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 Dr John-Philip Taylor

Convened by Dr Stephen Paine

Day 3 – 11 September 2024

Outline

Outline

Memory-Mapped Bus

Advanced Clocking

Clock Domains

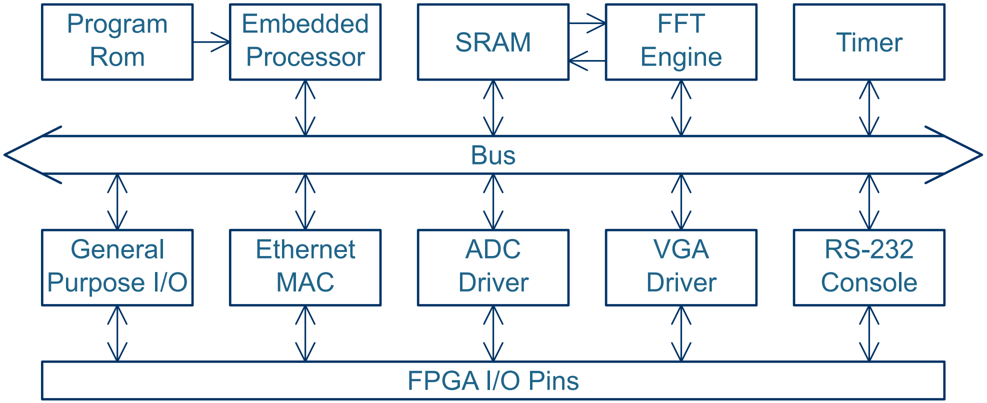
Pipelines

Streaming Processors

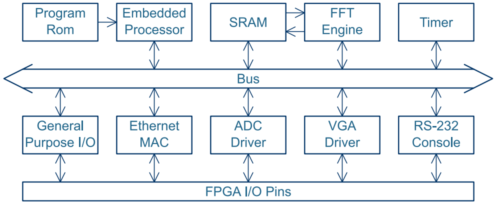
Flow Control

THE RADAR MASTERS COURSE

Memory-Mapped Bus



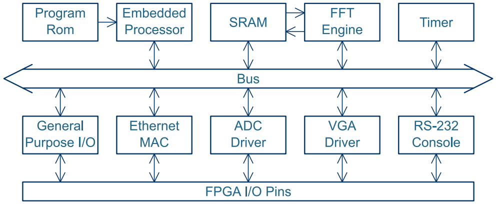
Memory-Mapped Bus



- 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

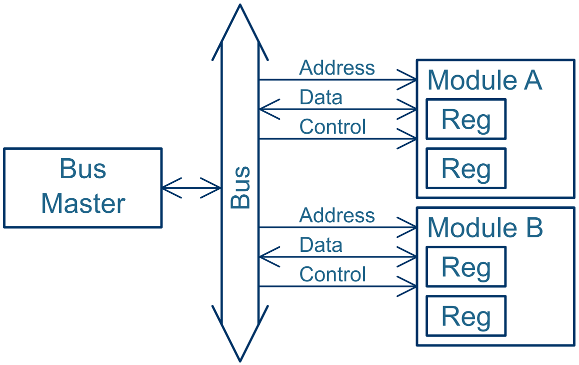
THE RADAR MASTERS COURSE

Memory-Mapped Bus

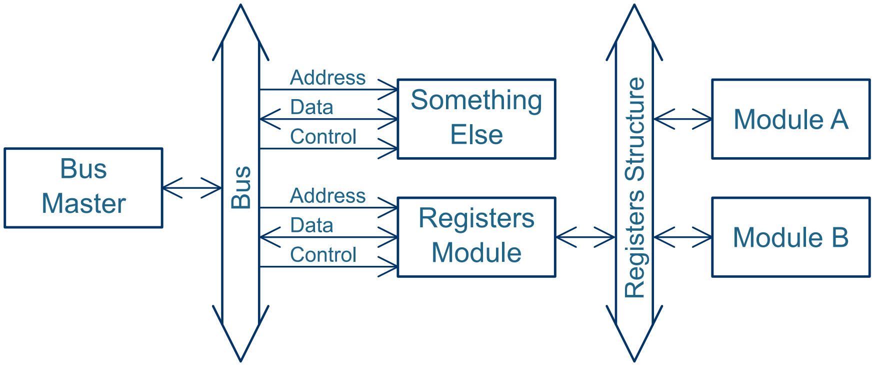


- Every node on the bus has an address range allocated
- Generally used for writing control registers, reading system status and accessing memory
- Altera Qsys uses Avalon
- Xilinx IP Integrator and ARM processors use AXI
- Many open-source projects use Wishbone

