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 6 – 4 May 2022

Outline

Outline

Advanced Timing Constraints

FIR Filters

Practical – FIR Filter

Projects

Tips and Tricks

Conclusion

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

THE RADAR MASTERS COURSE

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

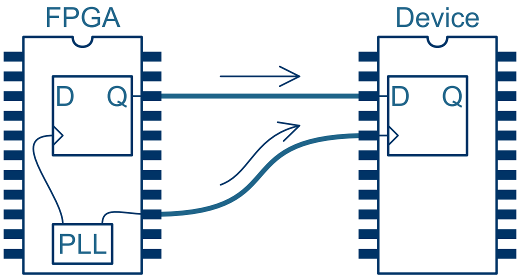
External Timing Requirements

- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing Requirements

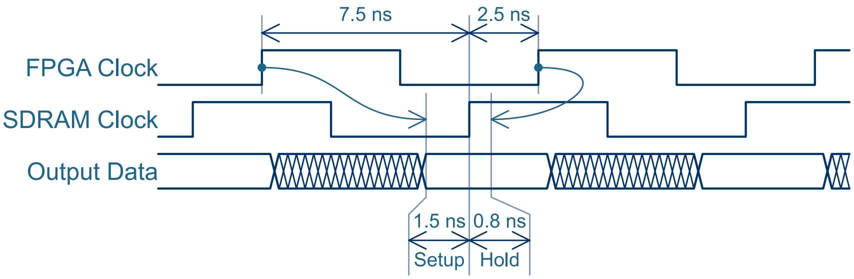
- Synopsis Design Constraints start with the ideal case:
  - There are no PCB trace delays
  - The external setup requirement is zero
  - The external hold requirement is zero
  - The external propagation delay is zero
- Increase the maximum output delay for external setup timing
- Decrease the minimum output delay for external hold timing
- Specify the minimum and maximum input delays according to the external propagation delay parameters
- Worsen the situation with PCB trace delays and uncertainties (clock jitter, manufacturing tolerances, etc.)

External Timing – Output



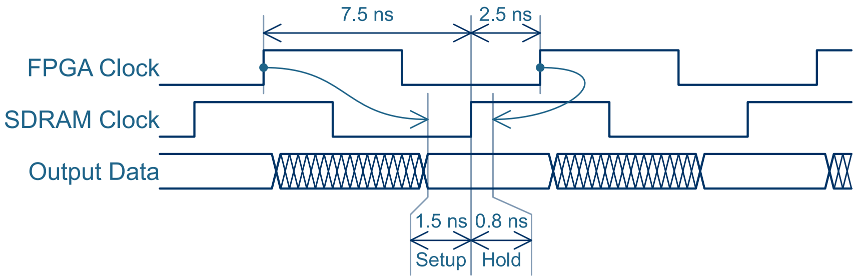
- FPGA internal delays and PCB trace delays cancel
- The important parameters are:
  - Setup and hold times of the external device
  - Clock jitter and other uncertainties
- Shift the external clock to ease the hold margin

External Timing – Output



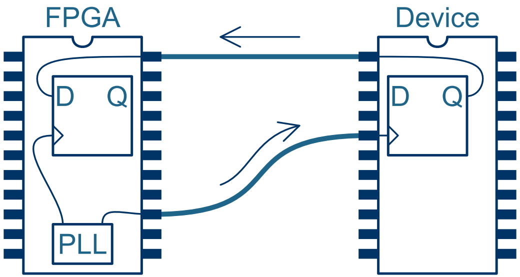
- FPGA internal delays and PCB trace delays cancel
- The important parameters are:
  - Setup and hold times of the external device
  - Clock jitter and other uncertainties
- Shift the external clock to ease the hold margin

External Timing – Output



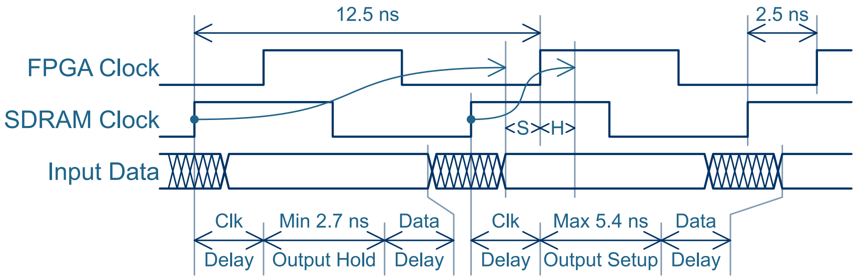
```
# Suppose 100 ps uncertainty
set_output_delay -max -clock DRAM_CLK  1.6 [get_ports opDRAM*]
set_output_delay -min -clock DRAM_CLK -0.9 [get_ports opDRAM*]
set_output_delay -max -clock DRAM_CLK  1.6 [get_ports bpDRAM*]
set_output_delay -min -clock DRAM_CLK -0.9 [get_ports bpDRAM*]
```

External Timing – Input



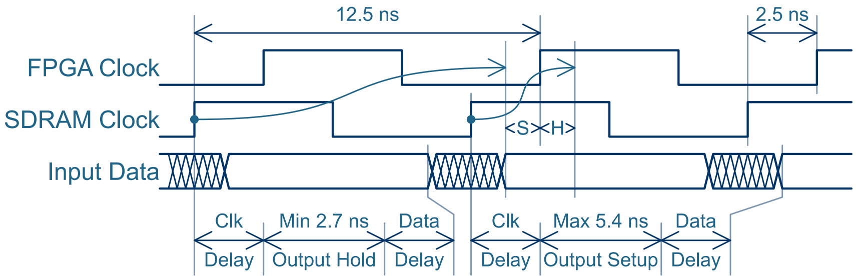
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



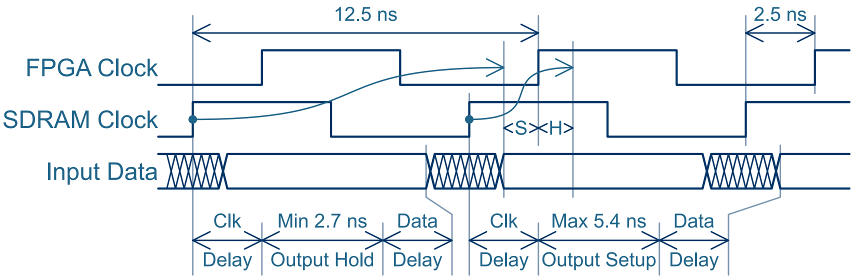
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



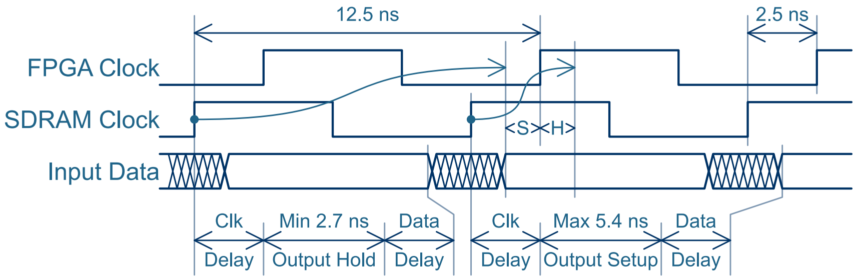
- Large FPGA internal delays and PCB trace delays
- Shift the external clock to ease the setup margin
- Multi-cycle requirement on the setup path (otherwise Quartus uses the 2.5 ns path) – the minimum propagation delay of 2.7 ns makes the 2.5 ns clock shift safe

External Timing – Input



```
set_multicycle_path                              \
  -from [get_clocks {DRAM_CLK}]                  \
  -to   [get_clocks {*altpll_0*clk[0]}]          \
  -setup 2
```

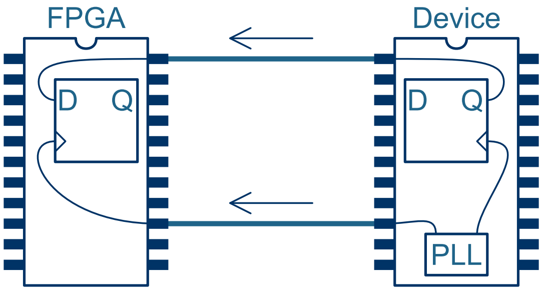
External Timing – Input



```
# Suppose 100 ps uncertainty and 200 ps PCB delay (each way)
set_input_delay -max -clock DRAM_CLK 5.9 [get_ports bpDRAM*]
set_input_delay -min -clock DRAM_CLK 3.0 [get_ports bpDRAM*]
```

