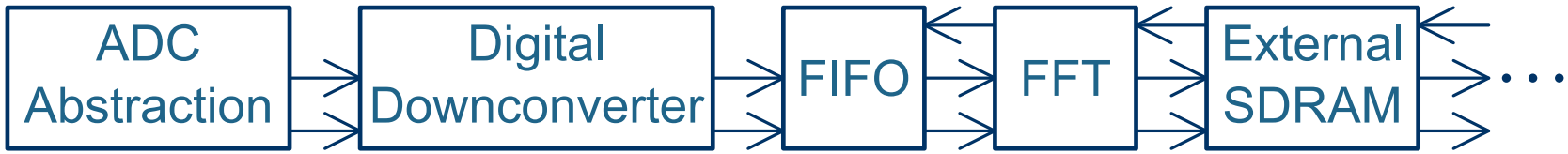
- Back-pressure makes it possible to move all the FIFO queues into a single queue
- Generally put the queue where the least amount of data is (saves resources)
- DSP chains have processing gain, so place the FIFO early in the chain

FIFO Queues



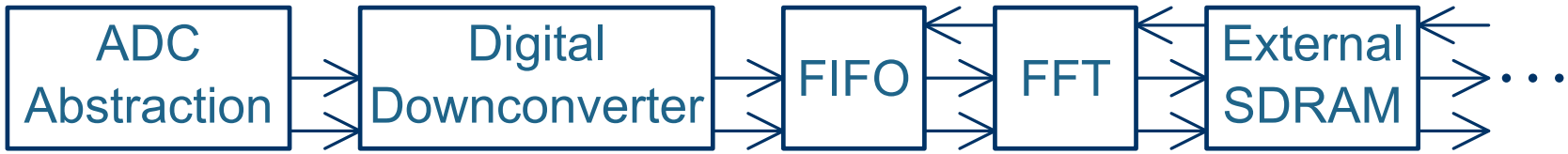
- Back-pressure makes it possible to move all the FIFO queues into a single queue
- Generally put the queue where the least amount of data is (saves resources)
- DSP chains have processing gain, so place the FIFO early in the chain

FIFO Queues



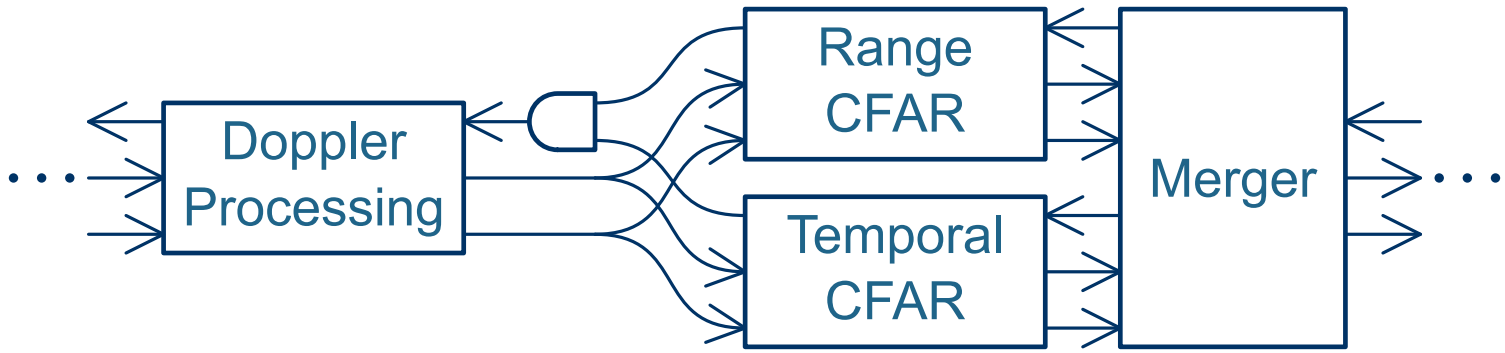
- Back-pressure makes it possible to move all the FIFO queues into a single queue
- Generally put the queue where the least amount of data is (saves resources)
- DSP chains have processing gain, so place the FIFO early in the chain

FIFO Queues



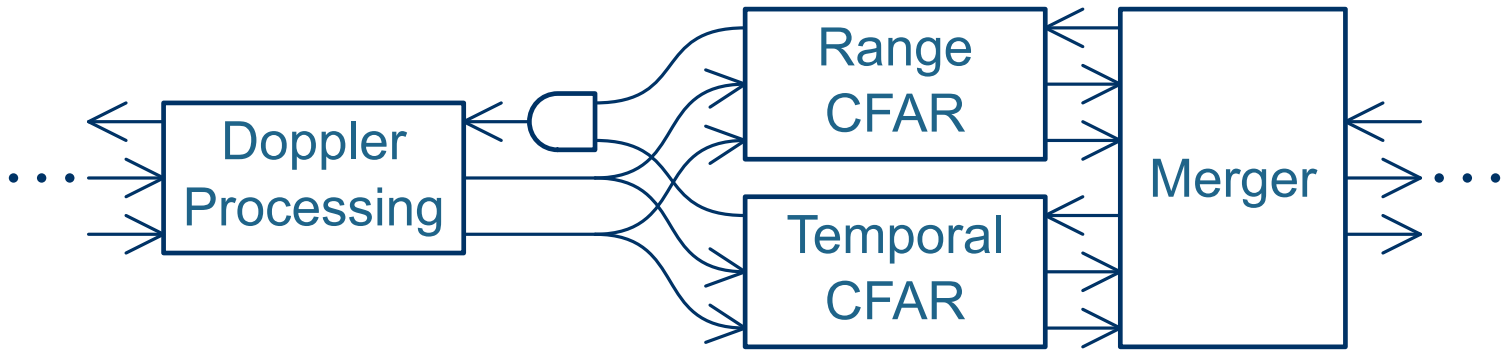
- Back-pressure makes it possible to move all the FIFO queues into a single queue
- Generally put the queue where the least amount of data is (saves resources)
- DSP chains have processing gain, so place the FIFO early in the chain

Complex Structures



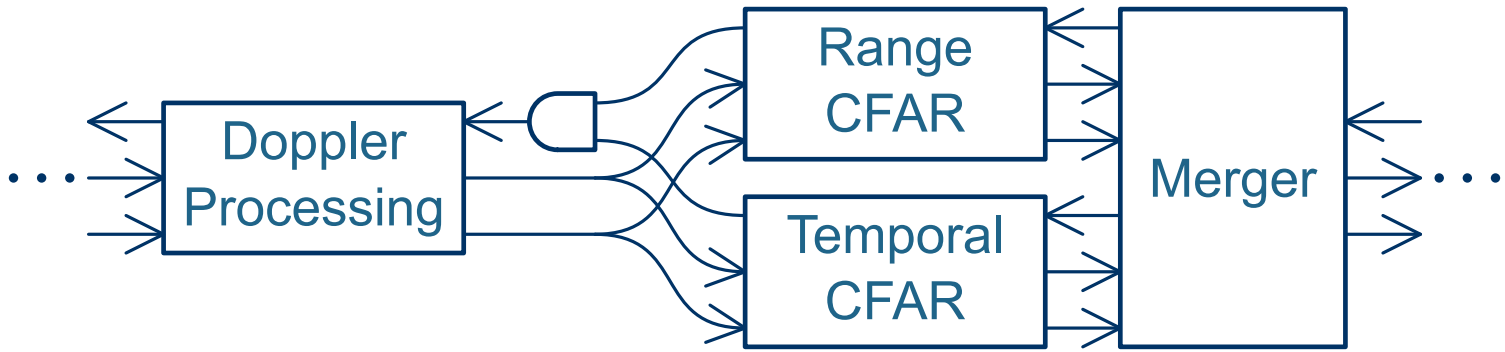
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:

Complex Structures



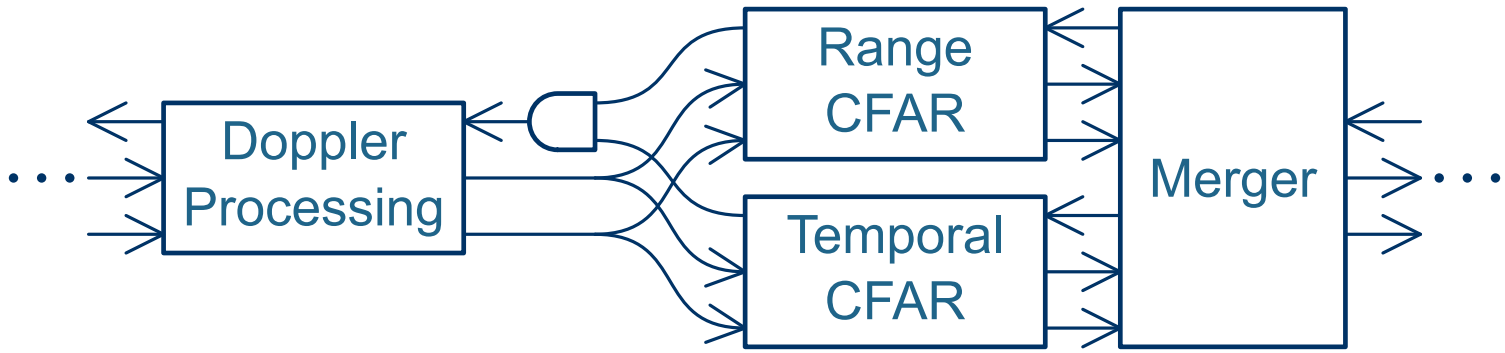
- Gate the “Valid” with the “Ready” (if either sink is not ready, both sinks must see a “not valid” input)
- The “Merger” can take various forms:

Complex Structures



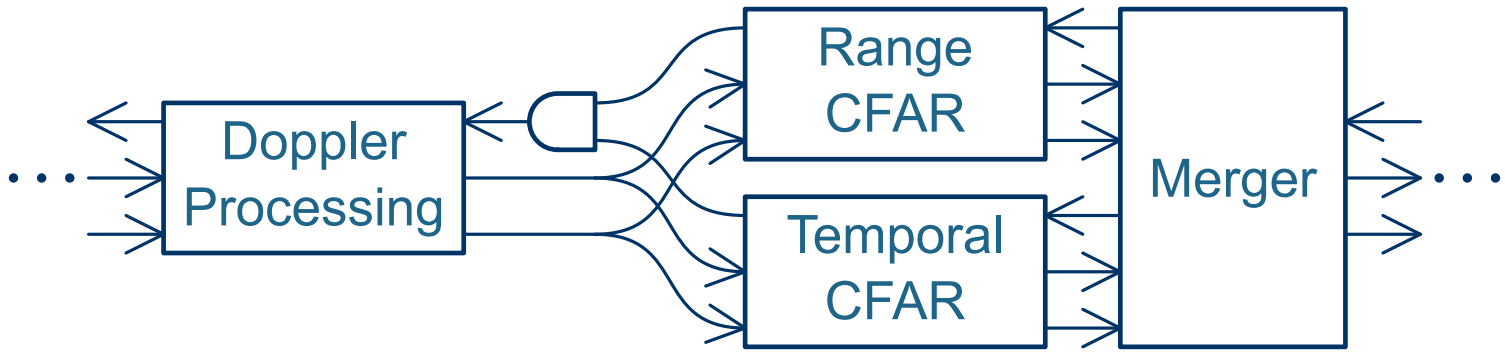
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Complex Structures



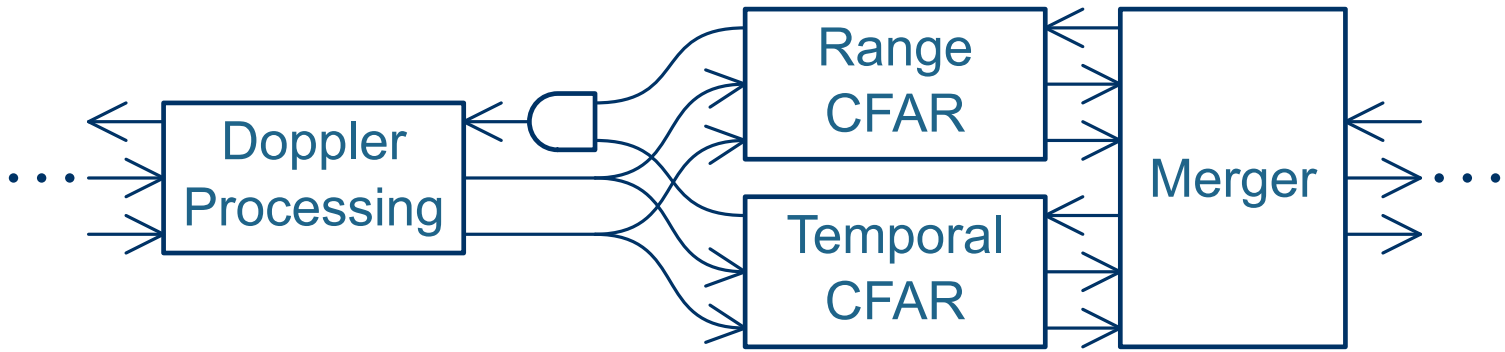
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Complex Structures