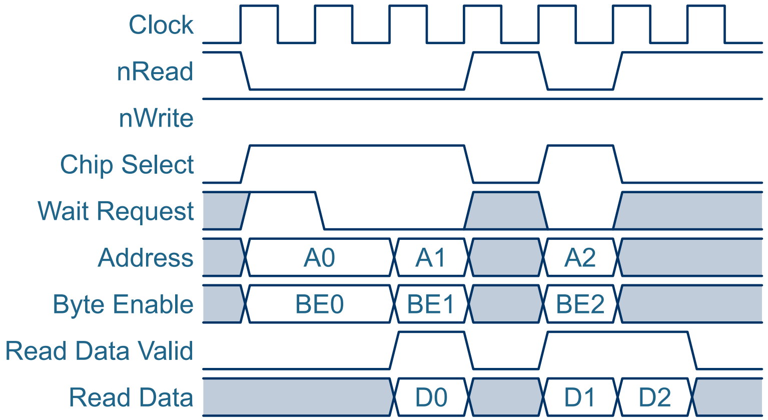
Memory-mapped Architectures



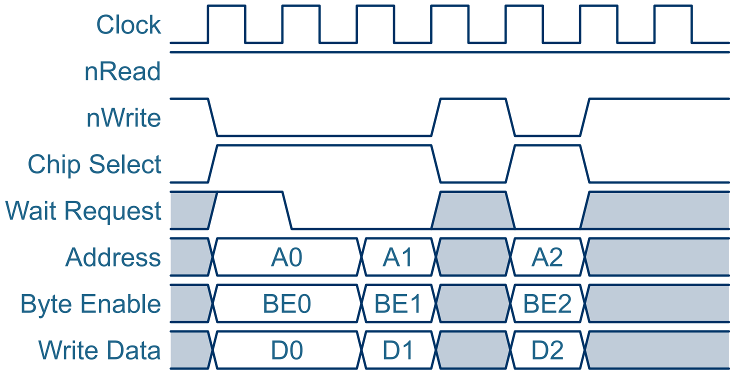
Memory-mapped Architectures



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

```
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,
  i.e. you need to map the structure members individually
- Luckily, most registers are control registers,
  and therefore input ports of the target modules

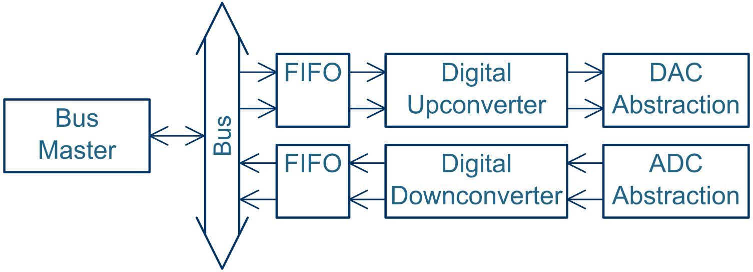
Structures

- 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,
  i.e. 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,
  i.e. you need to map the structure members individually
- Luckily, most registers are control registers,
  and therefore input ports of the target modules

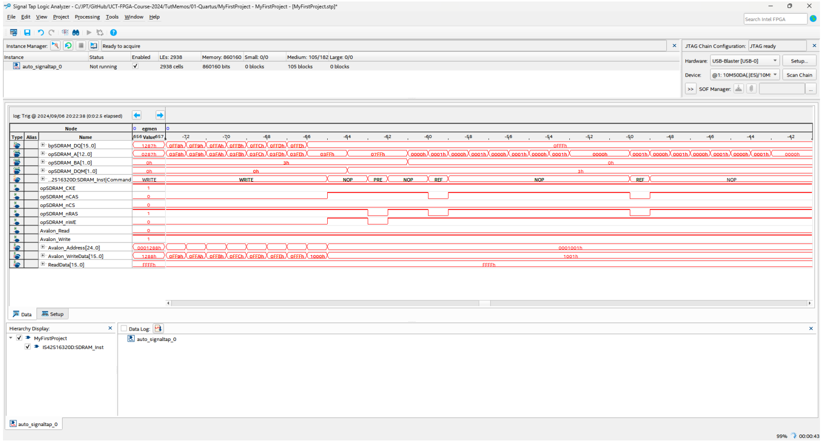
Memory-mapped Streams



- 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

Platform Designer

Practical `05` – `SDRAM` hooks up the SDRAM and introduces Signal Tap



Signal Tap Logic Analyzer - C:/JPT/GitHub/UCT-FPGA-Course-2024/TutMemos/01-Quartus/MyFirstProject - MyFirstProject - [MyFirstProject.stp]*

Instance Manager: Ready to acquire

Instance	Status	Enabled	LEs: 2938	Memory: 860160	Small: 0/0	Medium: 105/182	Large: 0/0
auto_signaltap_0	Not running	✓	2938 cells	860160 bits	0 blocks	105 blocks	0 blocks

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 10M50DA(.|ES)/10M!

log: Trig @ 2024/09/06 20:22:38 (0:0:2.5 elapsed)

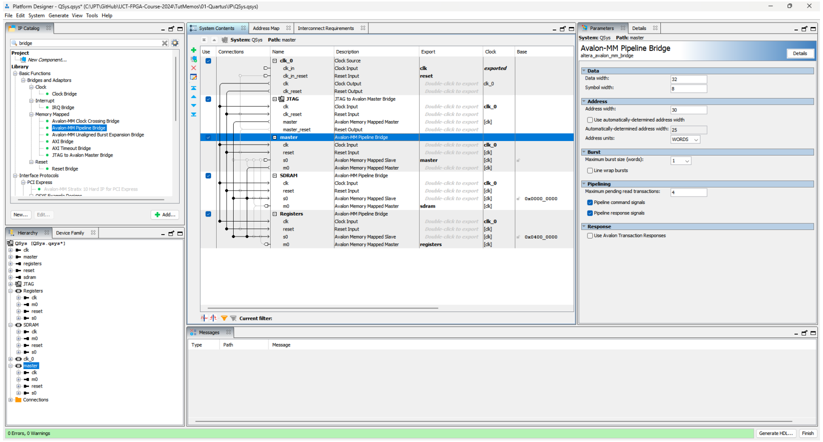
Name	Value
bpSDRAM_DQ[15..0]	1287h
opSDRAM_A[12..0]	0287h
opSDRAM_BA[1..0]	0h
opSDRAM_DQM[1..0]	0h
...2S16320D:SDRAM_Inst|Command	WRITE
opSDRAM_CKE	1
opSDRAM_nCAS	0
opSDRAM_nCS	0
opSDRAM_nRAS	1
opSDRAM_nWE	0
Avalon_Read	0
Avalon_Write	1
Avalon_Address[24..0]	0001288h
Avalon_WriteData[15..0]	1288h
ReadData[15..0]	FFFFh

Hierarchy Display:
MyFirstProject
IS42S16320D:SDRAM_Inst

Data Log:
auto_signaltap_0

Platform Designer

Practical `06 – Platform Designer` introduces the Platform Designer


Platform Designer - QSys.qsys* (C:\JPT\GitHub\UCT-FPGA-Course-2024\TutMemos\01-Quartus\IP\QSys.qsys)

IP Catalog (search: bridge)

- Project
  - *New Component...*
- Library
  - Basic Functions
    - Bridges and Adaptors
      - Clock
        - Clock Bridge
      - Interrupt
        - IRQ Bridge
      - Memory Mapped
        - Avalon-MM Clock Crossing Bridge
        - Avalon-MM Pipeline Bridge
        - Avalon-MM Unaligned Burst Expansion Bridge
        - AXI Bridge
        - AXI Timeout Bridge
        - JTAG to Avalon Master Bridge
      - Reset
        - Reset Bridge
  - Interface Protocols
    - PCI Express
      - Avalon-MM Stratix 10 Hard IP for PCI Express

Hierarchy

- QSys [QSys.qsys*]
  - clk
  - master
  - registers
  - reset
  - sdram
  - JTAG
  - Registers
    - clk
    - m0
    - reset
    - s0