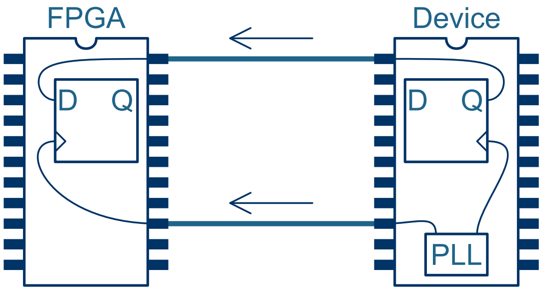
THE RADAR MASTERS COURSE

External Timing – DDR ADC



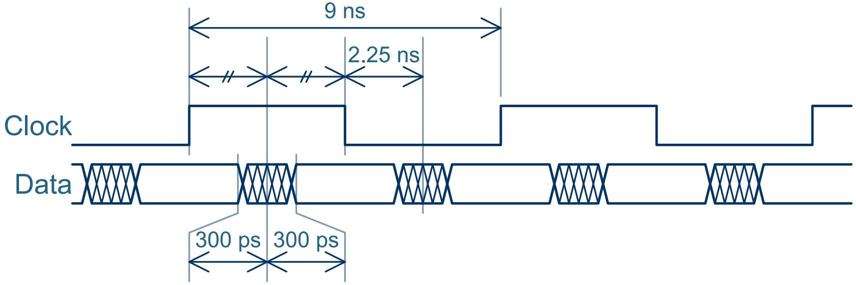
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC



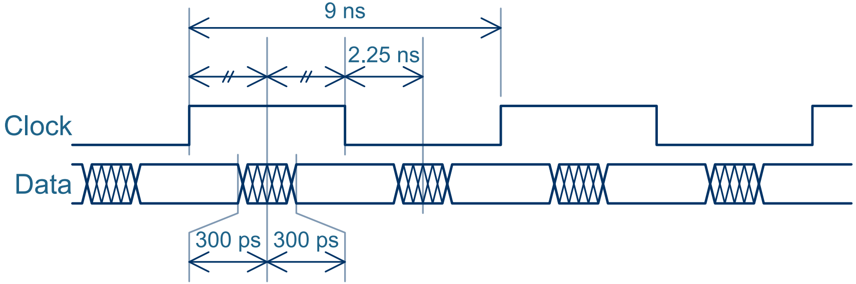
- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

External Timing – DDR ADC



- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

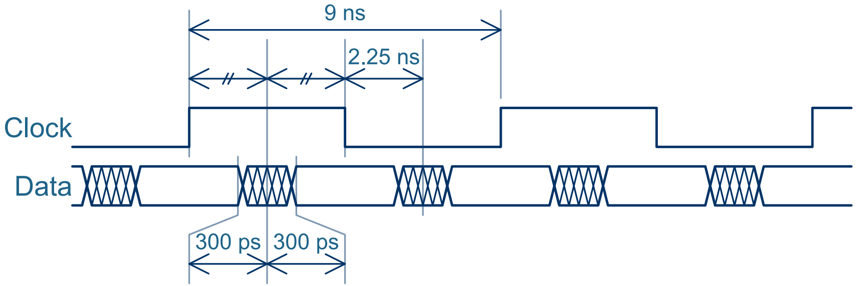
External Timing – DDR ADC



- The clock is sourced by the external device
- The PCB trace delays cancel
- The data is centre-aligned
- Quartus automatically does input port alignment;
  Xilinx must be manually tuned: use the IDELAY primitive

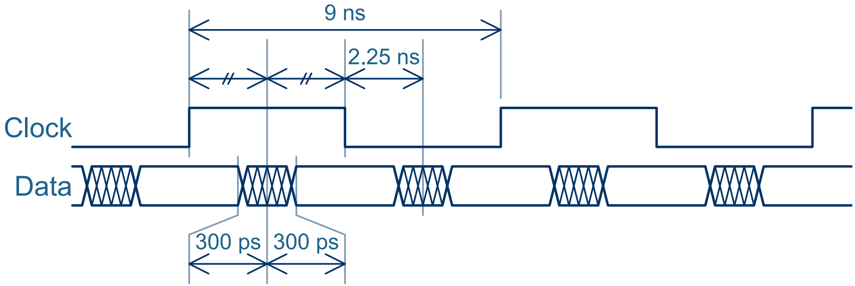
THE RADAR MASTERS COURSE

External Timing – DDR ADC



```
# Suppose 100 ps uncertainty
create_clock    -name  ADC_DCO \
  -period 9 [get_ports ipADC_DCLK_P]
set_input_delay -clock ADC_DCO \
  -min 1.85 [get_ports ipADC_CH*]
set_input_delay -clock ADC_DCO \
  -max 2.65 [get_ports ipADC_CH*]
```

External Timing – DDR ADC



```
set_input_delay -clock ADC_DCO -clock_fall \
  -min 1.85 [get_ports ipADC_CH*] -add_delay
set_input_delay -clock ADC_DCO -clock_fall \
  -max 2.65 [get_ports ipADC_CH*] -add_delay
```

Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
IOBUF PORT "opDRAM*" IO_TYPE=LVCMOS33 ;
IOBUF PORT "bpDRAM*" IO_TYPE=LVCMOS33 ;
IOBUF PORT "clkDRAM" IO_TYPE=LVCMOS33 ;
```

Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
IOBUF PORT "opDRAM*" IO_TYPE=LVCMOS33 DRIVE=8 ;
IOBUF PORT "bpDRAM*" IO_TYPE=LVCMOS33 DRIVE=8 ;
IOBUF PORT "clkDRAM" IO_TYPE=LVCMOS33 DRIVE=8 ;
```

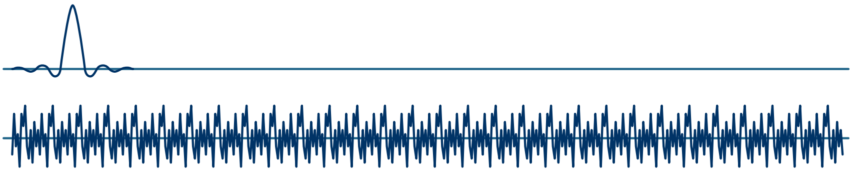
Timing-related Pin Attributes

- Use the correct I/O standard
- Set the correct output current
- Set the pin capacitance

```
OUTPUT PORT "opDRAM*"     LOAD 3.8 pF ;
OUTPUT PORT "bpDRAM_DQ*" LOAD 6.0 pF ;
OUTPUT PORT "CLK_DRAM"    LOAD 3.5 pF ;
```

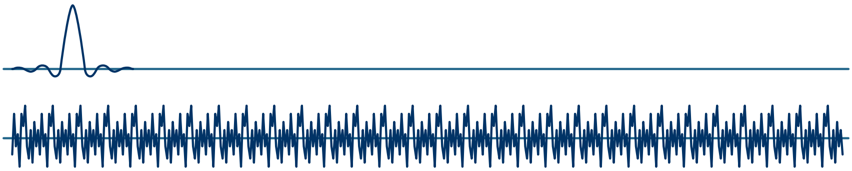
Outline

FIR Filter – Concept



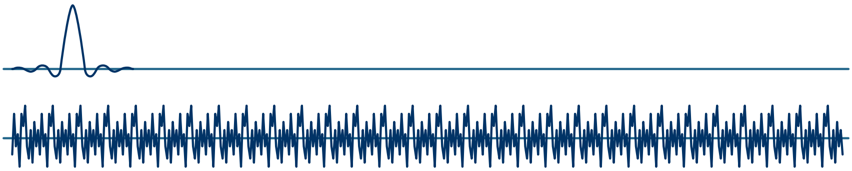
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



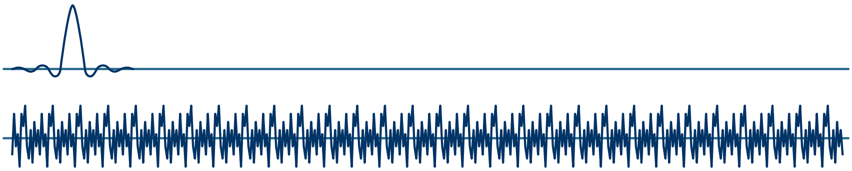
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



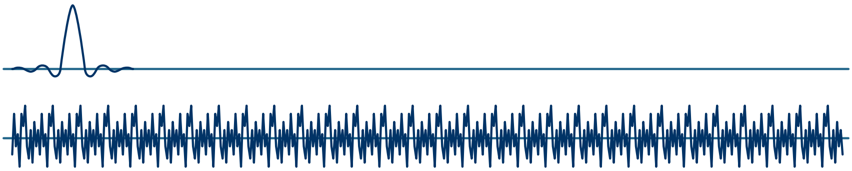
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



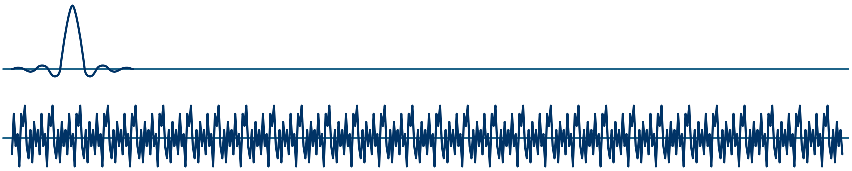
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