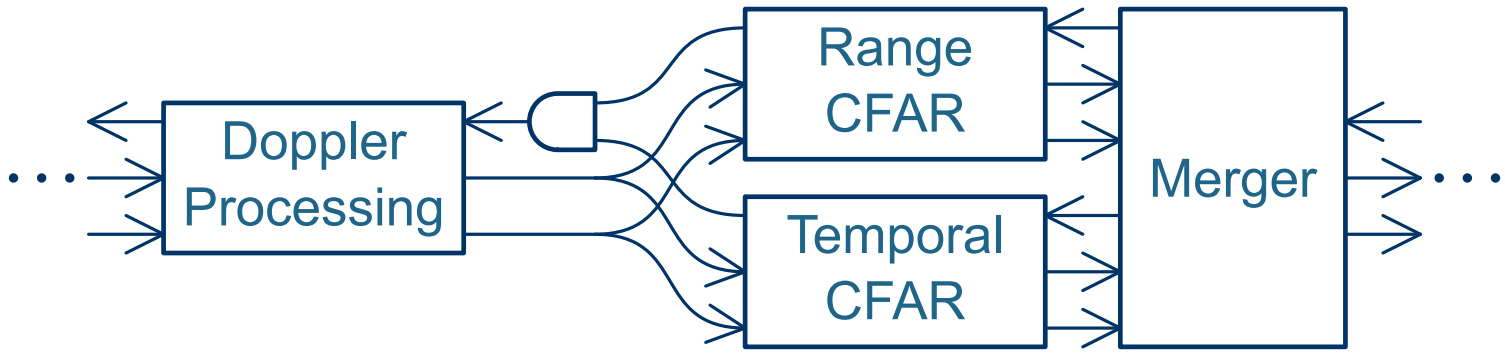
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Complex Structures



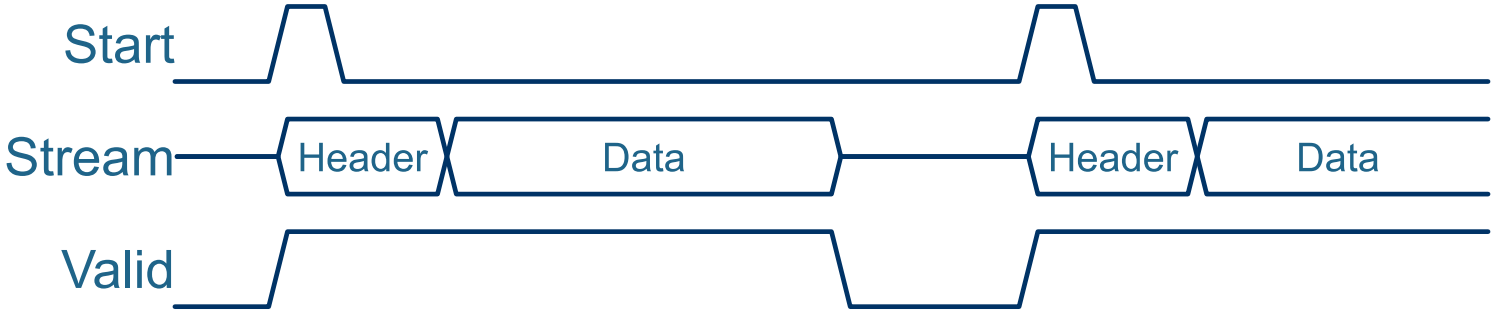
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Complex Structures



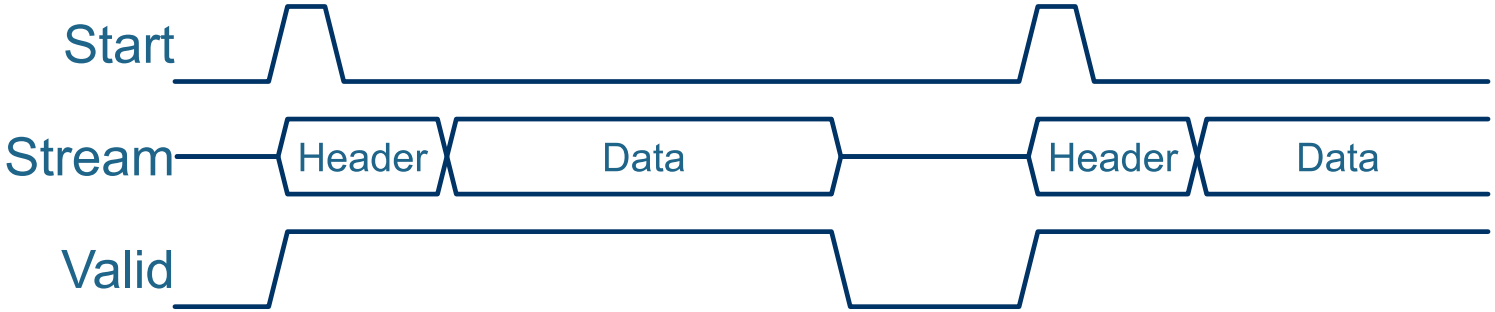
- Gate the “Valid” with the “Ready”
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Packet-based Streams



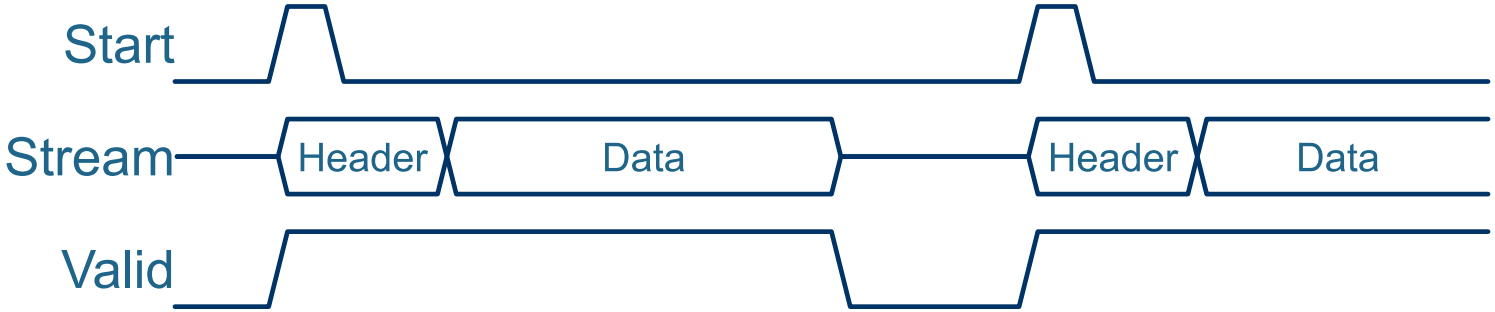
- Add a start-of-packet strobe (valid for one clock-cycle only)
- Natural fit for RADAR: one packet per PRI
- Natural fit for packet-based communication (e.g. Ethernet or UDP – makes it easy to implement Ethernet-based FPGA-in-the-loop testing)
- The header can contain all sorts of metadata...

Packet-based Streams



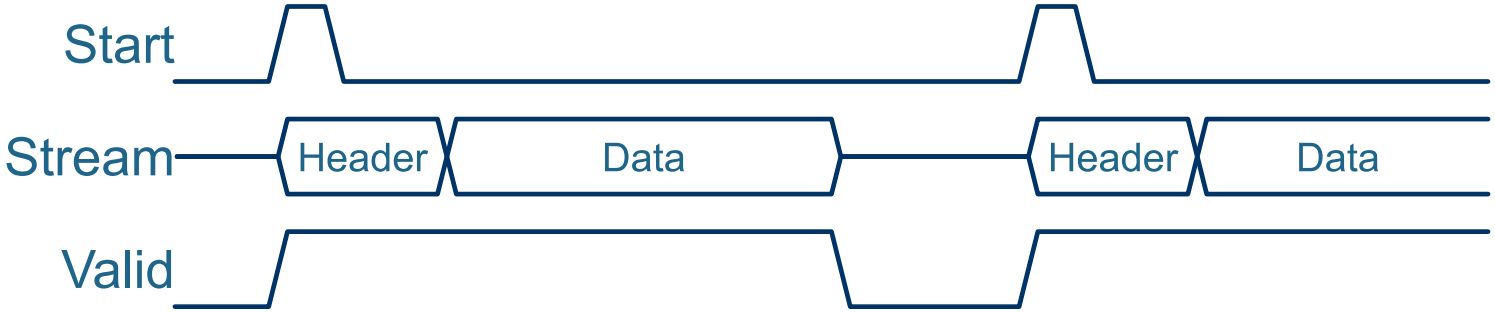
- Add a start-of-packet strobe (valid for one clock-cycle only)
- Natural fit for RADAR: one packet per PRI
- Natural fit for packet-based communication (e.g. Ethernet or UDP – makes it easy to implement Ethernet-based FPGA-in-the-loop testing)
- The header can contain all sorts of metadata...

Packet-based Streams



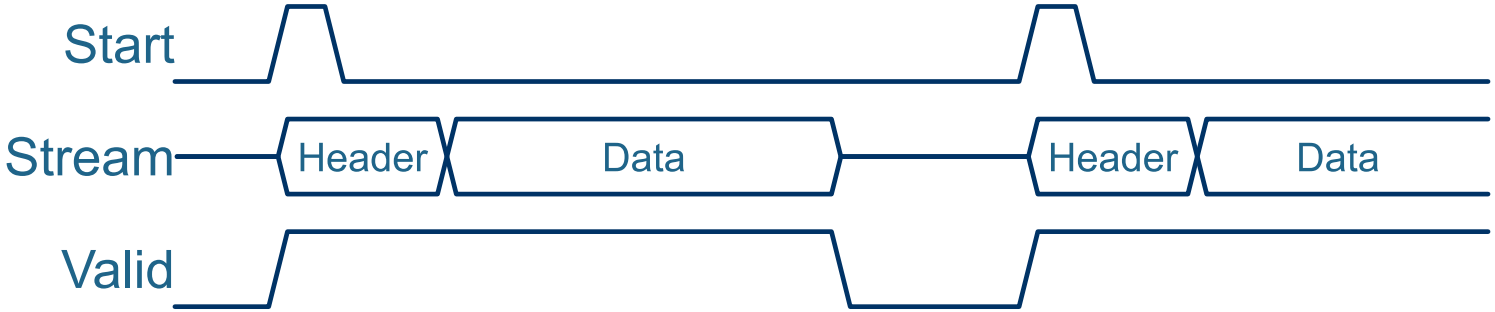
- Add a start-of-packet strobe (valid for one clock-cycle only)
- Natural fit for RADAR: one packet per PRI
- Natural fit for packet-based communication (e.g. Ethernet or UDP – makes it easy to implement Ethernet-based FPGA-in-the-loop testing)
- The header can contain all sorts of metadata...

Packet-based Streams



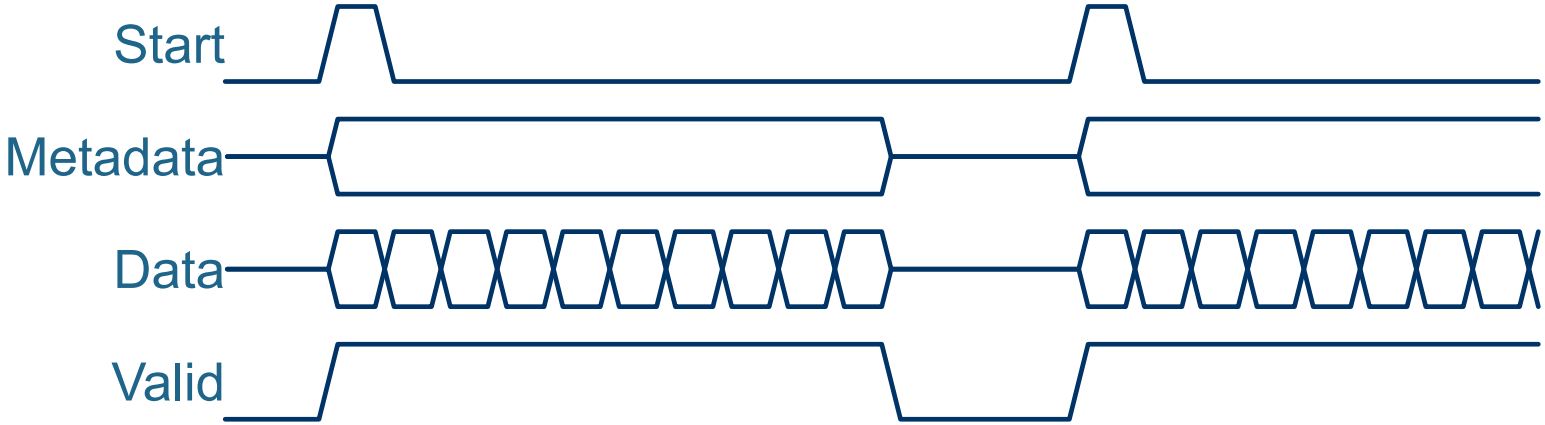
- Add a start-of-packet strobe (valid for one clock-cycle only)
- Natural fit for RADAR: one packet per PRI
- Natural fit for packet-based communication (e.g. Ethernet or UDP – makes it easy to implement Ethernet-based FPGA-in-the-loop testing)
- The header can contain all sorts of metadata...