  - SDRAM
    - clk
    - m0
    - reset
    - s0
  - clk_0
  - master
    - clk
    - m0
    - reset
    - s0
  - Connections

System Contents — System: QSys Path: master

Use	Name	Description	Export	Clock	Base
✓	clk_0	Clock Source			
	clk_in	Clock Input	**clk**	*exported*	
	clk_in_reset	Reset Input	**reset**		
	clk	Clock Output	*Double-click to export*	clk_0	
	clk_reset	Reset Output	*Double-click to export*		
✓	JTAG	JTAG to Avalon Master Bridge			
	clk	Clock Input	*Double-click to export*	clk_0	
	clk_reset	Reset Input	*Double-click to export*		
	master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
	master_reset	Reset Output	*Double-click to export*		
✓	master	Avalon-MM Pipeline Bridge			
	clk	Clock Input	*Double-click to export*	clk_0	
	reset	Reset Input	*Double-click to export*	[clk]	
	s0	Avalon Memory Mapped Slave	**master**	[clk]	
	m0	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
✓	SDRAM	Avalon-MM Pipeline Bridge			
	clk	Clock Input	*Double-click to export*	clk_0	
	reset	Reset Input	*Double-click to export*	[clk]	
	s0	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0000
	m0	Avalon Memory Mapped Master	**sdram**	[clk]	
✓	Registers	Avalon-MM Pipeline Bridge			
	clk	Clock Input	*Double-click to export*	clk_0	
	reset	Reset Input	*Double-click to export*	[clk]	
	s0	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0400_0000
	m0	Avalon Memory Mapped Master	**registers**	[clk]	

Parameters — System: QSys Path: master

Avalon-MM Pipeline Bridge
altera_avalon_mm_bridge

Data

Data width:	32
Symbol width:	8

Address

Address width:	30
☐ Use automatically-determined address width	
Automatically-determined address width:	25
Address units:	WORDS

Burst

Maximum burst size (words):	1
☐ Line wrap bursts	

Pipelining

Maximum pending read transactions:	4
✓ Pipeline command signals	
✓ Pipeline response signals	

Response

☐ Use Avalon Transaction Responses

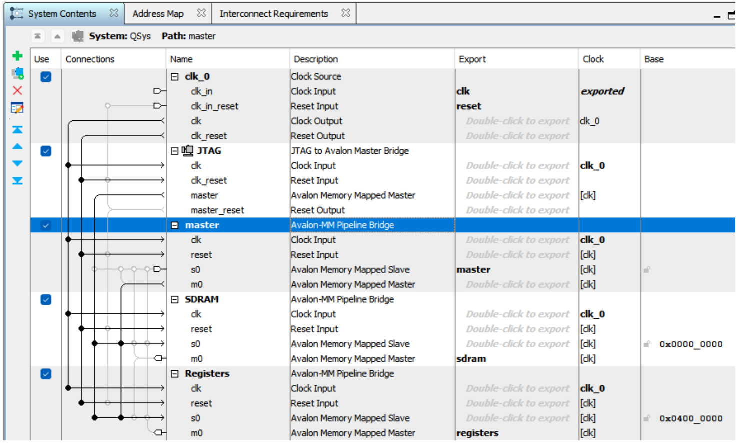
Messages

Type	Path	Message

0 Errors, 0 Warnings

Platform Designer

Practical `06 - Platform Designer` introduces the Platform Designer



System Contents | Address Map | Interconnect Requirements

System: QSys **Path:** master

Use	Connections	Name	Description	Export	Clock	Base
✓		**clk_0**	Clock Source			
		clk_in	Clock Input	**clk**	***exported***	
		clk_in_reset	Reset Input	**reset**		
		clk	Clock Output	*Double-click to export*	clk_0	
		clk_reset	Reset Output	*Double-click to export*		
✓		**JTAG**	JTAG to Avalon Master Bridge			
		clk	Clock Input	*Double-click to export*	**clk_0**	
		clk_reset	Reset Input	*Double-click to export*		
		master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
		master_reset	Reset Output	*Double-click to export*		
✓		**master**	Avalon-MM Pipeline Bridge			
		clk	Clock Input	*Double-click to export*	**clk_0**	