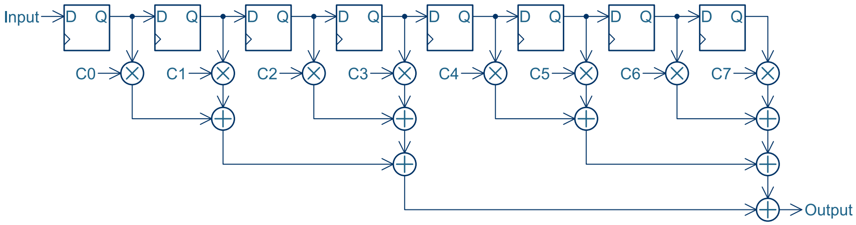
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



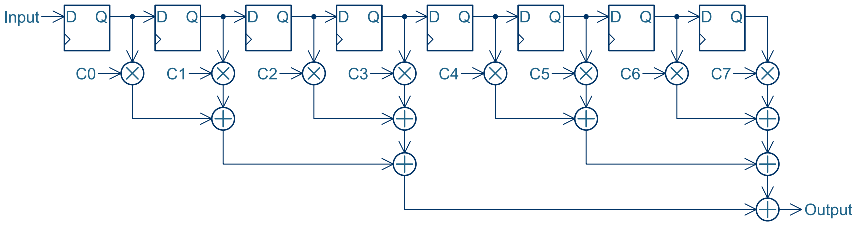
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Implementation



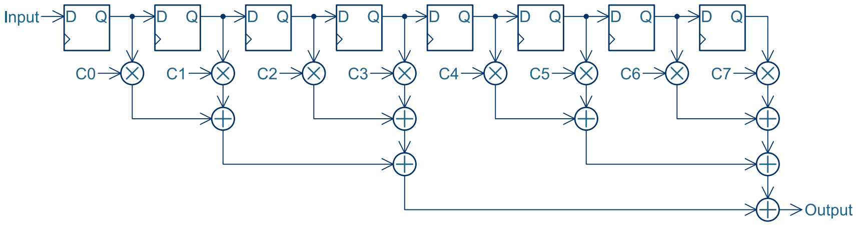
- Move the signal instead of the impulse-response
- To check the direction, inject an impulse (which should produce the impulse-response at the output)
- Pipeline the adder tree to increase the maximum clock frequency

FIR Filter – Implementation



- Move the signal instead of the impulse-response
- To check the direction, inject an impulse (which should produce the impulse-response at the output)
- Pipeline the adder tree to increase the maximum clock frequency

FIR Filter – Implementation



- Move the signal instead of the impulse-response
- To check the direction, inject an impulse (which should produce the impulse-response at the output)
- Pipeline the adder tree to increase the maximum clock frequency

Architecture Design

- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don't need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: sample rate, clock speed, power requirements, throughput requirements, decimation (if any), available resources, etc...

Architecture Design

- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don't need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: sample rate, clock speed, power requirements, throughput requirements, decimation (if any), available resources, etc...

Architecture Design

- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don't need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: sample rate, clock speed, power requirements, throughput requirements, decimation (if any), available resources, etc...

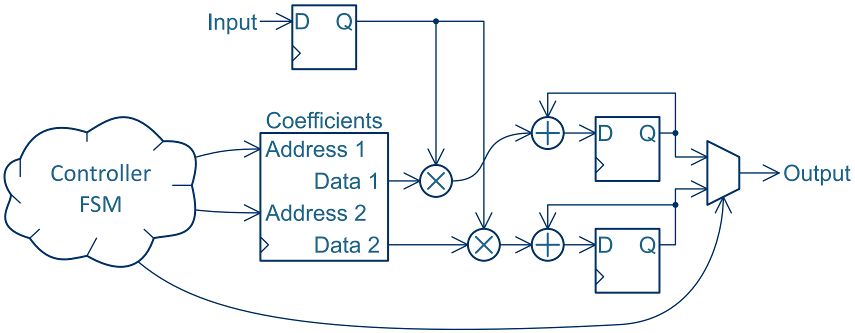
Architecture Design

- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don't need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: sample rate, clock speed, power requirements, throughput requirements, decimation (if any), available resources, etc...

Architecture Design

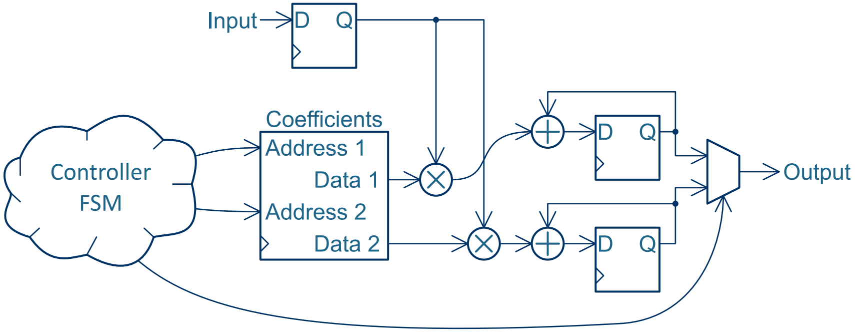
- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don’t need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: sample rate, clock speed, power requirements, throughput requirements, decimation (if any), available resources, etc...

Decimation



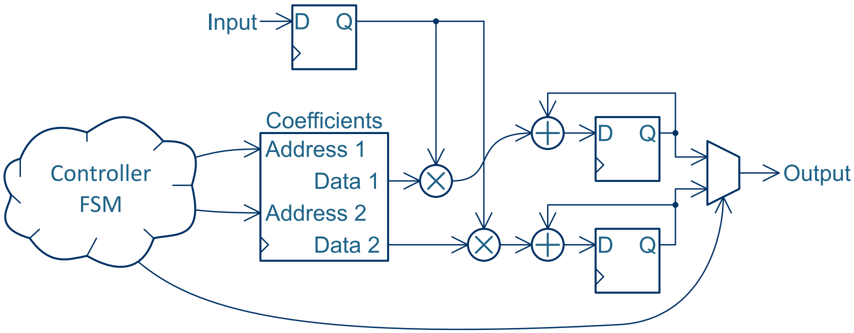
- This example decimates by $N/2$:
  a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

Decimation



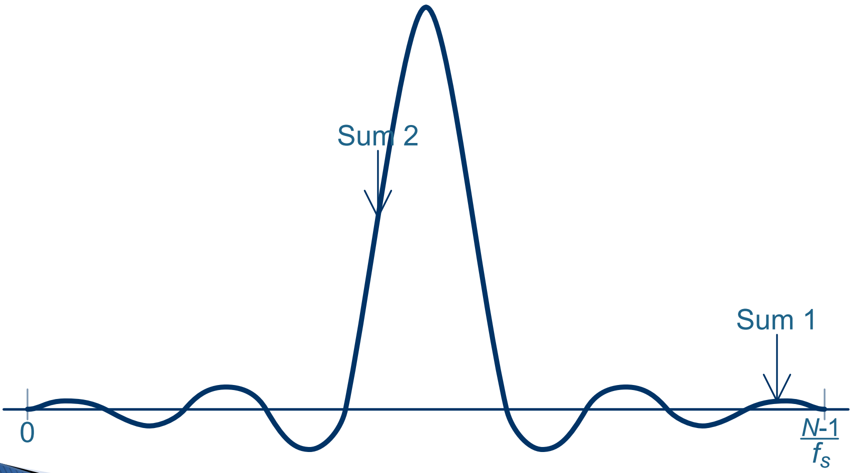
- This example decimates by $N/2$: a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

Decimation

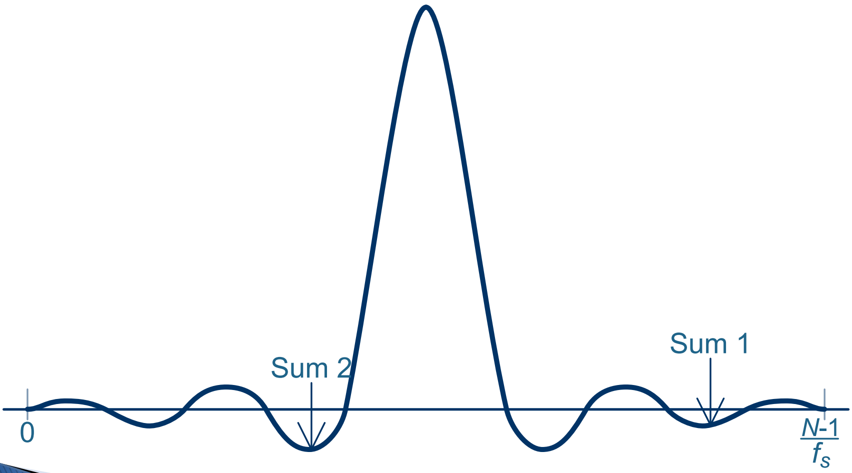


- This example decimates by $N/2$:
  a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