Packet-based Streams



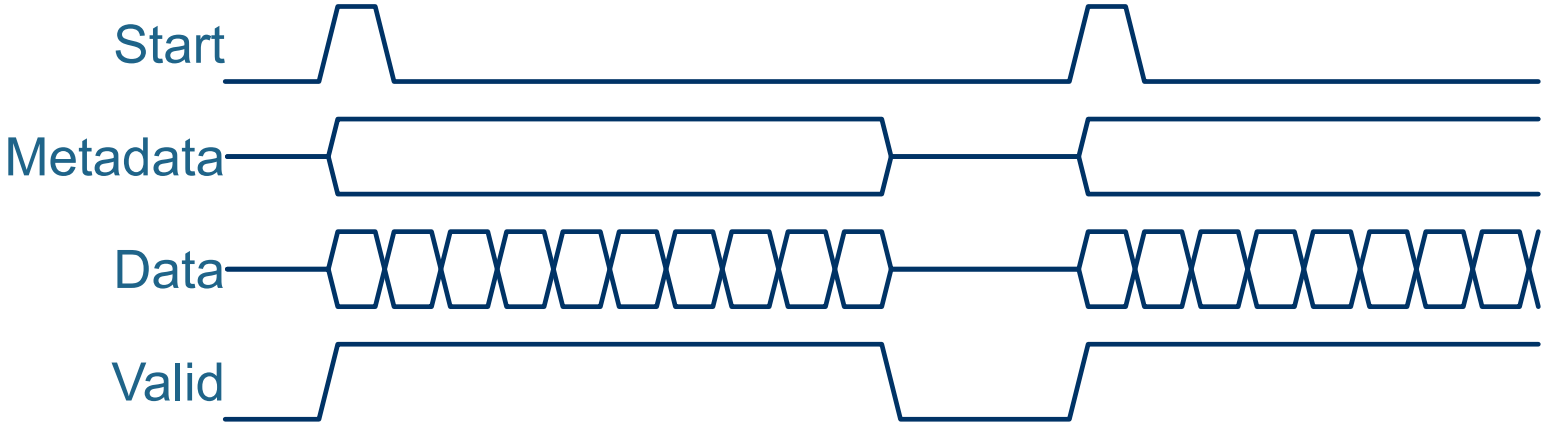
- Add a start-of-packet strobe (valid for one clock-cycle only)
- Natural fit for RADAR: one packet per PRI
- Natural fit for packet-based communication (e.g. Ethernet or UDP – makes it easy to implement Ethernet-based FPGA-in-the-loop testing)
- The header can contain all sorts of metadata...

Packet-based Streams



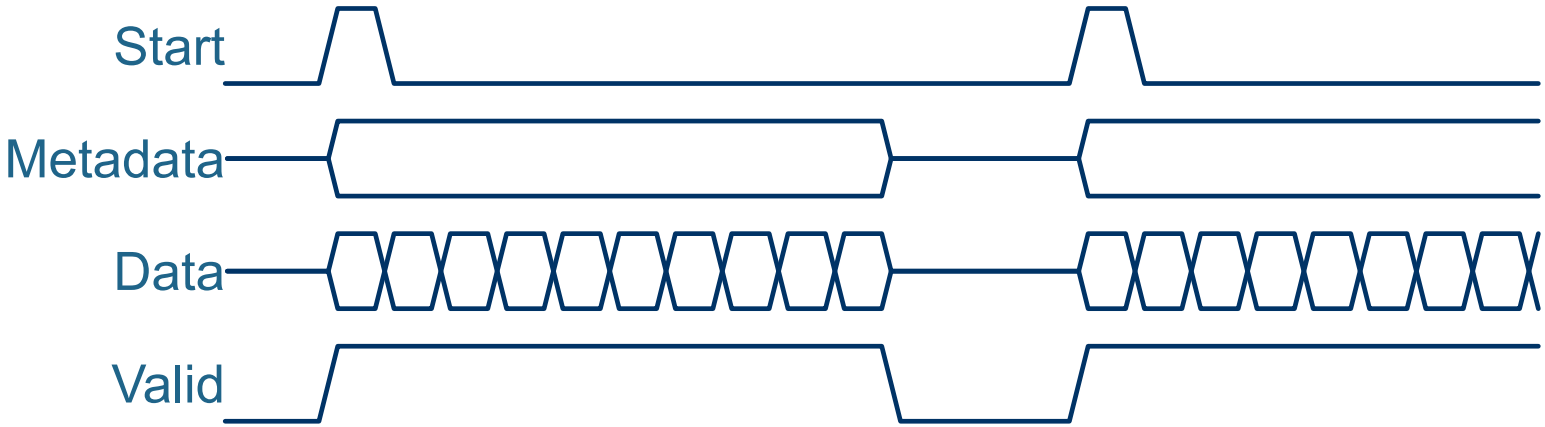
- Metadata can also be implemented as a side-channel
- The metadata is stable for the duration of the packet
- No need to extract and / or add the header
- Easier to filter the stream

Packet-based Streams



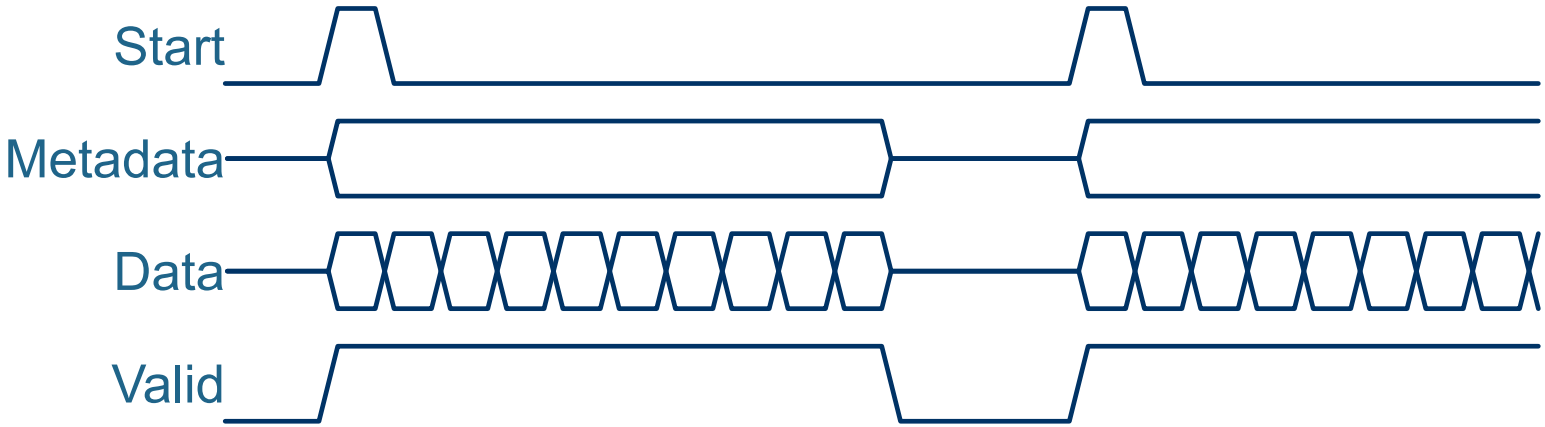
- Metadata can also be implemented as a side-channel
- The metadata is stable for the duration of the packet
- No need to extract and / or add the header
- Easier to filter the stream

Packet-based Streams



- Metadata can also be implemented as a side-channel
- The metadata is stable for the duration of the packet
- No need to extract and / or add the header
- Easier to filter the stream

Packet-based Streams



- Metadata can also be implemented as a side-channel
- The metadata is stable for the duration of the packet
- No need to extract and / or add the header
- Easier to filter the stream

Coffee Break...