		reset	Reset Input	*Double-click to export*	[clk]	
		s0	Avalon Memory Mapped Slave	**master**	[clk]	
		m0	Avalon Memory Mapped Master	*Double-click to export*	[clk]	
✓		**SDRAM**	Avalon-MM Pipeline Bridge			
		clk	Clock Input	*Double-click to export*	**clk_0**	
		reset	Reset Input	*Double-click to export*	[clk]	
		s0	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0000
		m0	Avalon Memory Mapped Master	**sdram**	[clk]	
✓		**Registers**	Avalon-MM Pipeline Bridge			
		clk	Clock Input	*Double-click to export*	**clk_0**	
		reset	Reset Input	*Double-click to export*	[clk]	
		s0	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0400_0000
		m0	Avalon Memory Mapped Master	**registers**	[clk]	

Outline

Memory-Mapped Bus

Advanced Clocking

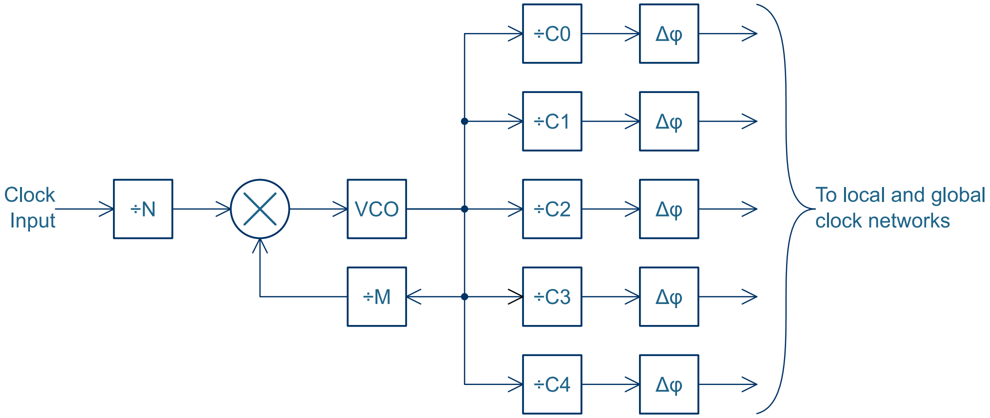
Clock Domains

Pipelines

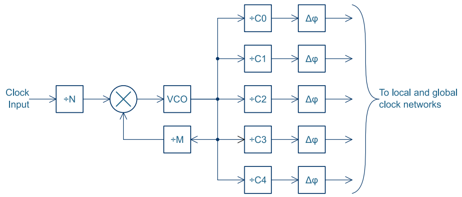
Streaming Processors

Flow Control

Phase-locked Loop

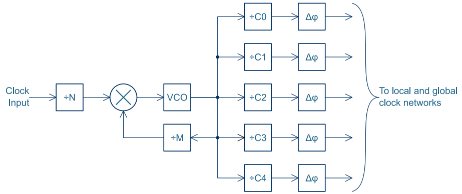


Phase-locked Loop



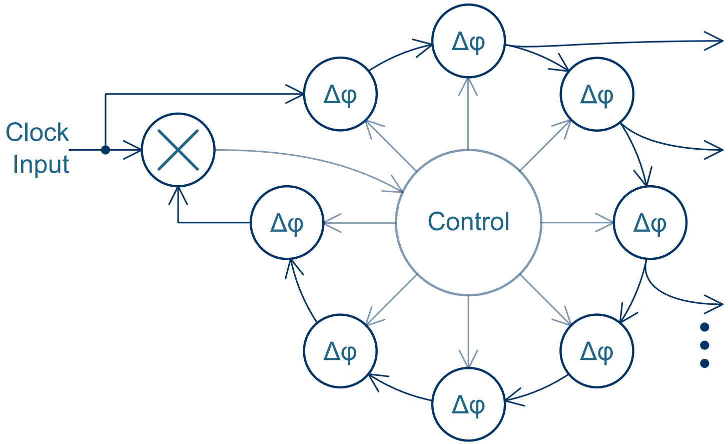
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications should avoid PLLs due to high phase noise

PLL Compensation

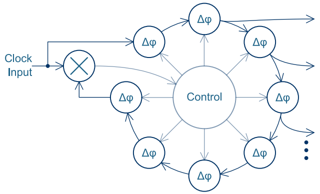


- **Direct**: No compensation, which results in better jitter performance
- **Normal**: The clock at the register is phase-aligned with the input clock pin
- **Source Synchronous**: The PLL ensures that the data delay from pin to register and the apparent clock delay from pin to register is the same
- **Zero-Delay Buffer**: The output clock pin is phase-aligned with the input clock pin

Delay-locked Loop

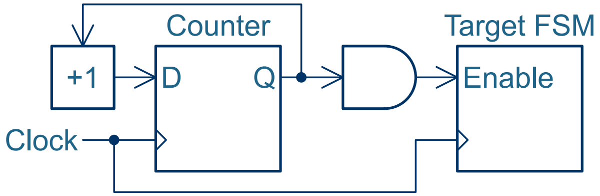


Delay-locked Loop

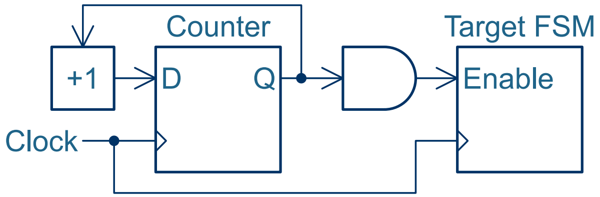


- Useful for performing phase compensation when interfacing to fast peripherals
- All output clocks can be considered to be in the same clock domain as the input clock

Clock-Enable Generated

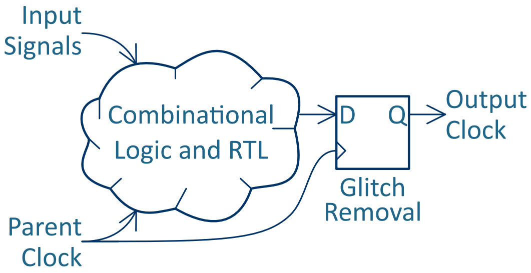


Clock-Enable Generated

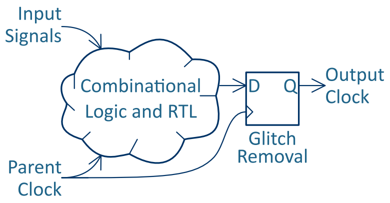


- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Ripple and Gated Clocks

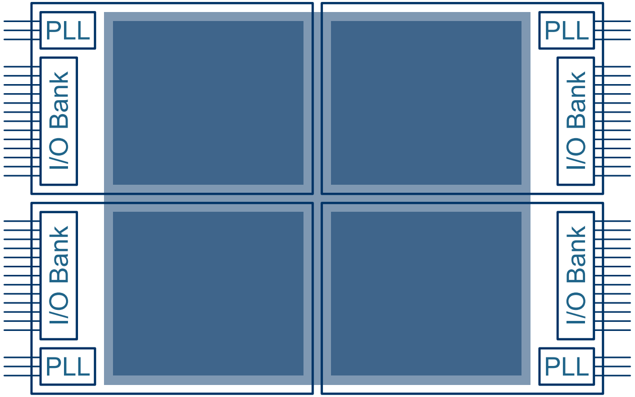


Ripple and Gated Clocks

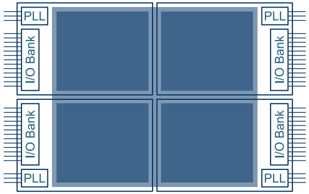


- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Clock Regions



Clock Regions



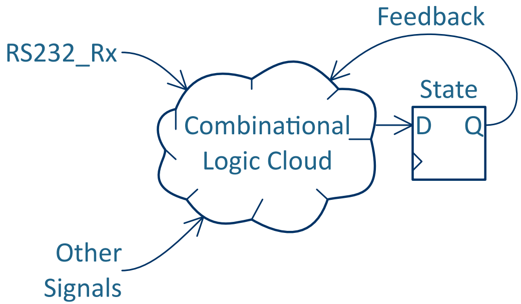
- FPGAs are generally organised into regions, each with its own PLL and I/O bank
- The local clock network is more efficient (and faster) than the global network
- When laying out the PCB, keep fast I/O in the same region as their clock

Outline

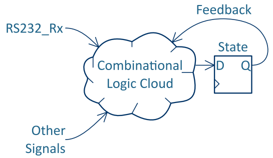
Common Mistake

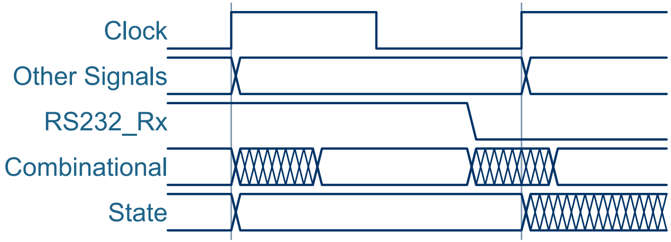
```
module RS232(
  input  ipClk, ipReset,
  input  ipRS232_Rx, // Asynchronous external signal
  output [7:0]opData
);