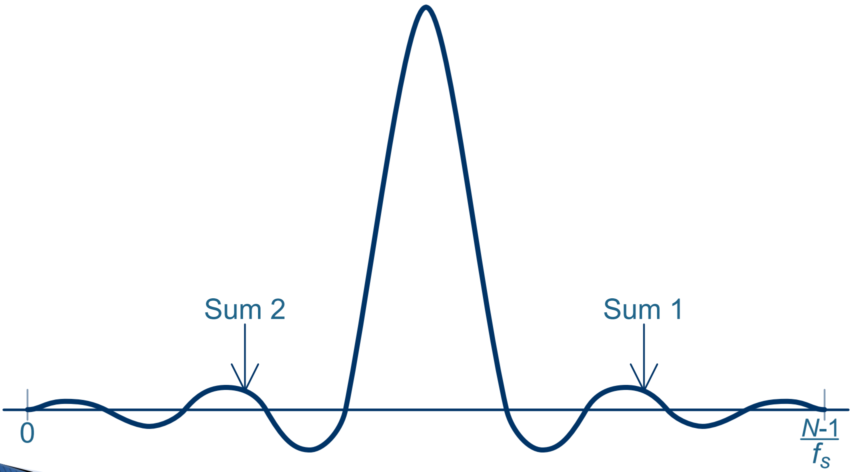
Decimation



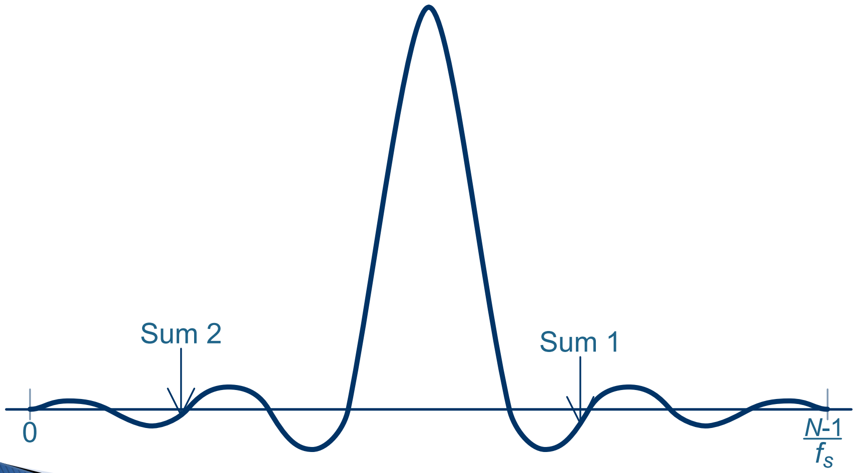
Decimation



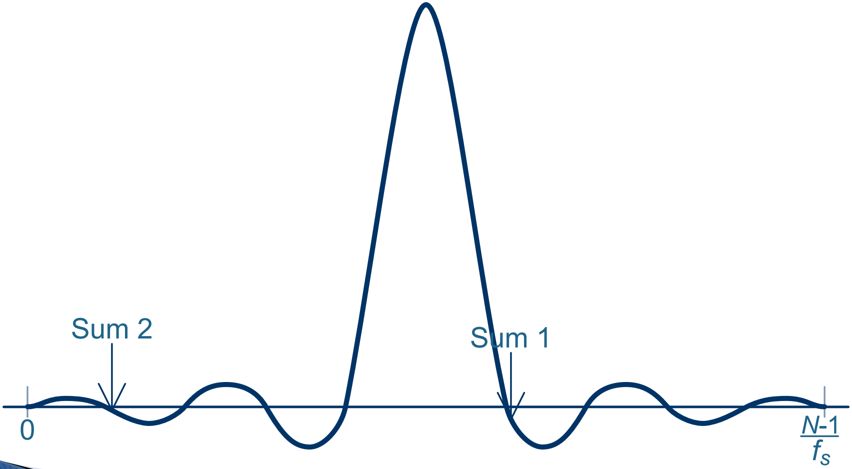
Decimation



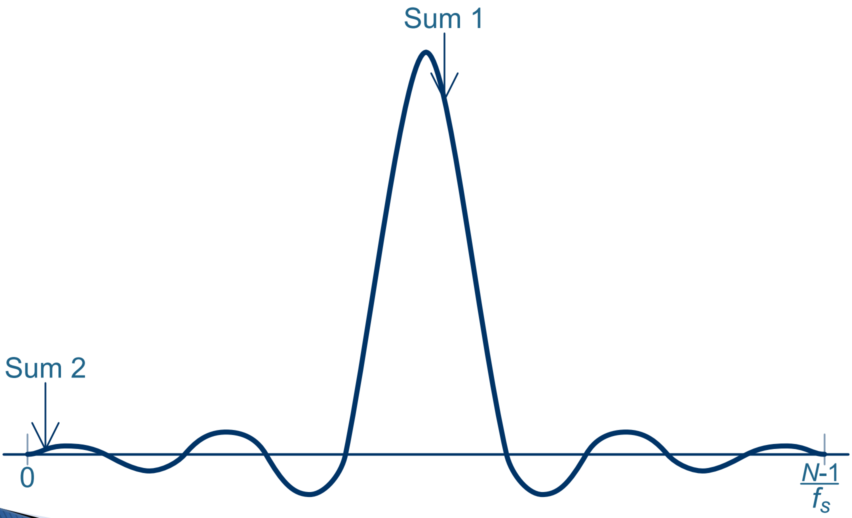
Decimation



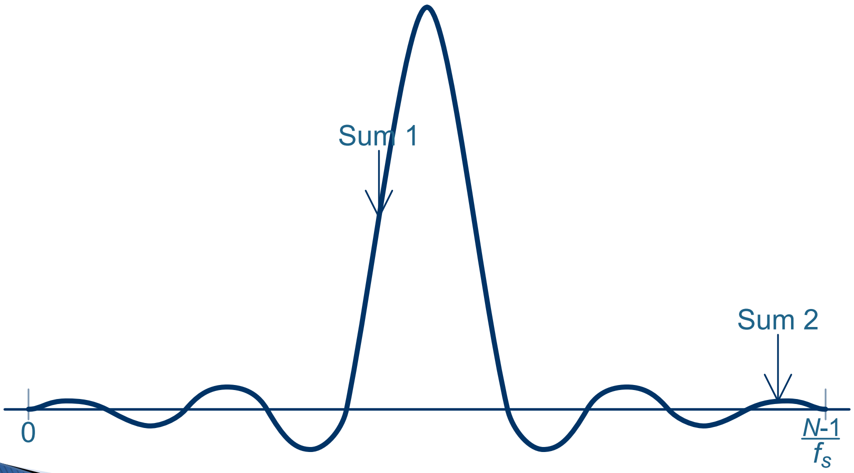
Decimation



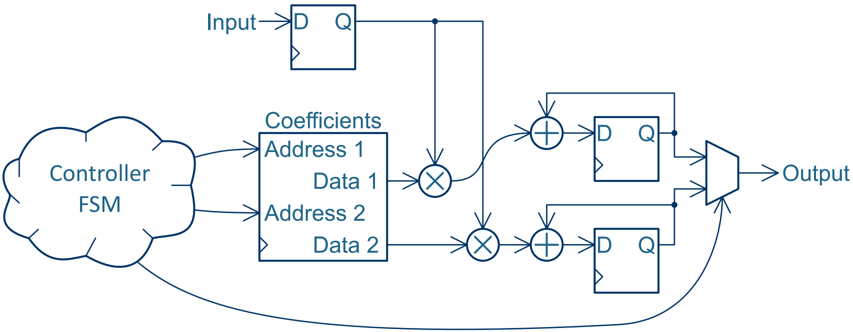
Decimation



Decimation

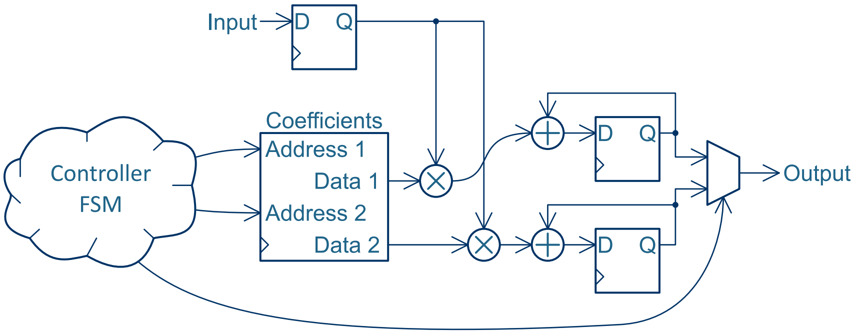


Faster System Clock



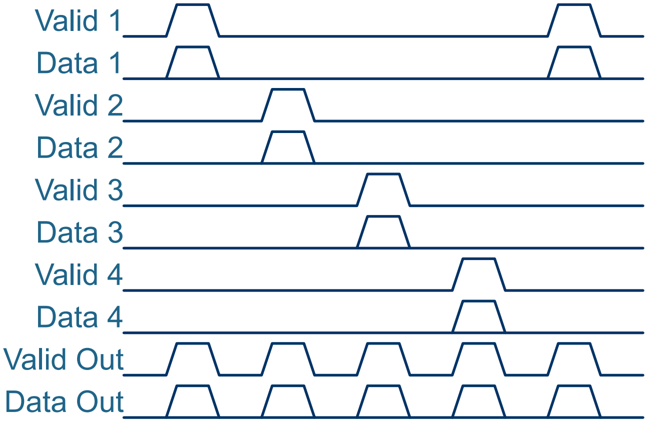
- If the sample clock is lower than the system clock, this same architecture can decimate by less:
  - Keep more than 2 sums – sometimes more efficient to keep these in BRAM as well