Outline

Pipelines

Streaming Processors

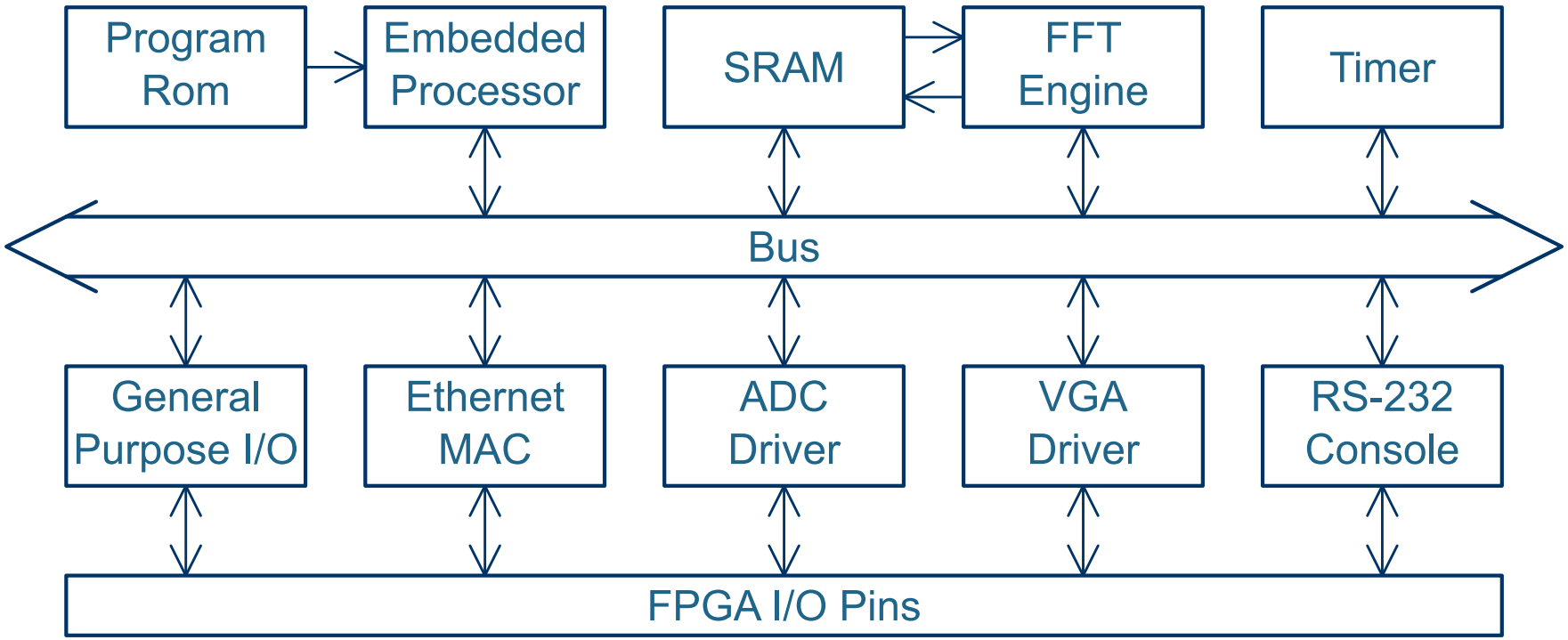
Memory-Mapped Bus

Pulse-width Modulation

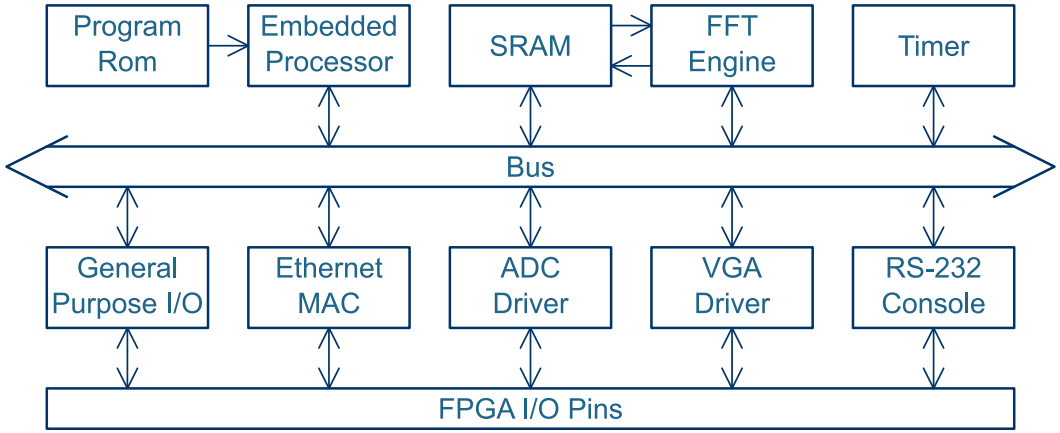
High Resolution PWM

THE RADAR MASTERS COURSE

Memory-Mapped Bus

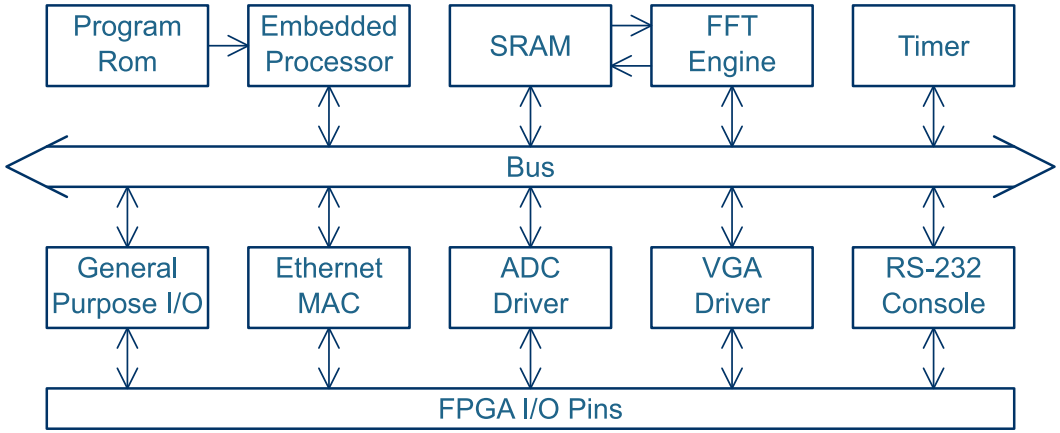


Memory-Mapped Bus



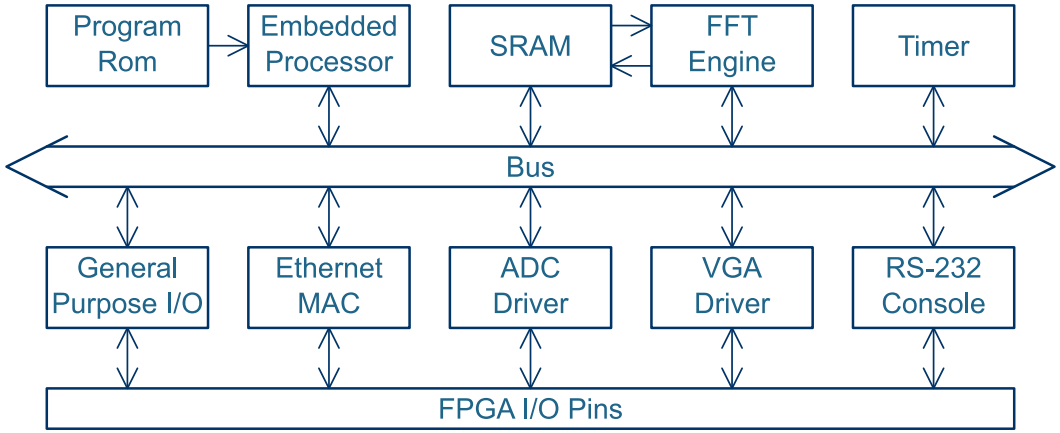
- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

Memory-Mapped Bus



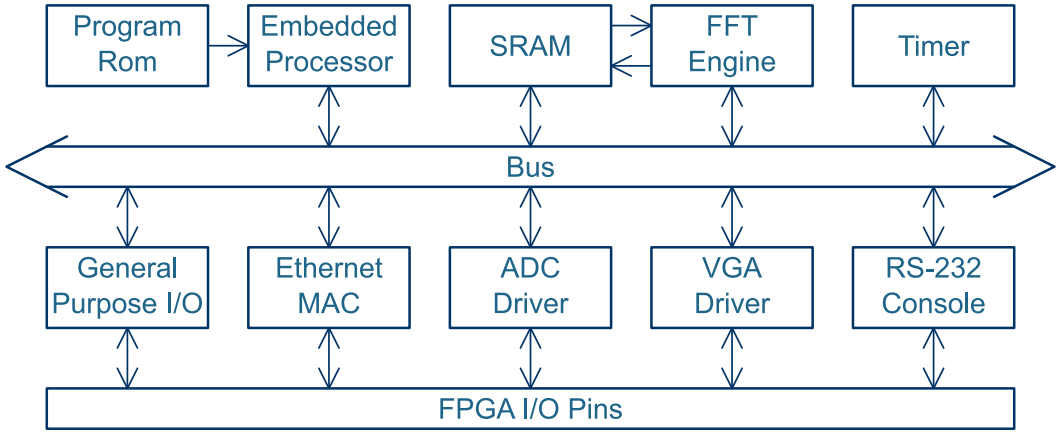
- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

Memory-Mapped Bus



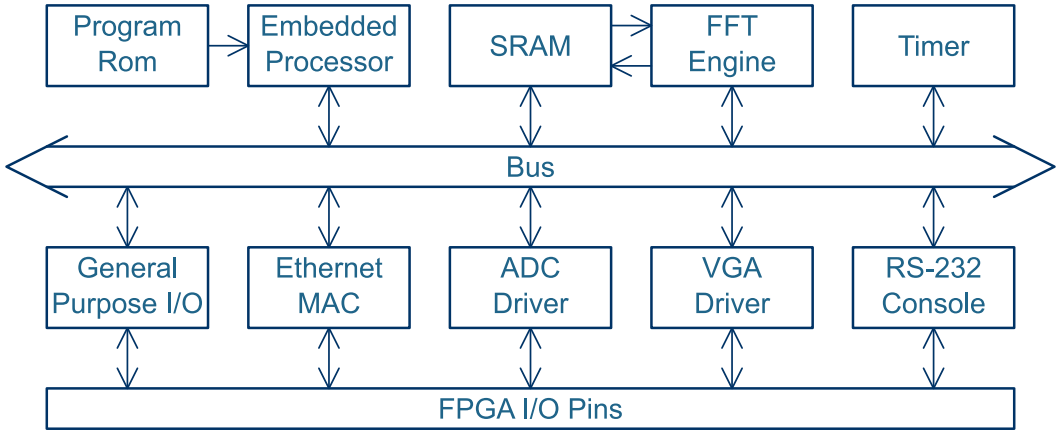
- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

Memory-Mapped Bus



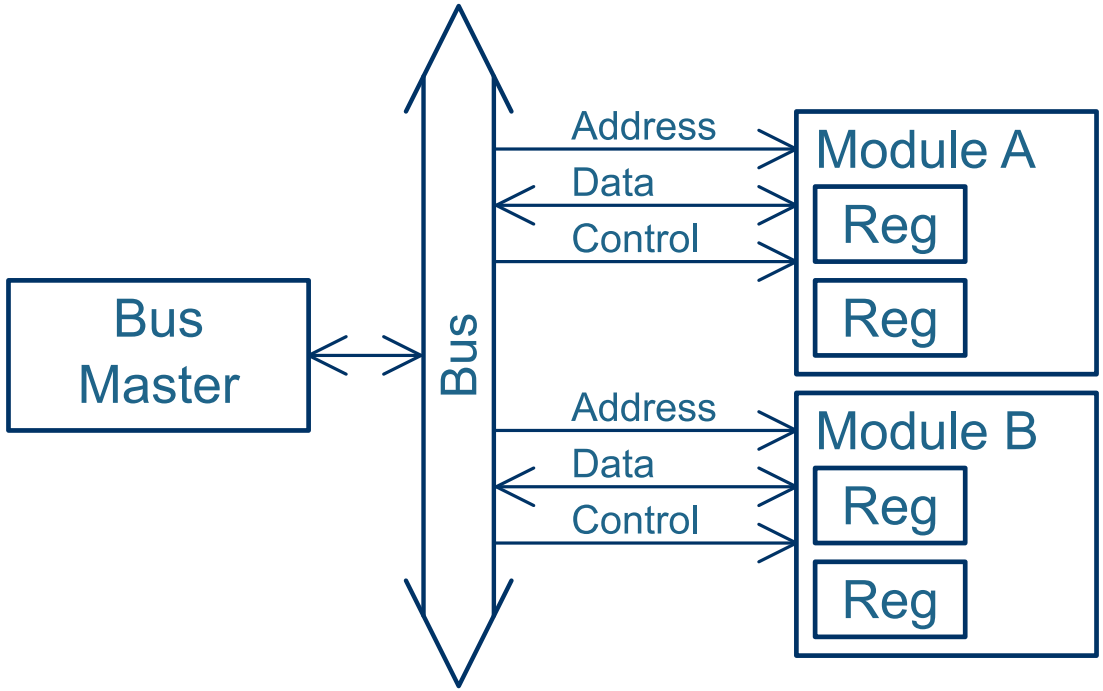
- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