... // Definitions, state machine boilerplate, etc.

  Idle: begin
    if(!ipRS232_Rx) begin
      State <= Receiving;
    end
  end

  Receiving: begin
...
```

Common Mistake

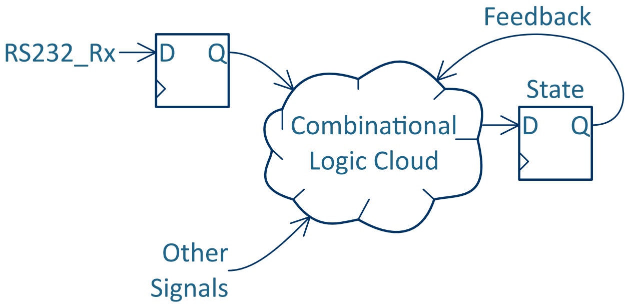


Common Mistake

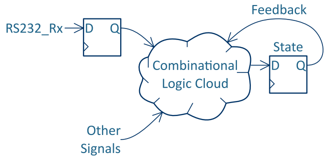


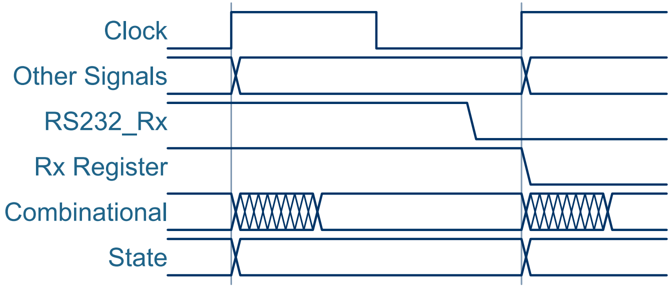


The Solution

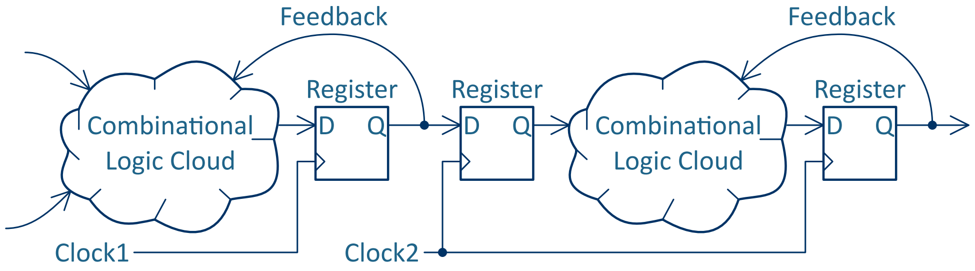


The Solution

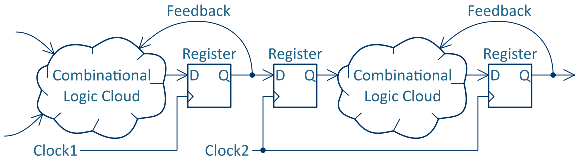




Register Chain

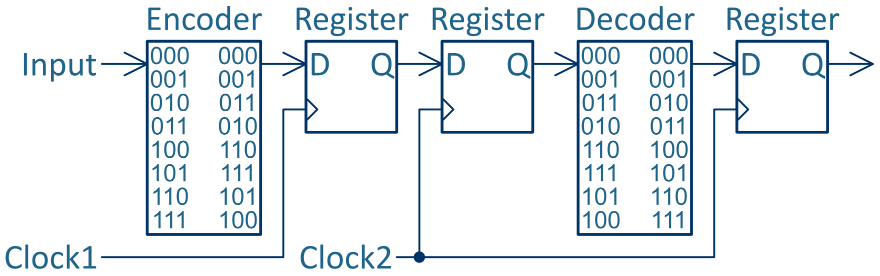


Register Chain

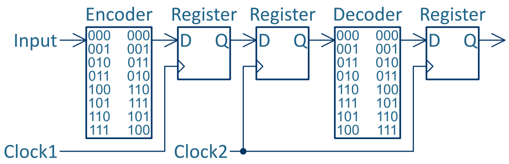


- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Gray Coding

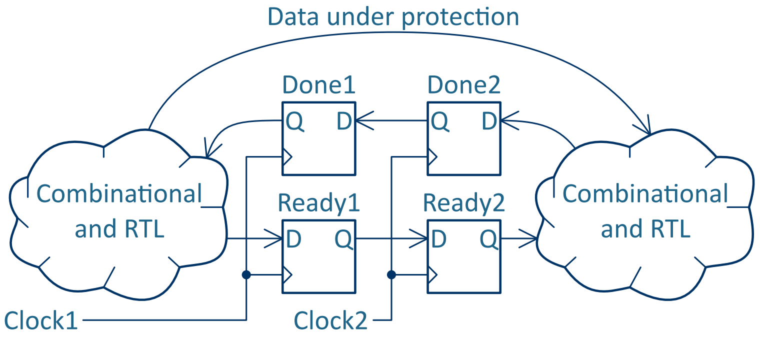

Encoder
Register
Register
Decoder
Register
Input
000 000
001 001
010 011
011 010
100 110
101 111
110 101
111 100
D Q
D Q
000 000
001 001
011 010
010 011
110 100
111 101
101 110
100 111
D Q
Clock1
Clock2

Gray Coding

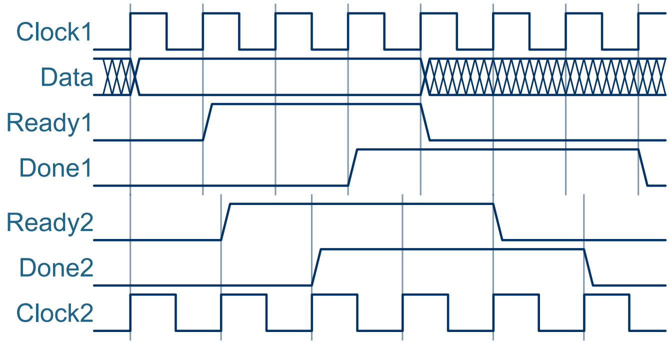


- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Works in both directions (fast to slow / slow to fast)

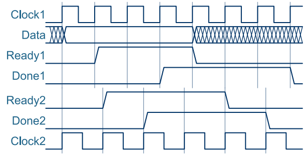
Hand-Shaking



Hand-Shaking

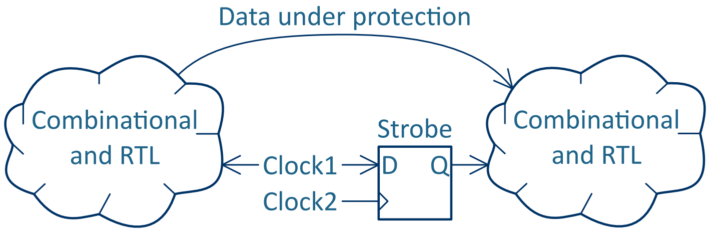


Hand-Shaking

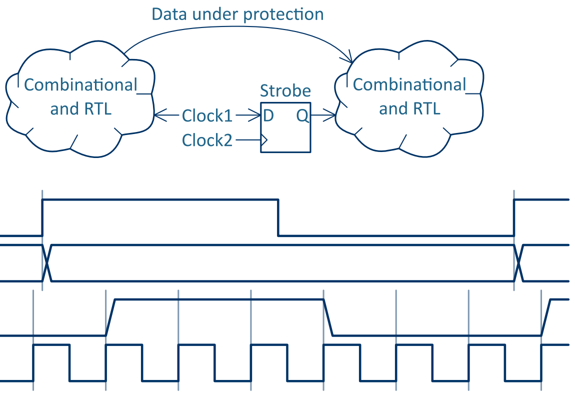


- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

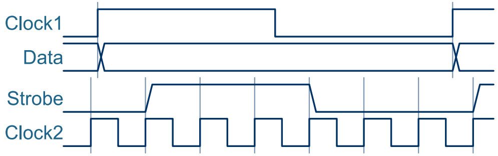
Strobe-based



Strobe-based

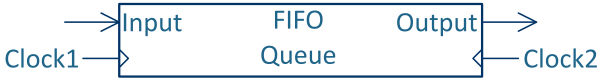


Strobe-based



- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

FIFO Queues



- Flow-control signals are crossed separately
- Control signal crossing latency can be hidden in the length of the queue
- Mixed-width RAM can be used to keep the data-rate constant across different frequencies (especially useful in DDR-based external memory)

Clock Groups

- Unrelated clock domains must be kept separate in the timing constraints as well