Faster System Clock

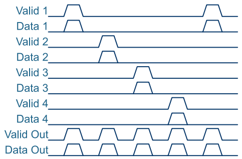


- If the sample clock is lower than the system clock, this same architecture can decimate by less:
- Keep more than 2 sums – sometimes more efficient to keep these in BRAM as well

Combining FIR Filter Units

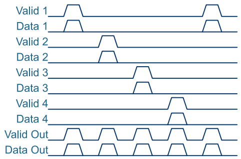


FIR Filter



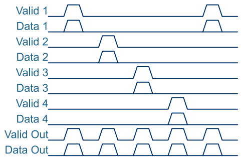
- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

FIR Filter



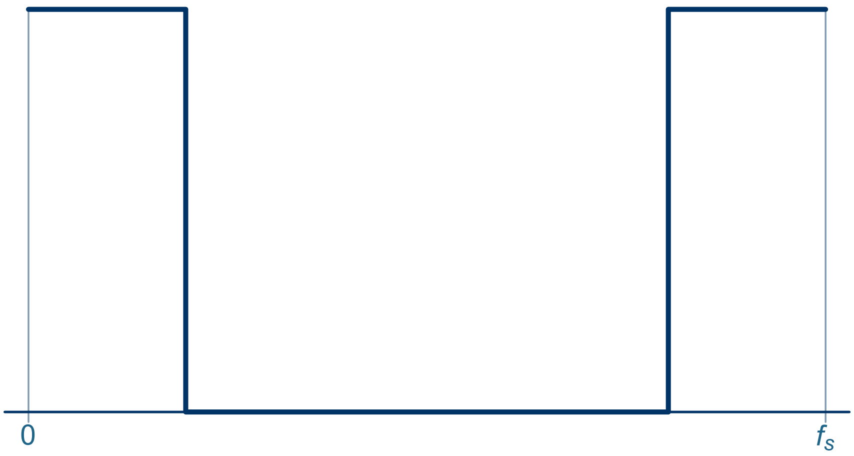
- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

FIR Filter

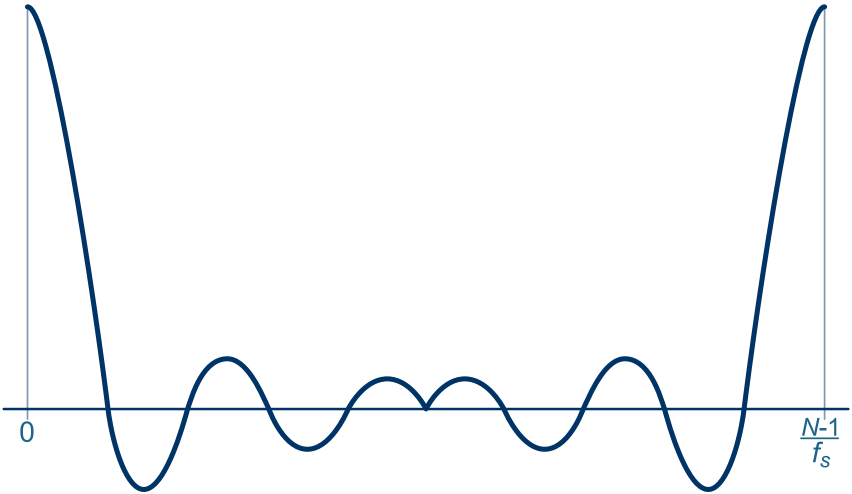


- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

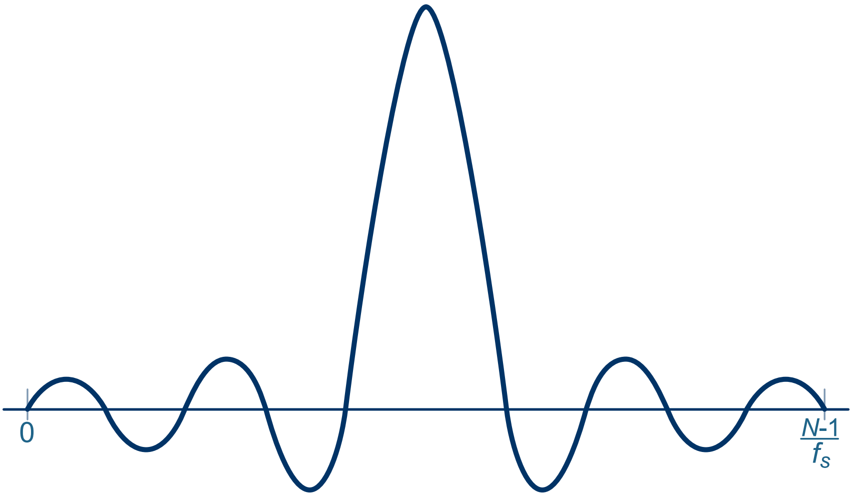
FIR Filter – Desired Response



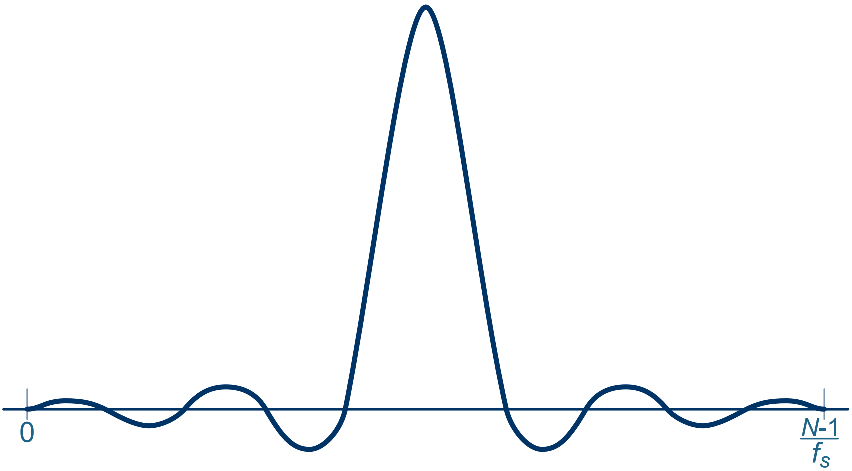
FIR Filter – IFFT (real part)



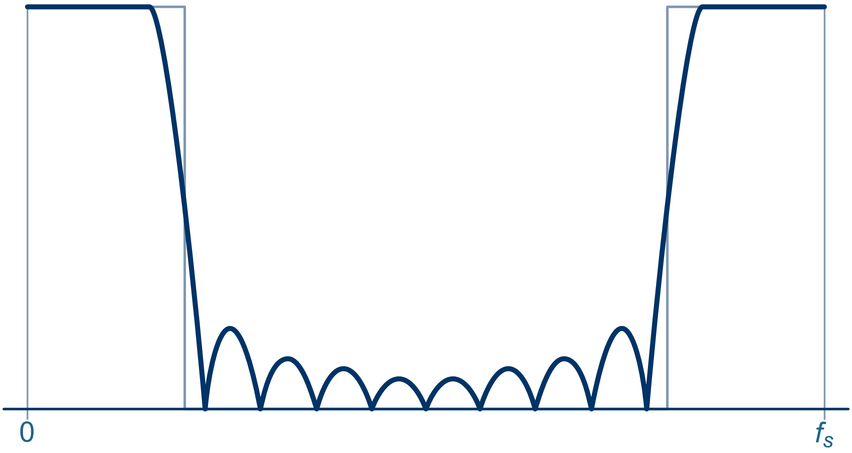
FIR Filter – Time-shifted



FIR Filter – Windowed



FIR Filter – Actual Response



(Zero-pad to see the side-lobes)