Memory-Mapped Bus

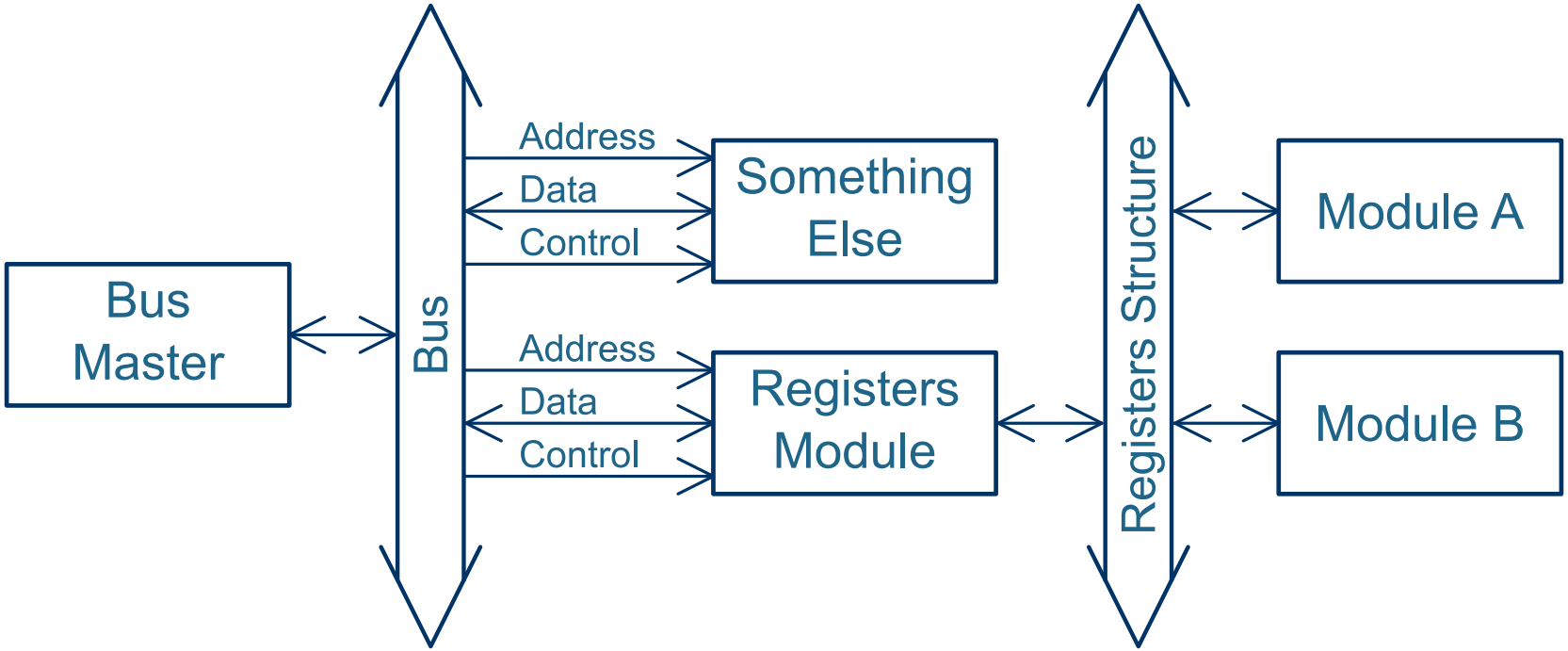


- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

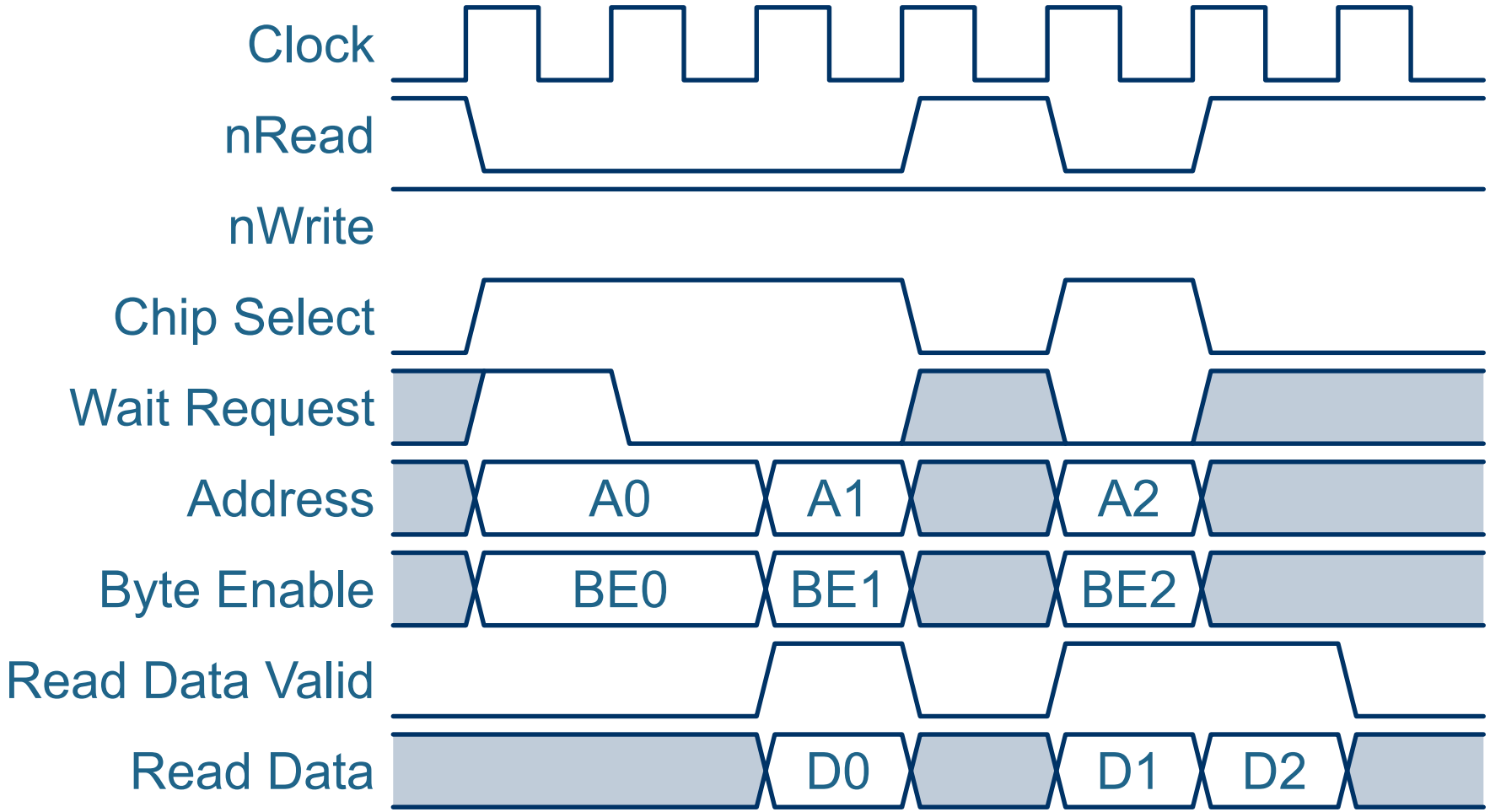
Memory-mapped Architectures



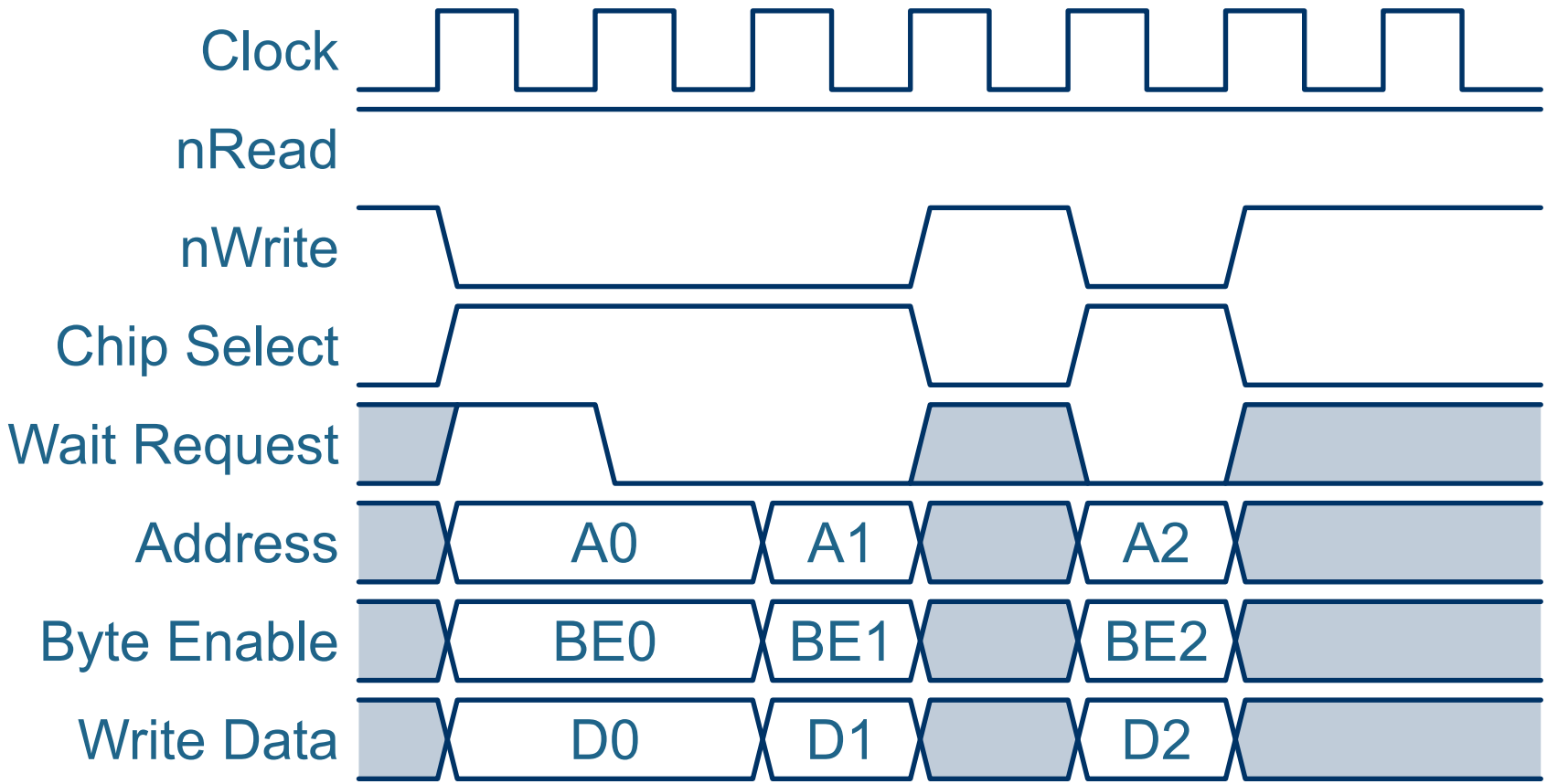
Memory-mapped Architectures



Interface Timing Diagrams



Interface Timing Diagrams



Structures

- SystemVerilog only
- Define these once in a separate file, typically as part of a package

```
package Structures;
  typedef struct{
    logic [ 1:0]Buttons;
    logic [ 9:0]Switches;
    logic [31:0]StatusBits;
  } RD_REGISTERS;

  typedef struct{
    logic [ 9:0]LEDs;
    logic [31:0]NCO_Frequency;
    logic [ 2:0]Bandwidth;
  } WR_REGISTERS;
endpackage
```

Structures

- SystemVerilog only
- Define these once in a separate file, typically as part of a package

```
package Structures;
  typedef struct{
    logic [ 1:0]Buttons;
    logic [ 9:0]Switches;
    logic [31:0]StatusBits;
  } RD_REGISTERS;

  typedef struct{
    logic [ 9:0]LEDs;
    logic [31:0]NCO_Frequency;
    logic [ 2:0]Bandwidth;
  } WR_REGISTERS;
endpackage
```

Structures


```
import Structures::*;

module Registers(
   input ipClk, ipReset,

   input   [15:0]ipAddress,
   output  [31:0]opRdData,
   input   [31:0]ipWrData,
   input         ipWrEn,

   input   RD_REGISTERS ipRdRegisters,
   output  WR_REGISTERS opWrRegisters
);
...
```

Structures


```
// In the top-level module...

RD_REGISTERS RdRegisters;
WR_REGISTERS WrRegisters;

Registers Registers_Inst(
  Clk, Reset,
  Address, RdData, WrData, WrEn,
  RdRegisters, WrRegisters
);

NCO NCO_Inst(
  Clk, Reset,
  WrRegisters.NCO_Frequency,
  RdRegisters.StatusBits[15]
);
```

Structures

- You can map a structure to an input port and then use only what you need within the submodule
- You can not map the same structure to an output port of more than one module – you need to map the structure members individually
- Luckily, most registers are control registers, and therefore input ports of the target modules

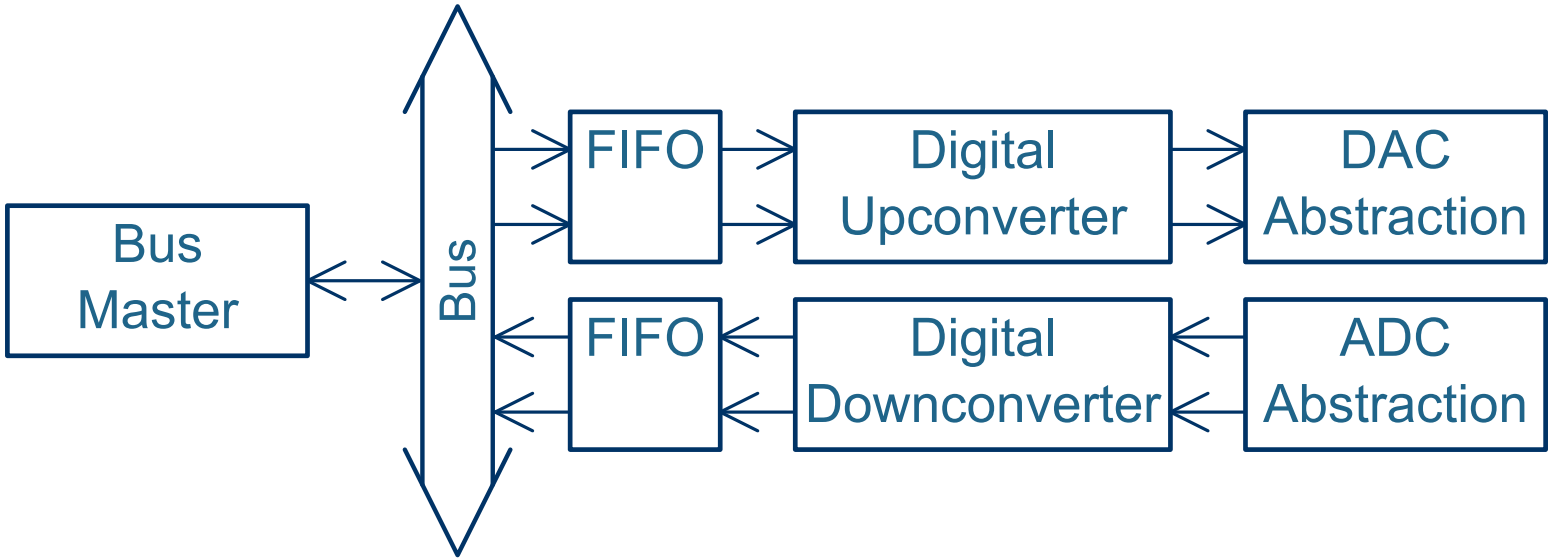
Structures

- You can map a structure to an input port and then use only what you need within the submodule
- You can not map the same structure to an output port of more than one module – you need to map the structure members individually
- Luckily, most registers are control registers, and therefore input ports of the target modules

Structures

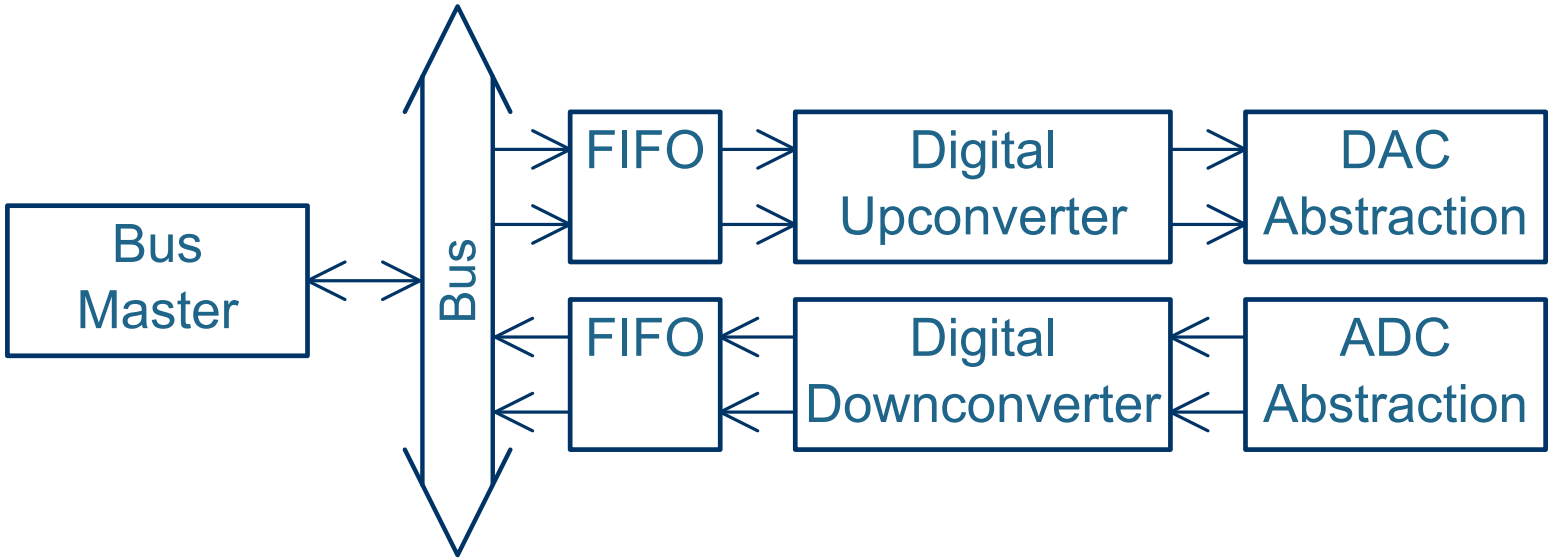
- You can map a structure to an input port and then use only what you need within the submodule
- You can not map the same structure to an output port of more than one module – you need to map the structure members individually
- Luckily, most registers are control registers, and therefore input ports of the target modules