```
set_clock_groups -logically_exclusive \
  -group [get_clocks ADC_CLK_10]       \
  -group [get_clocks MAX10_CLK1_50]    \
  -group [get_clocks MAX10_CLK2_50]    \
  -group [get_clocks {opClk_SDRAM *altpll_component*}]
```

THE RADAR MASTERS COURSE

Coffee Break...

Outline

Memory-Mapped Bus

Advanced Clocking

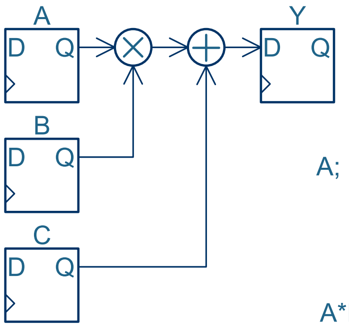
Clock Domains

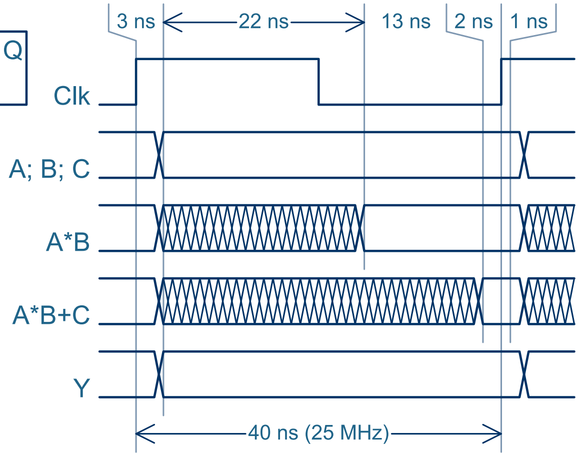
Pipelines

Streaming Processors

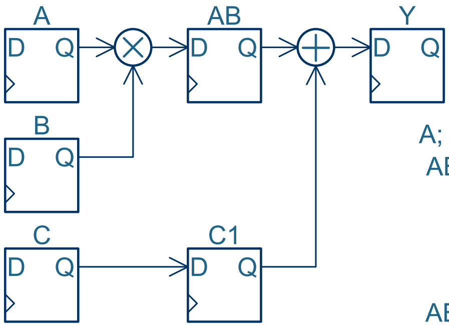
Flow Control

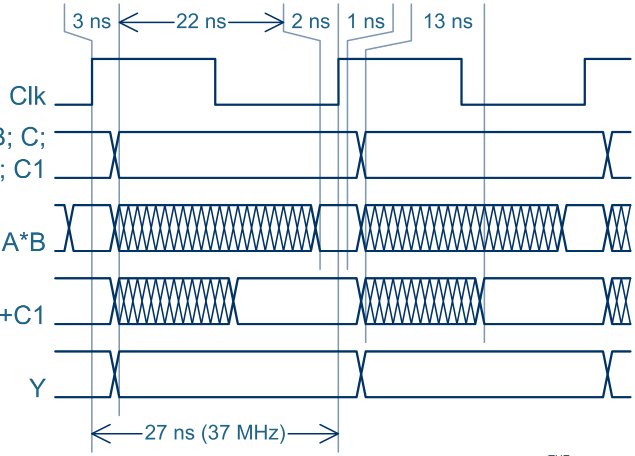
Simple Pipelines





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
```

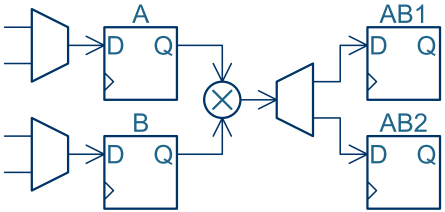
Simple Pipelines

- 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

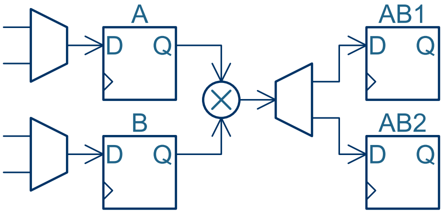
```
reg  [ 7:0]A, B;
wire [15:0]AB = A * B;

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

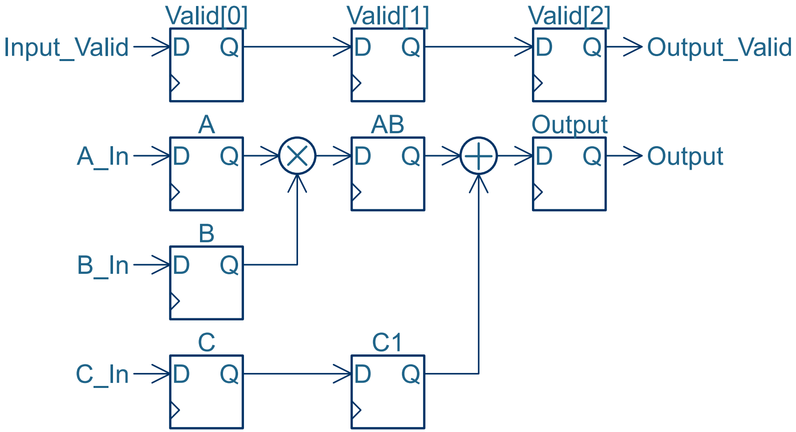
```
reg  [ 7:0]A, B;
wire [15:0]AB = A * B;