FIR Filter Implementation

- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

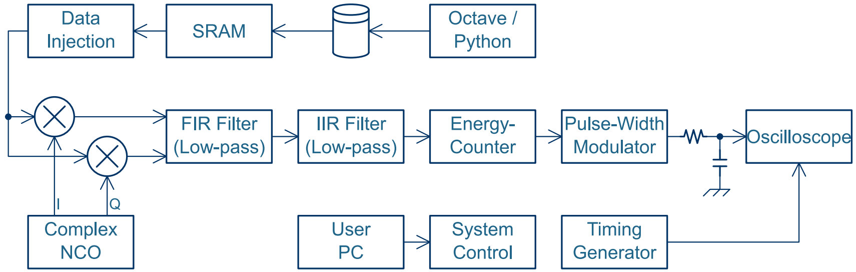
- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

FIR Filter Implementation

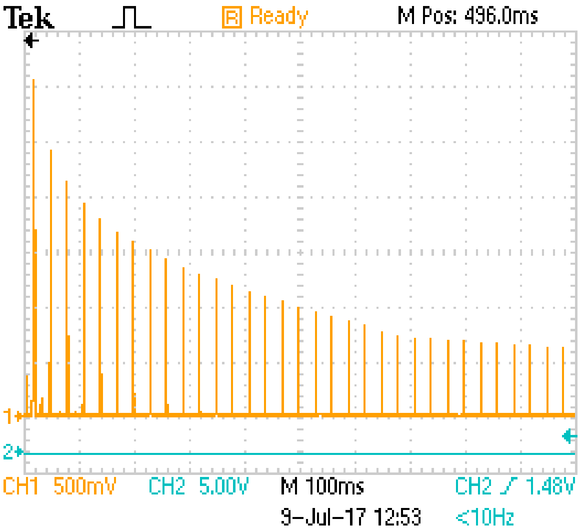
- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the sub-sampling rate to N/8
- Verify on FPGA

Outline

Practical – FIR Filter



Practical


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

Coffee Break...

Outline

Advanced Timing Constraints

FIR Filters

Practical – FIR Filter

Projects

Tips and Tricks

Conclusion

Project Overview

- Everybody must do a different project, but the projects are interlinked. Too make it more fun, make sure the system parameters are compatible across projects.
- You can propose a project: preferably in line with your current MSc research
- You need to design the DSP chain and choose appropriate system parameters
- Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC
- Your system must be controllable from the PC (typically over the UART)

Project Overview

- Everybody must do a different project
- You can propose a project: preferably in line with your current MSc research
- You need to design the DSP chain and choose appropriate system parameters
- Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC
- Your system must be controllable from the PC (typically over the UART)

Project Overview

- Everybody must do a different project
- You can propose a project: preferably in line with your current MSc research
- You need to design the DSP chain and choose appropriate system parameters: ADC sampling-rate, decimation (if any), architecture etc.
- Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC
- Your system must be controllable from the PC (typically over the UART)

Project Overview

- Everybody must do a different project
- You can propose a project: preferably in line with your current MSc research
- You need to design the DSP chain and choose appropriate system parameters: ADC sampling-rate, decimation (if any), architecture etc.
- Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC
- Your system must be controllable from the PC (typically over the UART)

Project Overview

- Everybody must do a different project
- You can propose a project: preferably in line with your current MSc research
- You need to design the DSP chain and choose appropriate system parameters: ADC sampling-rate, decimation (if any), architecture etc.
- Typically, a design will inject data from SDRAM into the DSP-chain, and then store the result in the same SDRAM, which is then read and analysed by the PC
- Your system must be controllable from the PC (typically over the UART)

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
    - Demonstrate a working FPGA-based DSP chain
    - Give a 15-minute presentation on the design and performance results
    - Submit the source-code for review
    - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
  - Demonstrate a working FPGA-based DSP chain
  - Give a 15-minute presentation on the design and performance results
  - Submit the source-code for review
  - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
  - Demonstrate a working FPGA-based DSP chain
  - Give a 15-minute presentation on the design and performance results
  - Submit the source-code for review
  - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
  - Demonstrate a working FPGA-based DSP chain
  - Give a 15-minute presentation on the design and performance results
  - Submit the source-code for review
  - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
  - Demonstrate a working FPGA-based DSP chain
  - Give a 15-minute presentation on the design and performance results
  - Submit the source-code for review
  - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

The Demonstration

- The demonstrations are scheduled for the week of 26 July – a date will be chosen at a later time
- You need to:
  - Demonstrate a working FPGA-based DSP chain
  - Give a 15-minute presentation on the design and performance results
  - Submit the source-code for review
  - Submit a project design report: nothing fancy – keep it to within 10 pages of overview, detail design and performance results.

Project Notes

- Remember that you are not designing the Radar / Communication system / etc.
- You are designing the FPGA-based processor, on a very small FPGA with limited resources
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for example: always assume that targets are slow-moving, that there are no multipath effects and that you have a low sampling-rate)
- At the same time, the project must show FPGA competence, so don't make it too trivial
- Use the tools available, including the libraries and high-level design tools, where appropriate

Project Notes

- Remember that you are not designing the Radar / Communication system / etc.
- You are designing the FPGA-based processor, on a very small FPGA with limited resources
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for example: always assume that targets are slow-moving, that there are no multipath effects and that you have a low sampling-rate)
- At the same time, the project must show FPGA competence, so don't make it too trivial
- Use the tools available, including the libraries and high-level design tools, where appropriate

Project Notes

- Remember that you are not designing the Radar / Communication system / etc.
- You are designing the FPGA-based processor, on a very small FPGA with limited resources
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for example: always assume that targets are slow-moving, that there are no multipath effects and that you have a low sampling-rate)
- At the same time, the project must show FPGA competence, so don't make it too trivial
- Use the tools available, including the libraries and high-level design tools, where appropriate

Project Notes

- Remember that you are not designing the Radar / Communication system / etc.
- You are designing the FPGA-based processor, on a very small FPGA with limited resources
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for example: always assume that targets are slow-moving, that there are no multipath effects and that you have a low sampling-rate)
- At the same time, the project must show FPGA competence, so don't make it too trivial
- Use the tools available, including the libraries and high-level design tools, where appropriate

Project Notes

- Remember that you are not designing the Radar / Communication system / etc.
- You are designing the FPGA-based processor, on a very small FPGA with limited resources
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for example: always assume that targets are slow-moving, that there are no multipath effects and that you have a low sampling-rate)
- At the same time, the project must show FPGA competence, so don’t make it too trivial
- Use the tools available, including the libraries and high-level design tools, where appropriate

Asking for Help

- I'm not on campus, but you can reach me via email
- Clearly explain what you're trying to do, and what you're struggling with
- Include your source code and, when applicable, some pictures of the architecture you're trying to implement and a test-bench to highlight the problem
- You'll find some good resources on Google, but be very careful – more often than not the people don't know what they're talking about and lead you down the wrong path

THE RADAR MASTERS COURSE

Asking for Help

- I'm not on campus, but you can reach me via email
- Clearly explain what you're trying to do, and what you're struggling with
- Include your source code and, when applicable, some pictures of the architecture you're trying to implement and a test-bench to highlight the problem
- You'll find some good resources on Google, but be very careful – more often than not the people don't know what they're talking about and lead you down the wrong path

Asking for Help

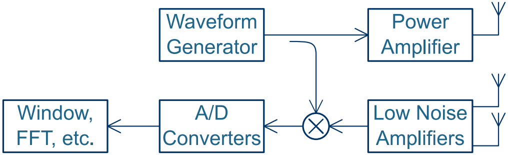
- I'm not on campus, but you can reach me via email
- Clearly explain what you're trying to do, and what you're struggling with
- Include your source code and, when applicable, some pictures of the architecture you're trying to implement and a test-bench to highlight the problem
- You'll find some good resources on Google, but be very careful – more often than not the people don't know what they're talking about and lead you down the wrong path

Asking for Help

- I'm not on campus, but you can reach me via email
- Clearly explain what you're trying to do, and what you're struggling with
- Include your source code and, when applicable, some pictures of the architecture you're trying to implement and a test-bench to highlight the problem
- You'll find some good resources on Google, but be very careful – more often than not the people don't know what they're talking about and lead you down the wrong path

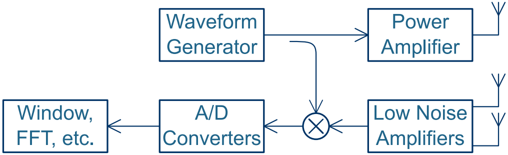
Project Ideas

Project 1 – FMCW Front-end



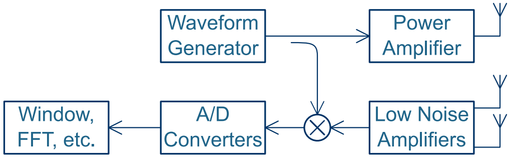
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