Memory-mapped Streams



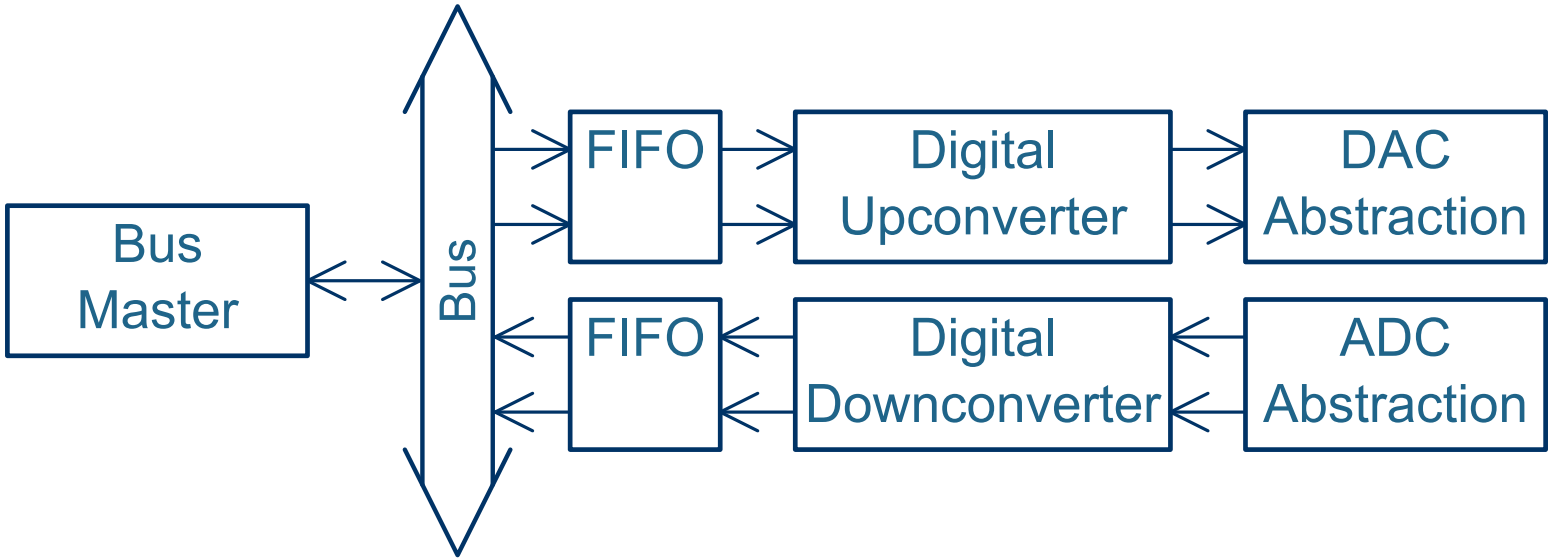
- It is often convenient to allocate a bus address to a stream
- A write to that address injects one unit into the stream
- A read from that address reads one unit from the stream
- Most often unidirectional and FIFO-buffered with feed-back registers

Memory-mapped Streams



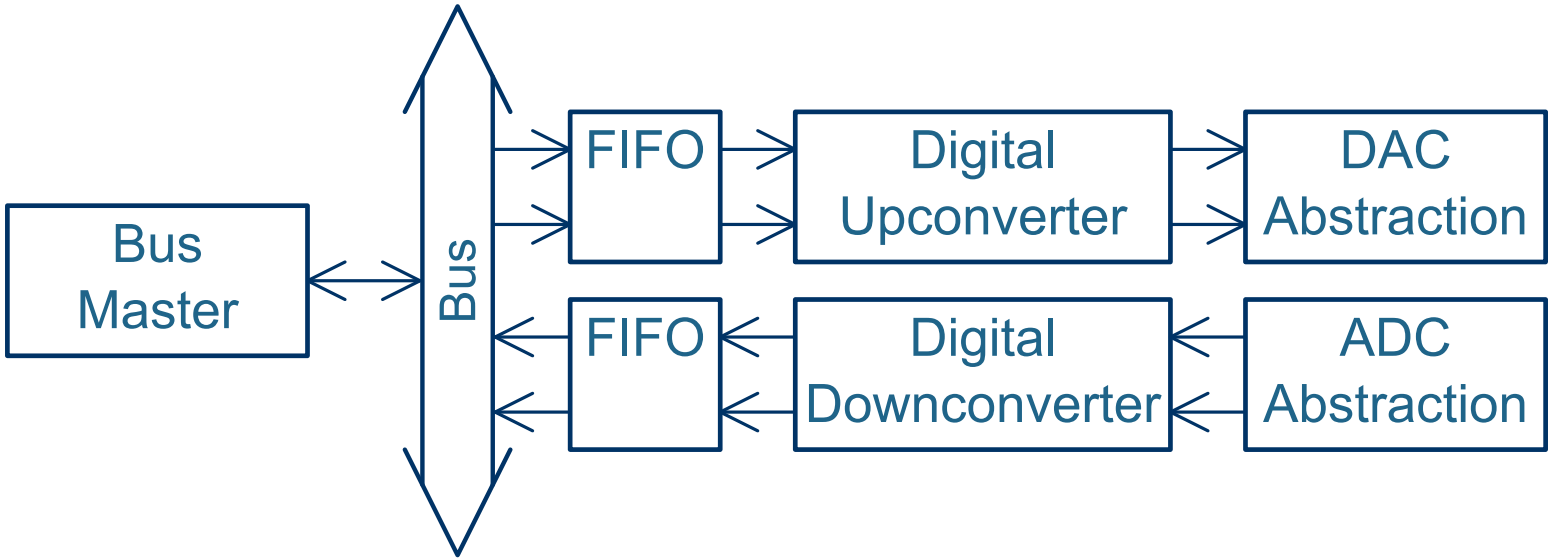
- It is often convenient to allocate a bus address to a stream
- A write to that address injects one unit into the stream
- A read from that address reads one unit from the stream
- Most often unidirectional and FIFO-buffered with feed-back registers

Memory-mapped Streams



- It is often convenient to allocate a bus address to a stream
- A write to that address injects one unit into the stream
- A read from that address reads one unit from the stream
- Most often unidirectional and FIFO-buffered with feed-back registers

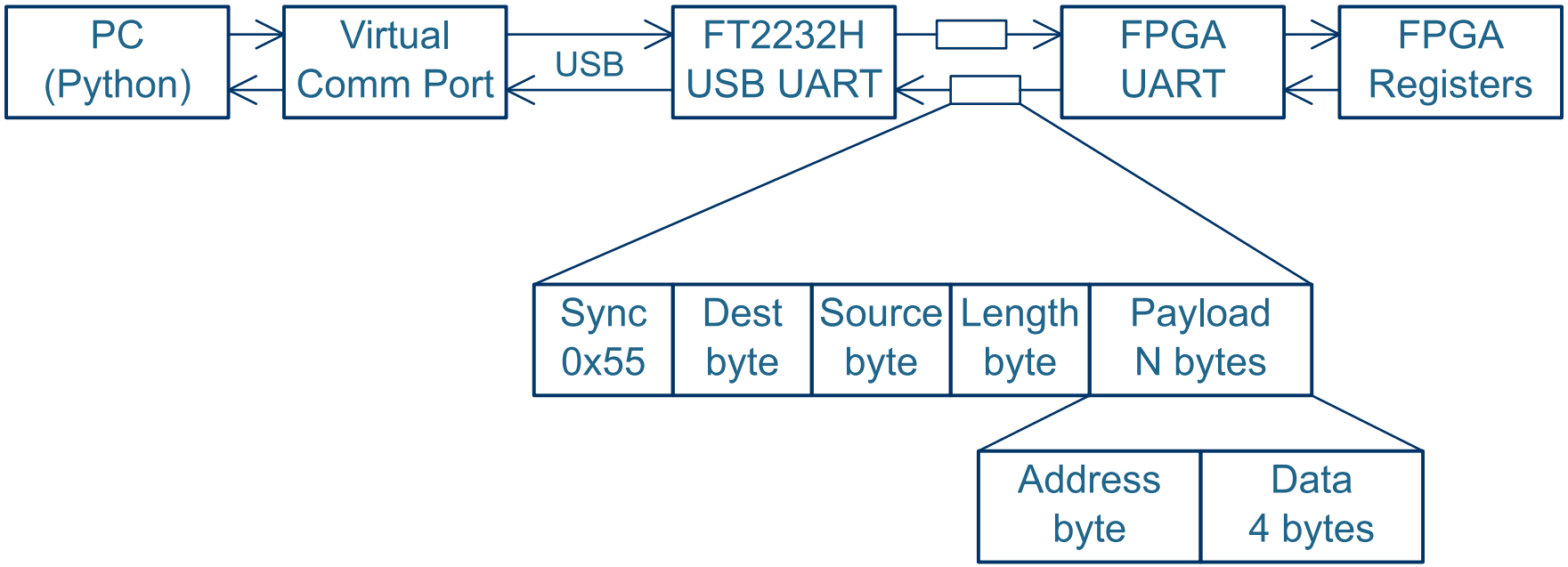
Memory-mapped Streams



- It is often convenient to allocate a bus address to a stream
- A write to that address injects one unit into the stream
- A read from that address reads one unit from the stream
- Most often unidirectional and FIFO-buffered with feed-back registers

UART Controlled Bus

Practical `05 - Registers` makes use of a UART interface to control memory-mapped FPGA registers.