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

Stream Pipeline



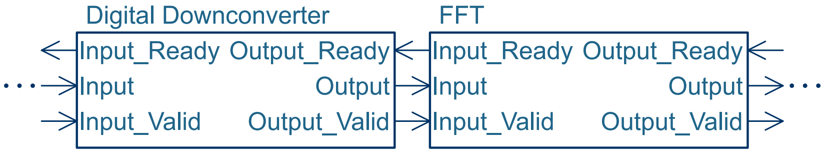
Stream Pipeline

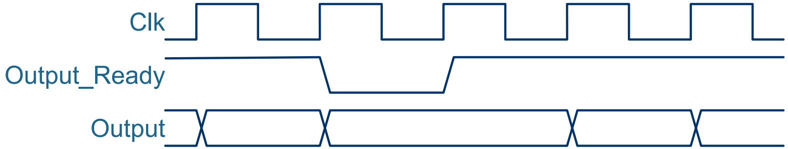
```
reg [2:0]Valid;

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`.

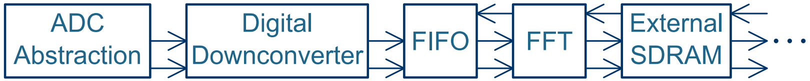
- **Do not connect an Avalon source to an AXI sink.**

Handshaking Strategies

Different buses have different handshaking strategies:

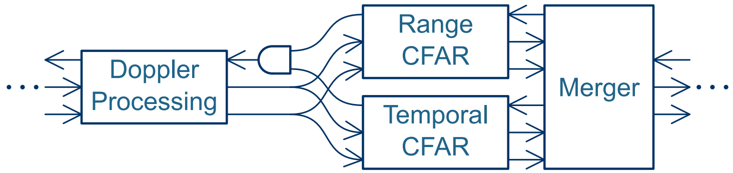
- An Avalon source is allowed to wait for the sink to be ready before asserting the `Valid`.
- An AXI sink is allowed to wait for the source to be valid before asserting its `Ready`.
- A Wishbone slave must wait for valid data from the master before asserting its `Ack`.
- **Do not connect an Avalon source to an AXI sink.**

FIFO Queues



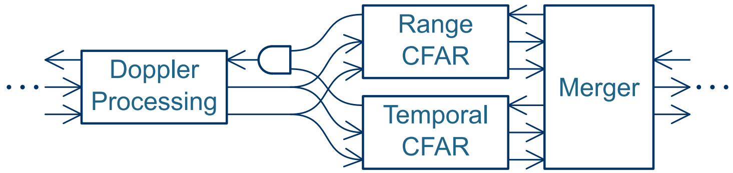
- Back-pressure makes it possible to move all the FIFO queues into a single queue
- Generally put the queue where the least amount of data is (saves resources)
- DSP chains have processing gain, so place the FIFO early in the chain

Complex Structures



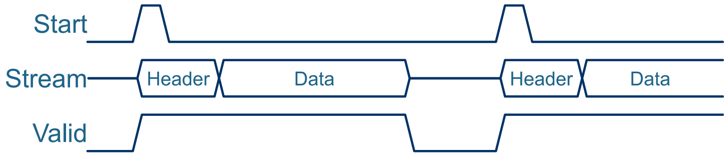
- Gate the “Valid” with the “Ready”
  (if either sink is not ready, both sinks must see a “not valid” input)
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

Complex Structures



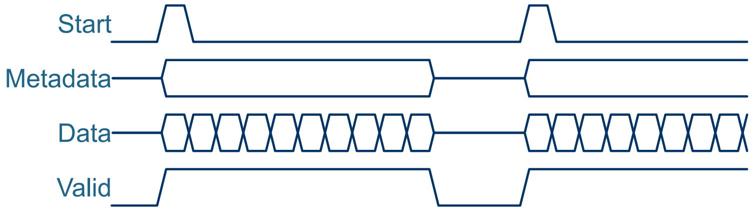
- Gate the “Valid” with the “Ready”
  (if either sink is not ready, both sinks must see a “not valid” input)
- The “Merger” can take various forms:
  - Interleave
  - Alternate (sent one logical unit from the one, then the other, then repeat)
  - Combine through calculation
  - etc.

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



- 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

Outline

Memory-Mapped Bus

Advanced Clocking

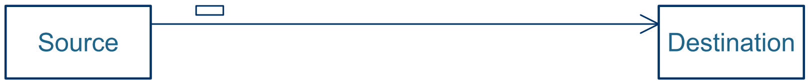
Clock Domains

Pipelines

Streaming Processors

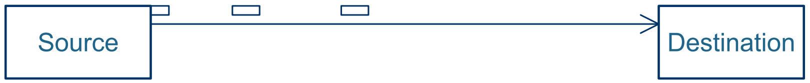
Flow Control

Fire and Forget



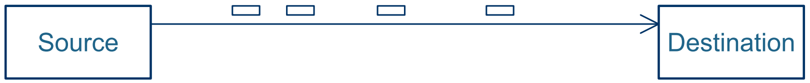
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



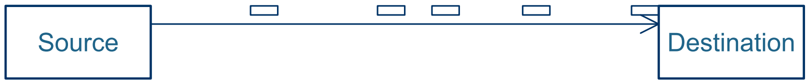
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



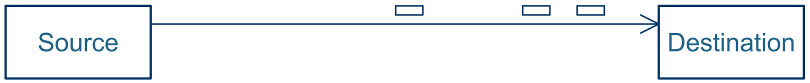
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



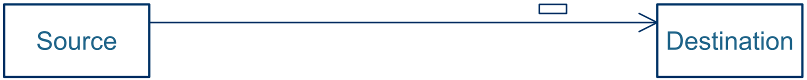
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



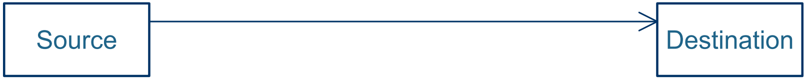
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



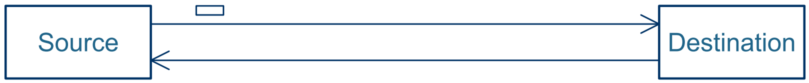
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



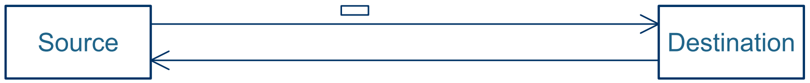
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Command-Response



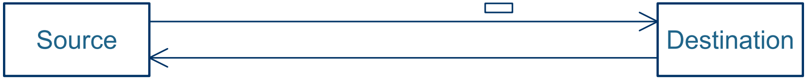
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



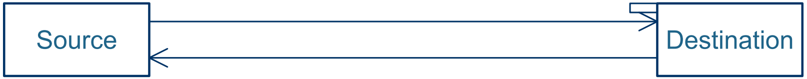
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



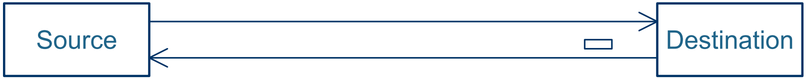
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



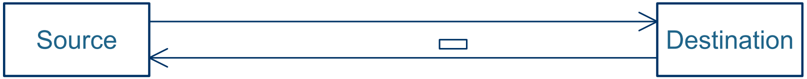
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



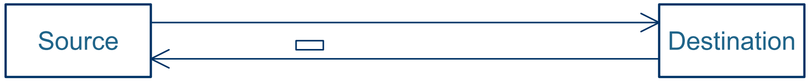
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



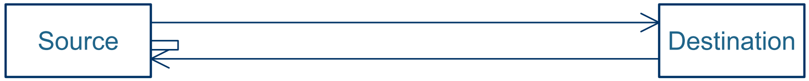
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



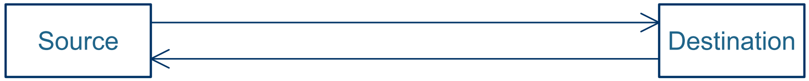
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



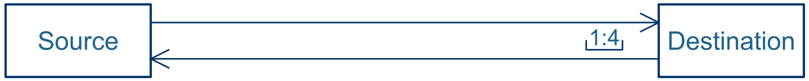
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



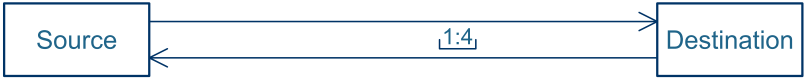
- The source sends one packet at a time
- And waits for the response from the destination

Sliding Window



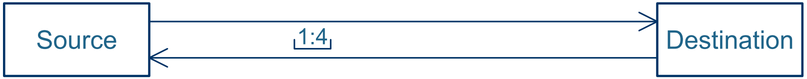
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



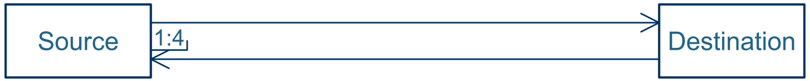
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



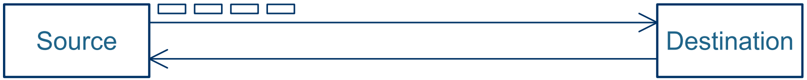