Project 1 – FMCW Front-end



- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

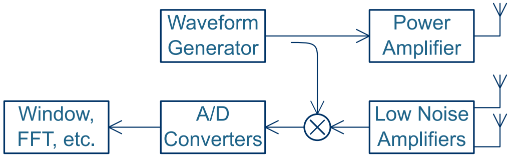
Project 1 – FMCW Front-end



- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

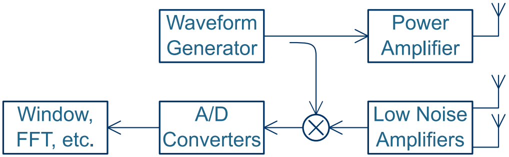
THE RADAR MASTERS COURSE

Project 1 – FMCW Front-end



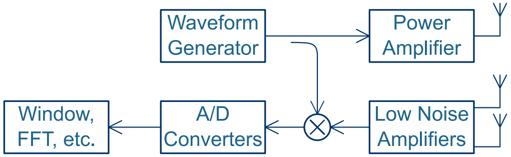
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

Project 1 – FMCW Front-end



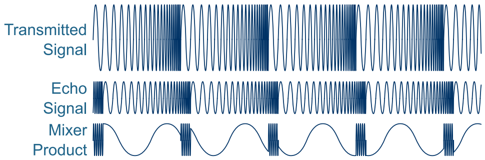
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

Project 1 – FMCW Front-end



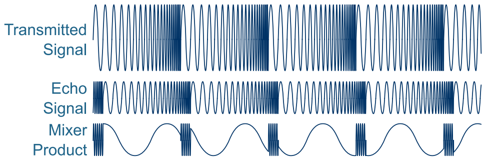
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps per burst (this is used for Doppler processing later in the chain)

Project 1 – FMCW Front-end



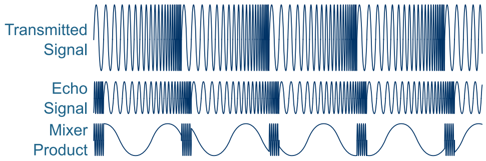
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 1 – FMCW Front-end



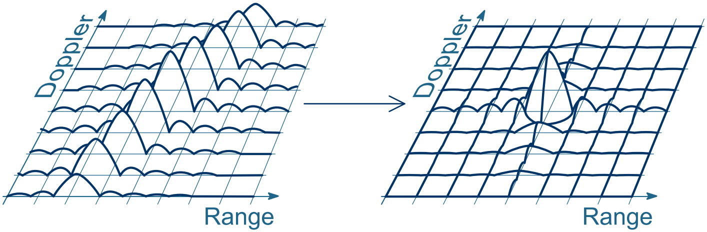
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 1 – FMCW Front-end



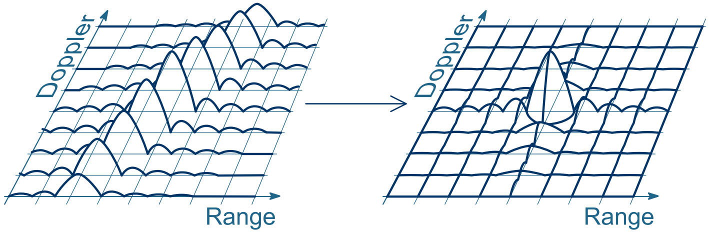
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 2 – Doppler FFTs



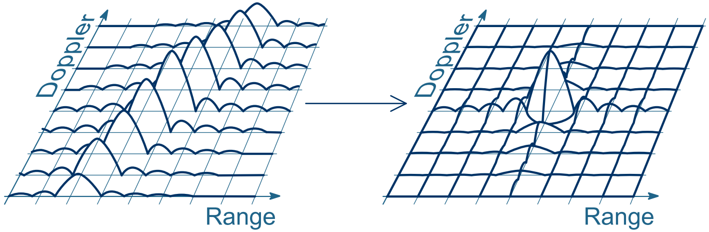
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 2 – Doppler FFTs



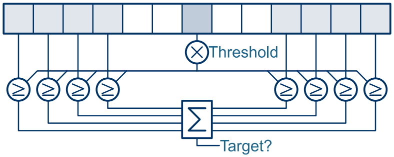
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 2 – Doppler FFTs



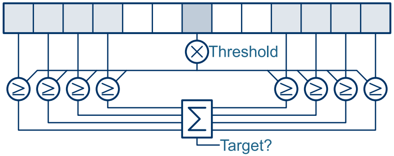
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 3 – Range CFAR



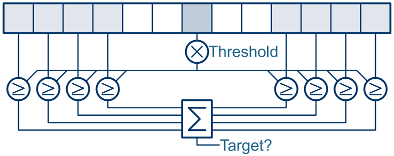
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly to the next step, but for this project, store the stream in SDRAM

Project 3 – Range CFAR



- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly to the next step, but for this project, store the stream in SDRAM

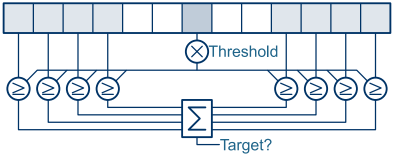
Project 3 – Range CFAR



- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly to the next step, but for this project, store the stream in SDRAM

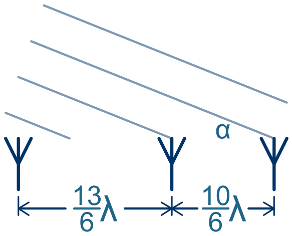
THE RADAR MASTERS COURSE

Project 3 – Range CFAR



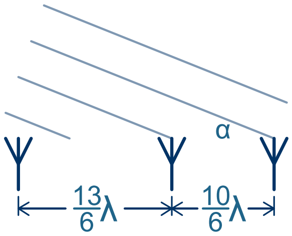
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly to the next step, but for this project, store the stream in SDRAM

Project 4 – Angle Extraction



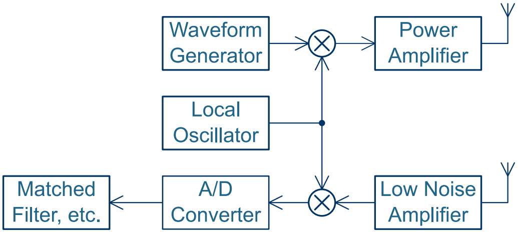
- Simulate the data output of Project 3 and inject the anticipated result into the SDRAM
- Play back the target stream and perform angle extraction for the sparse array

Project 4 – Angle Extraction



- Simulate the data output of Project 3 and inject the anticipated result into the SDRAM
- Play back the target stream and perform angle extraction for the sparse array

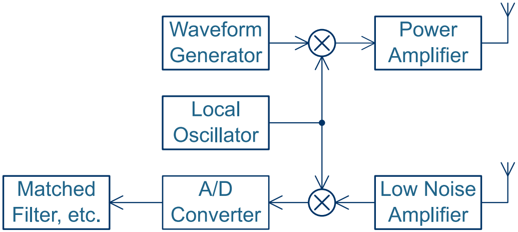
Project 5 – Pulsed Front-end



- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope

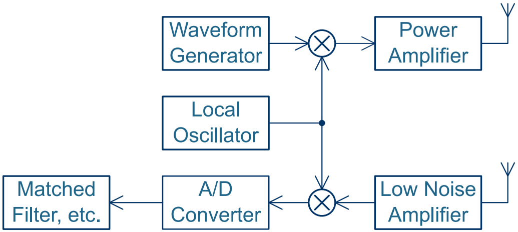
THE RADAR MASTERS COURSE

Project 5 – Pulsed Front-end



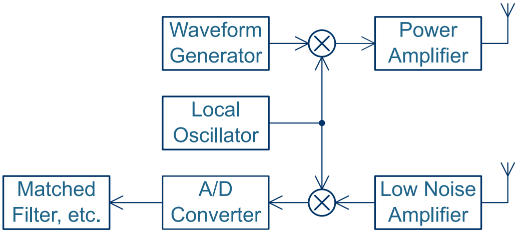
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope

Project 5 – Pulsed Front-end



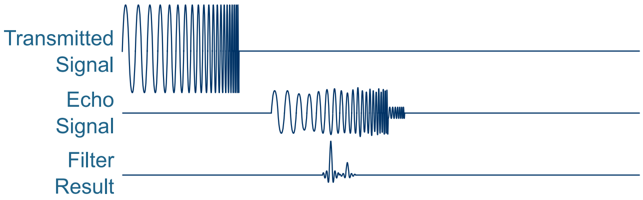
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope (Range is about 6.7 μs/km, so it is practical to output the signal from a long-range RADAR (350 km or so) on 40 kHz bandwidth PWM (filter time-constant of 4 μs)...