Outline

Pipelines

Streaming Processors

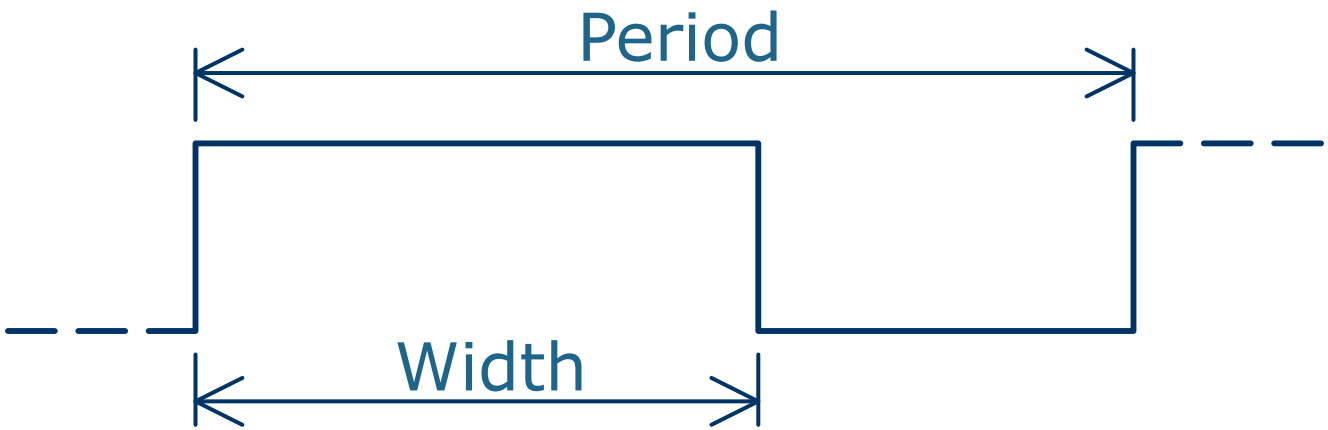
Memory-Mapped Bus

Pulse-width Modulation

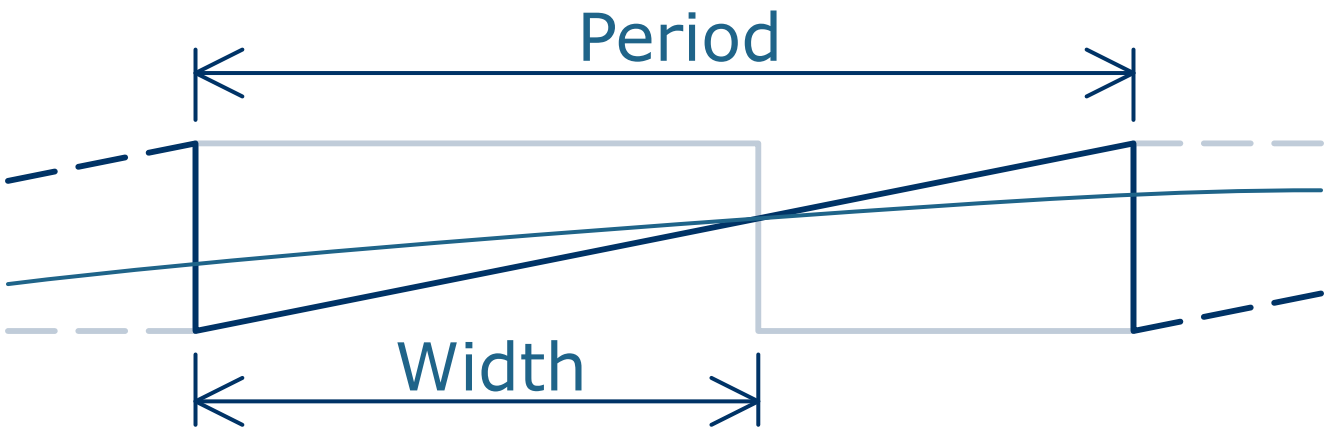
High Resolution PWM

THE RADAR MASTERS COURSE

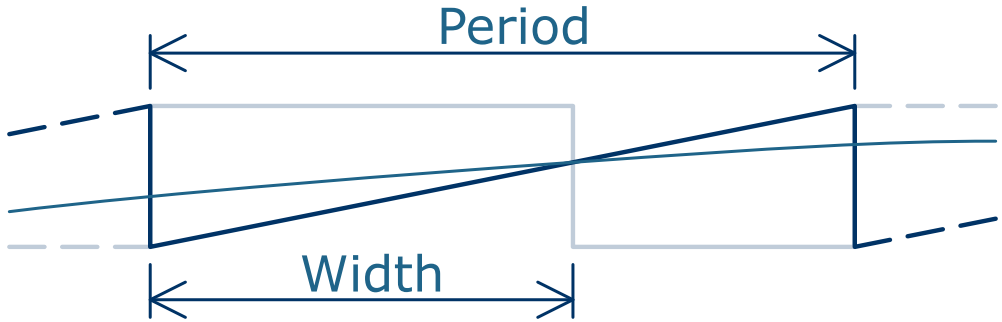
Pulse-width Modulation



Pulse-width Modulation

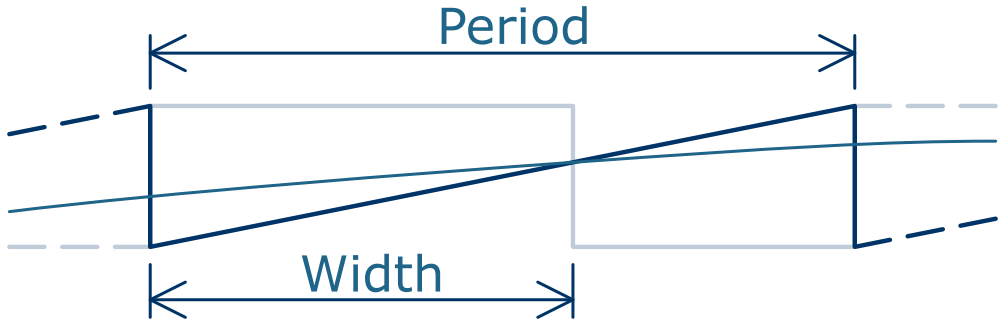


Pulse-width Modulation



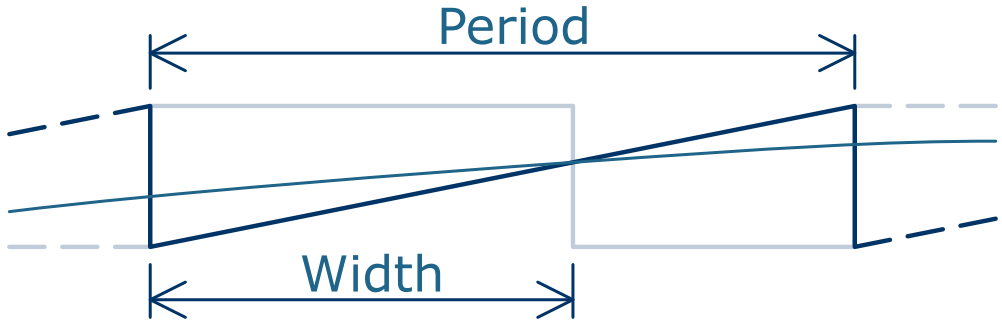
- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation



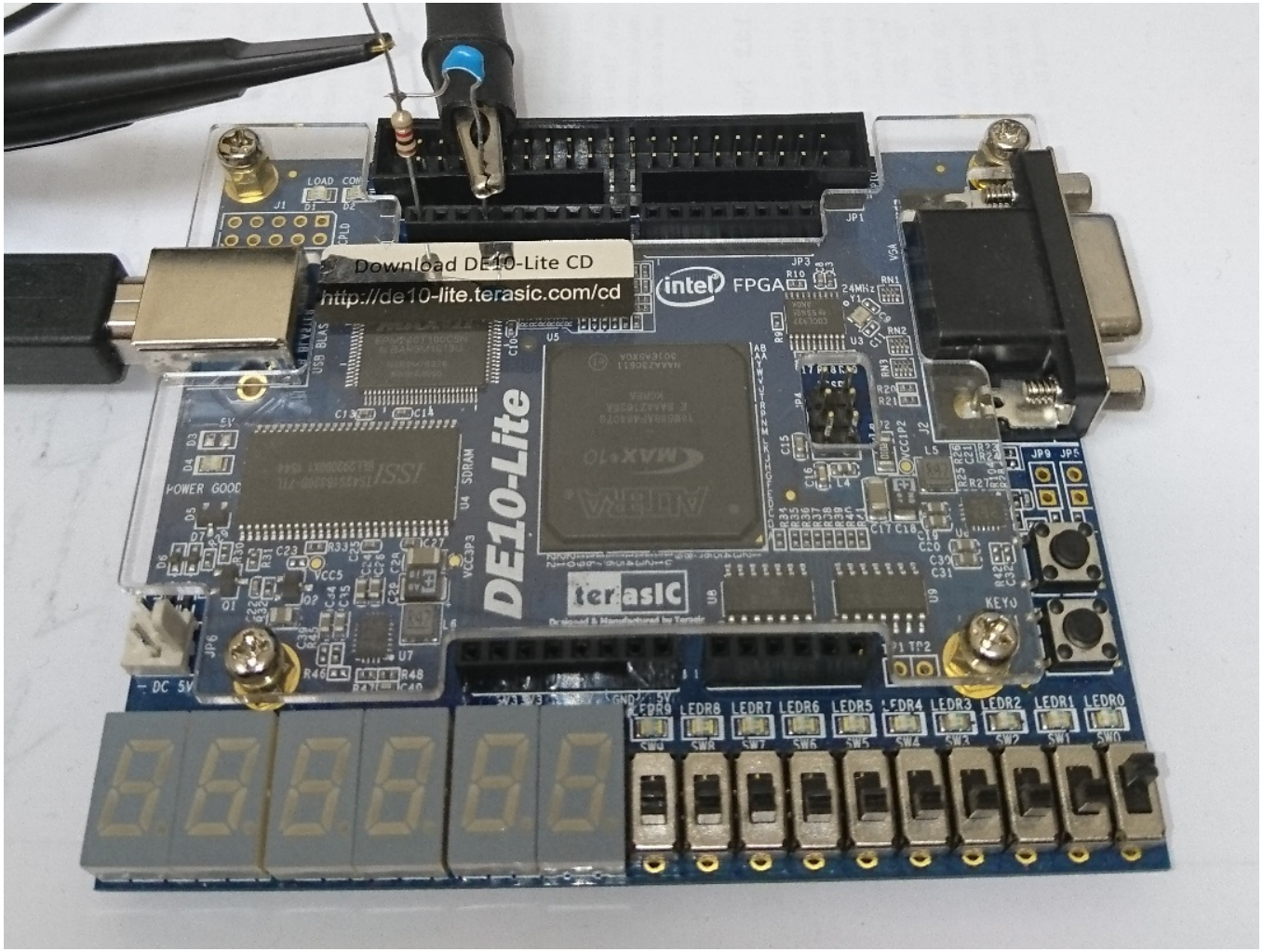
- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation