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



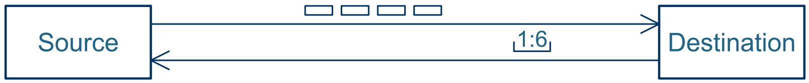
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



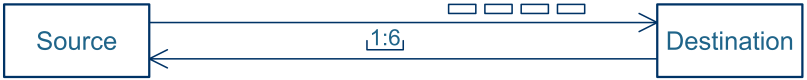
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



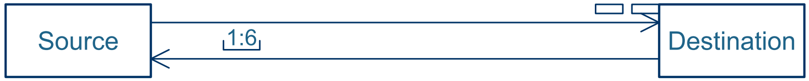
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



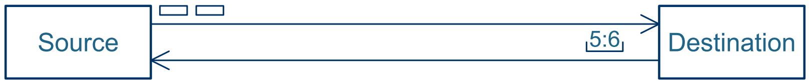
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



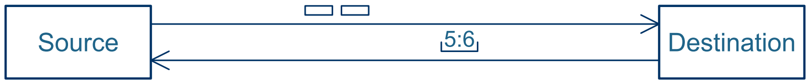
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



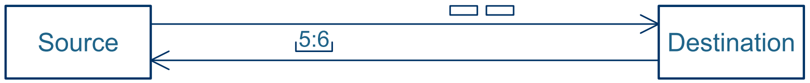
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



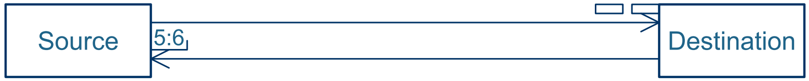
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



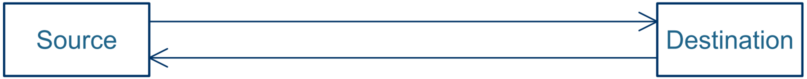
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



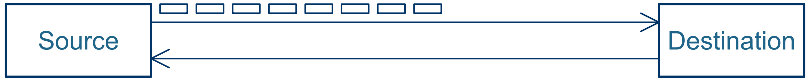
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



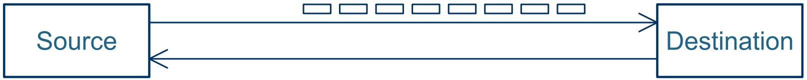
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Credit Based (1)



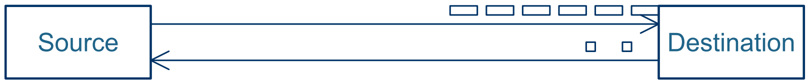
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



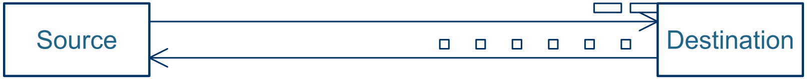
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



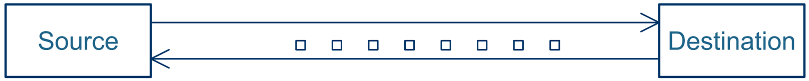
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



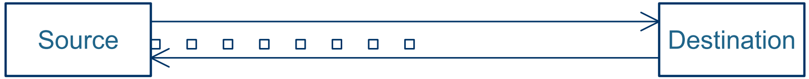
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



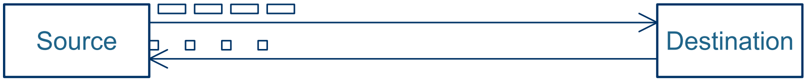
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



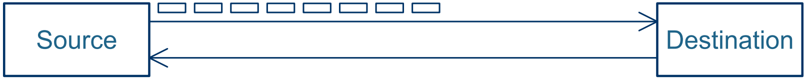
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



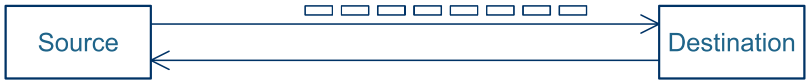
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



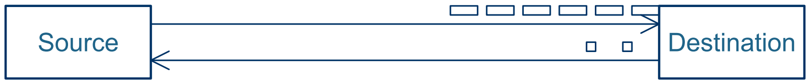
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



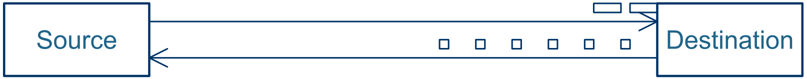
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



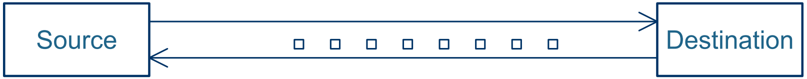
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



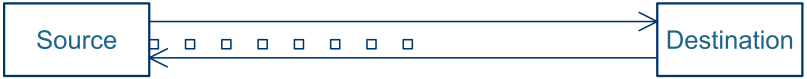
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



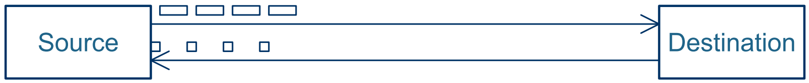
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



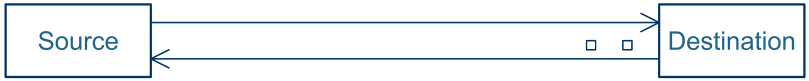
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



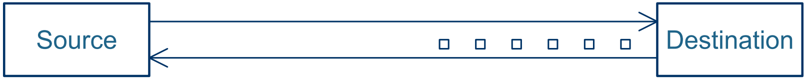
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (2)



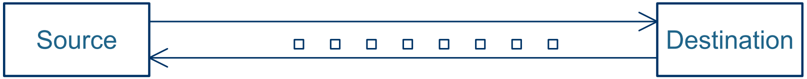
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



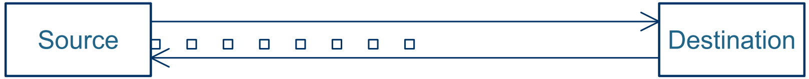
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



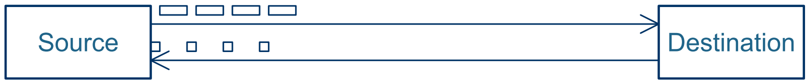
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



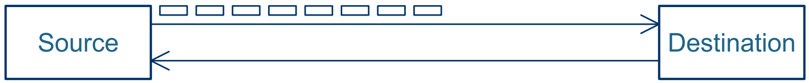
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



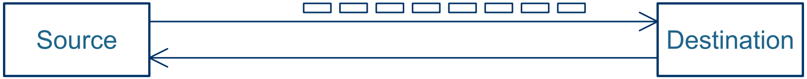
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



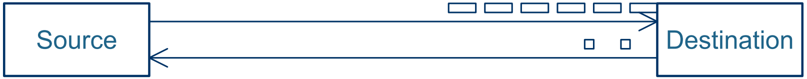
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



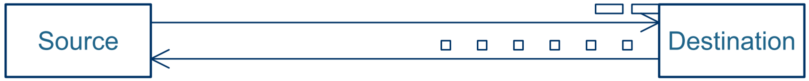
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



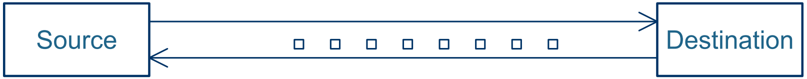
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



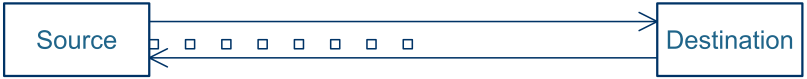
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



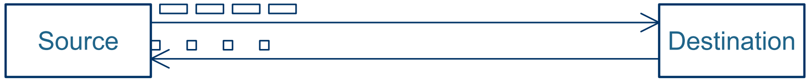
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



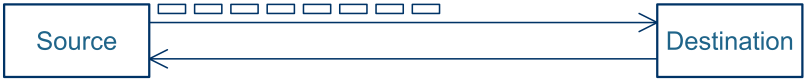
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



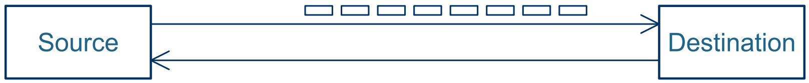
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



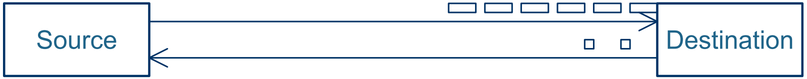
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

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 Dr John-Philip Taylor

Convened by Dr Stephen Paine

Day 3 – 11 September 2024