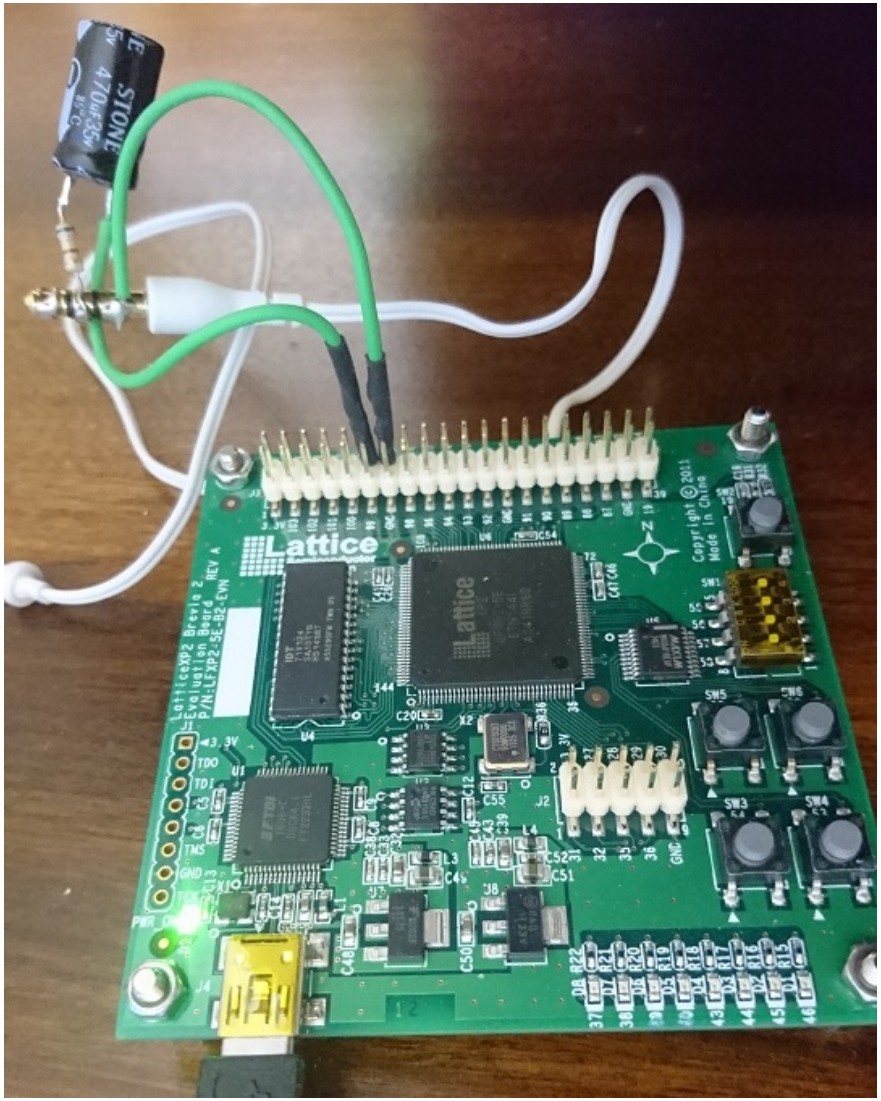


- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation

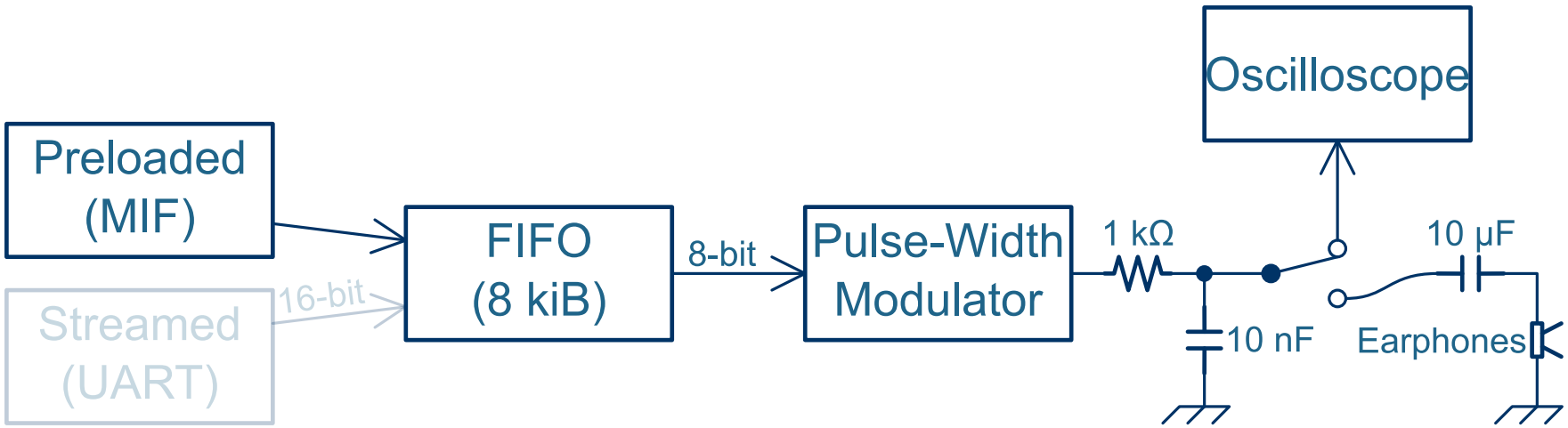


Pulse-width Modulation



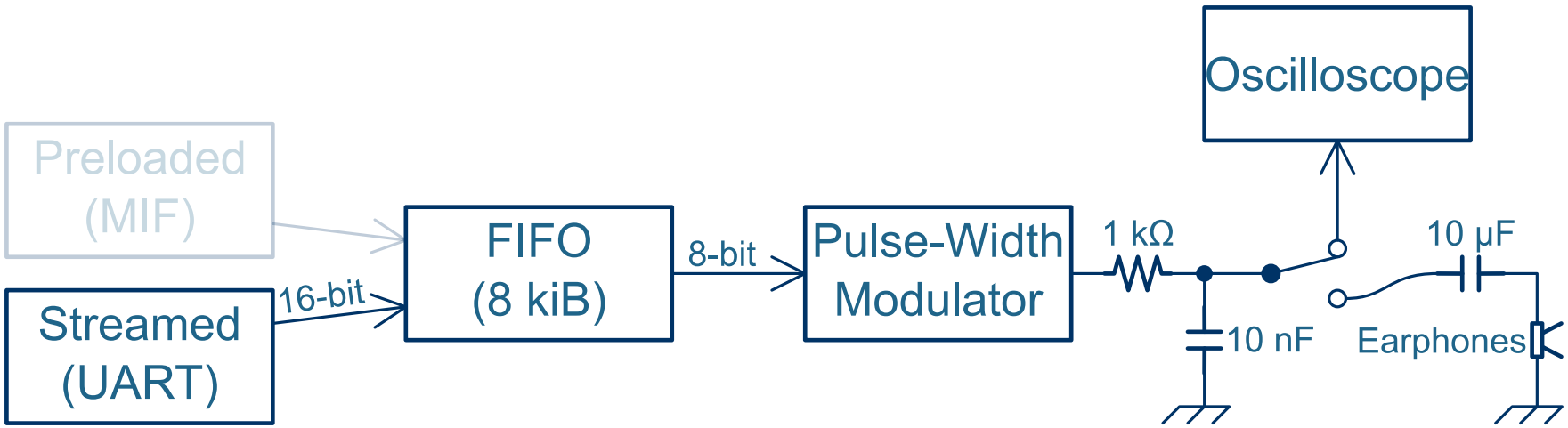
Simple Injection

Before continuing,
get the PWM and simple injection modules working...



UART Injection

Keep in mind that the same FIFO-based injection module must interface with the 16-bit UART or SRAM based injection...



Outline

Pipelines

Streaming Processors

Memory-Mapped Bus

Pulse-width Modulation

High Resolution PWM

THE RADAR MASTERS COURSE

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ 2^{16} = 13.1072 GHz
- That is a FAST clock!
- The solution...

High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

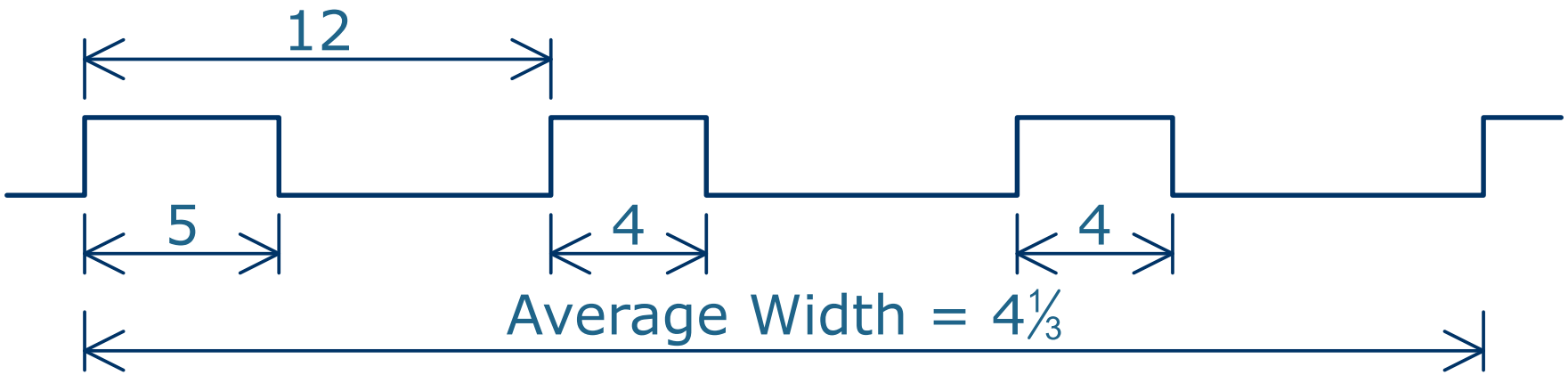
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

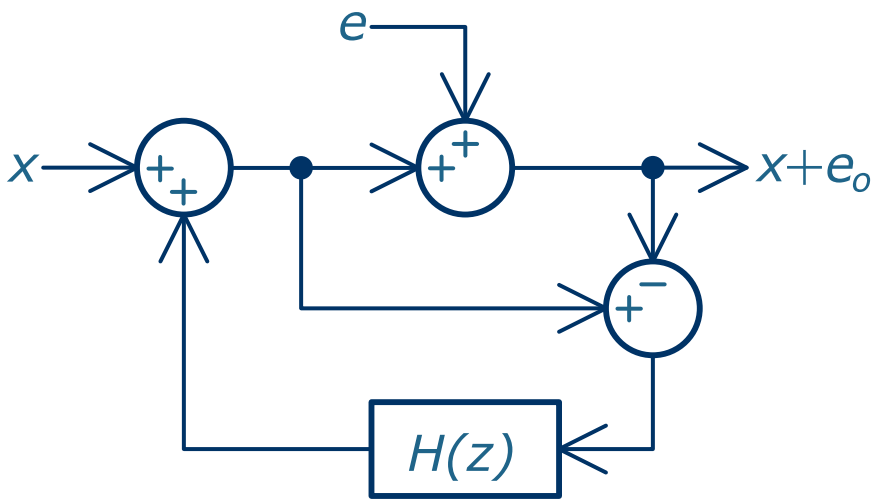
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

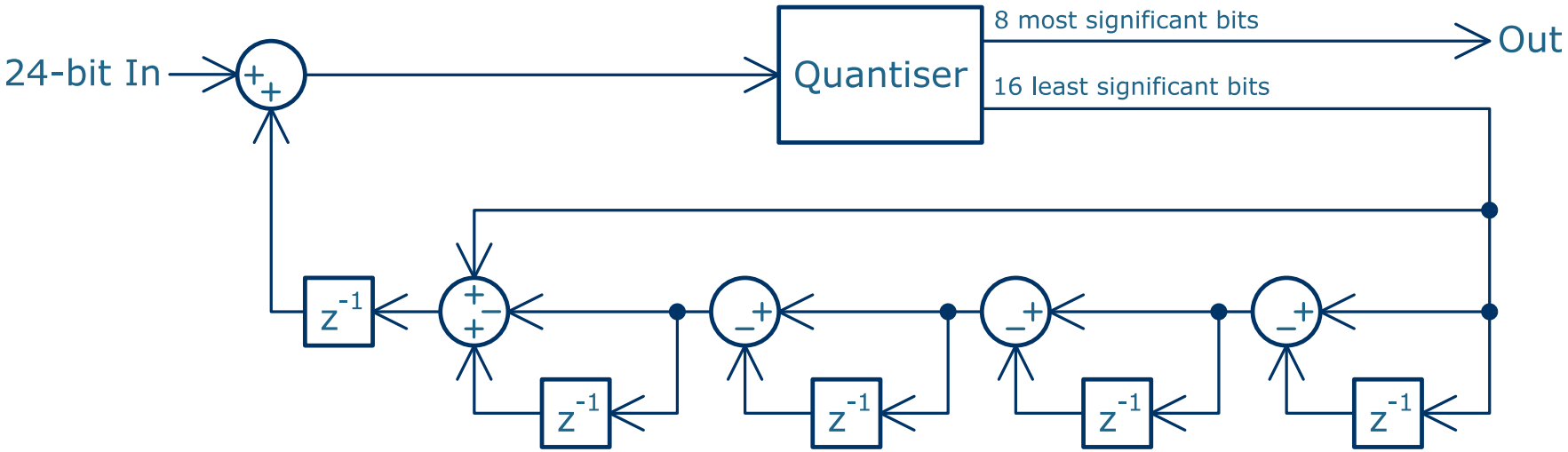
Noise Shaping



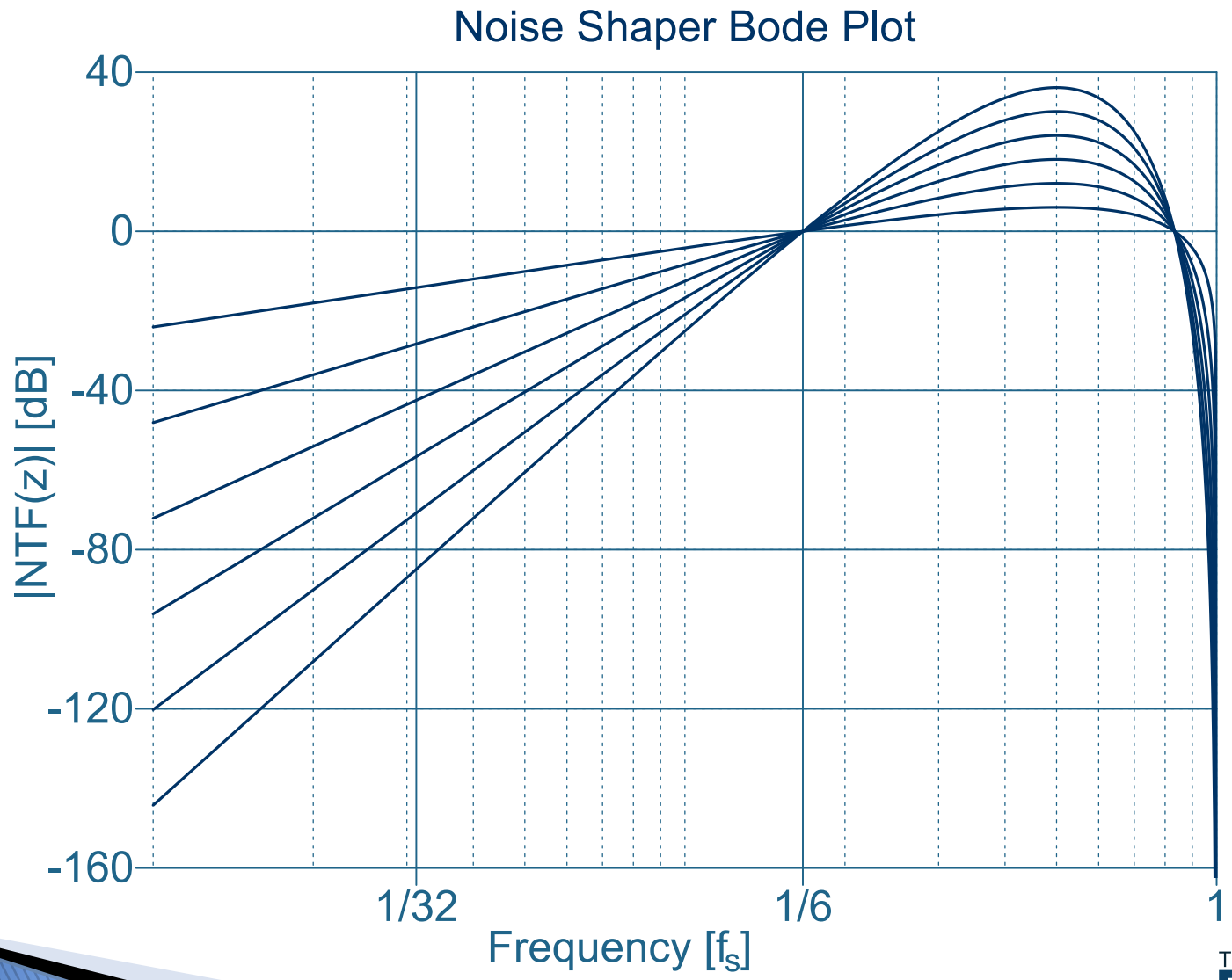
Noise Shaping



Noise Shaping



Noise Shaping



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 3 – 29 April 2022