Project 5 – Pulsed Front-end



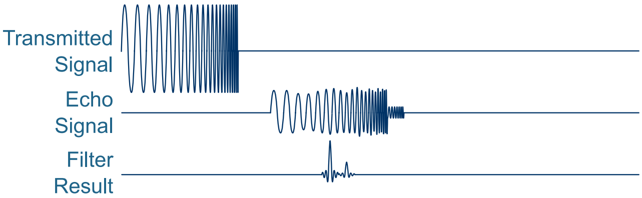
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI’s on the oscilloscope
- The rest of a typical processing chain is conceptually similar to projects 2 to 4, which can be adapted to process the results from this front-end

Project 5 – Pulsed Front-end



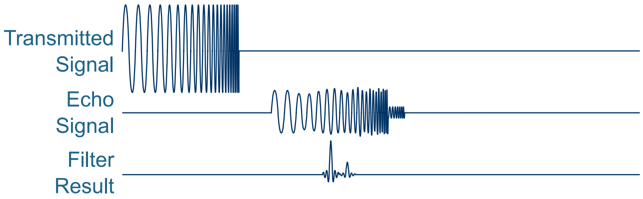
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 5 – Pulsed Front-end



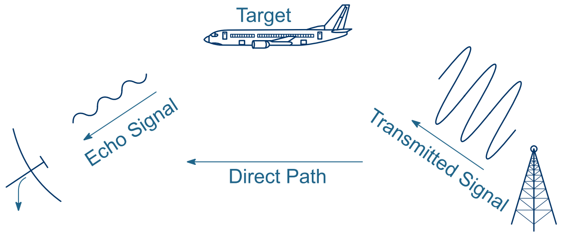
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 5 – Pulsed Front-end



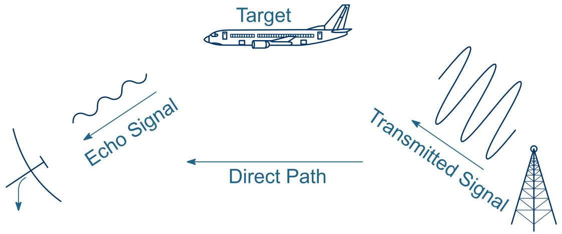
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially take this output data and processes it further

Project 6 – Commensal RADAR



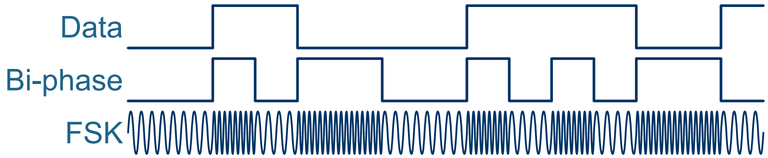
- Design and implement a commensal RADAR front-end
- Inject raw ADC data and implement range extraction (correlate the direct path with the echo signal)

Project 6 – Commensal RADAR



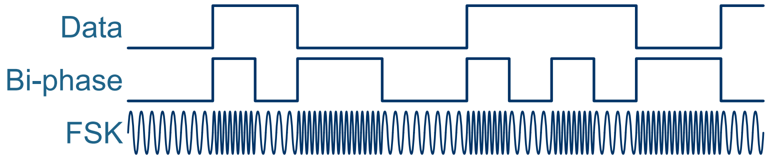
- Design and implement a commensal RADAR front-end
- Inject raw ADC data and implement range extraction (correlate the direct path with the echo signal)

Project 7 – FSK Communication



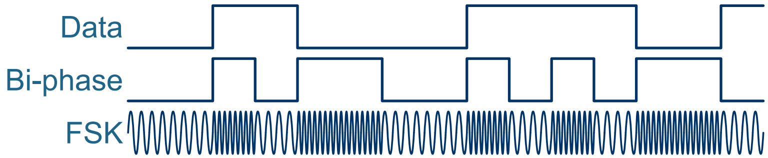
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel performance in the presence of noise

Project 7 – FSK Communication



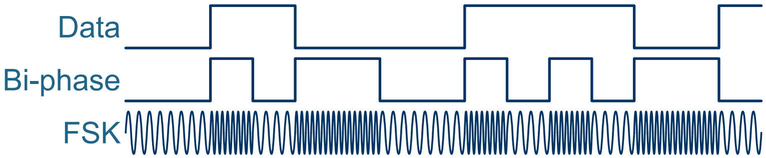
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel performance in the presence of noise

Project 7 – FSK Communication



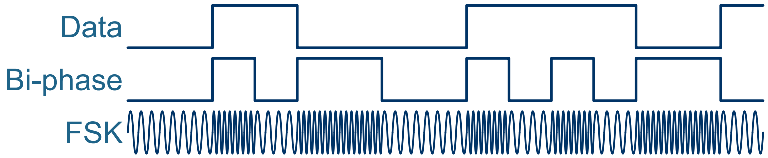
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel performance in the presence of noise

Project 7 – FSK Communication



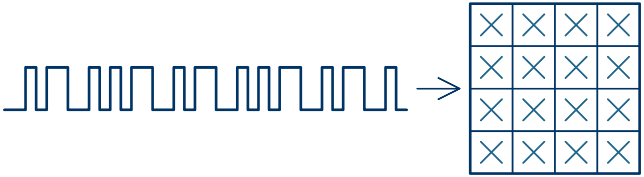
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel performance in the presence of noise

Project 7 – FSK Communication



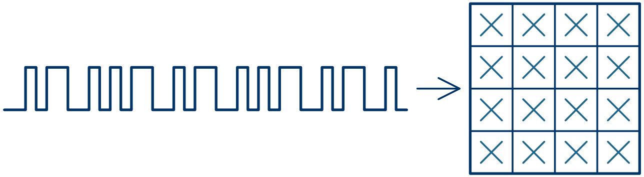
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel performance in the presence of noise

Project 8 – QAM Modulator



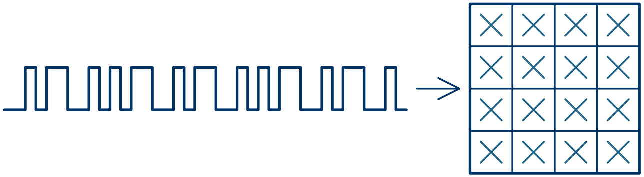
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



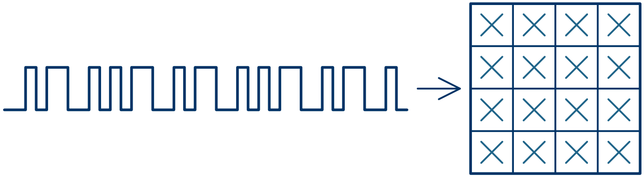
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



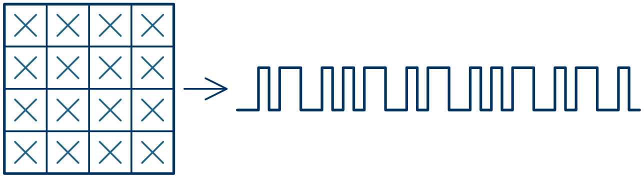
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



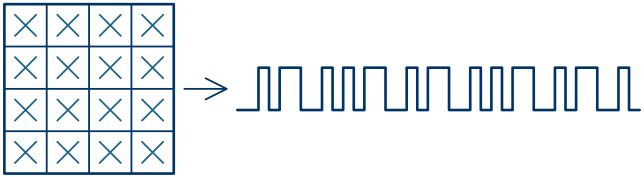
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 9 – QAM Demodulator



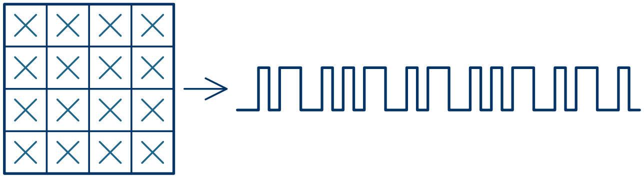
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end
(i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



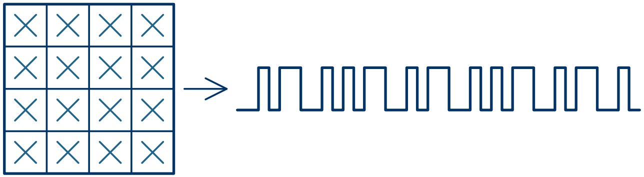
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end
  (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



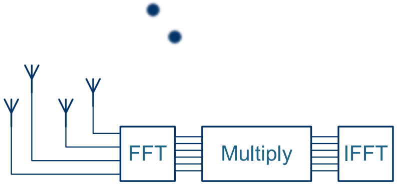
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end
  (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

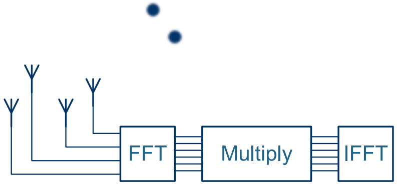
Project 10 – Astronomy Receiver



- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't fit, do them sequentially

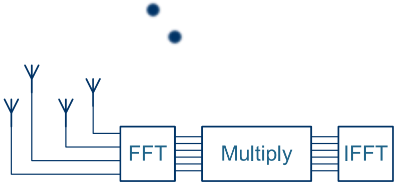
THE RADAR MASTERS COURSE

Project 10 – Astronomy Receiver



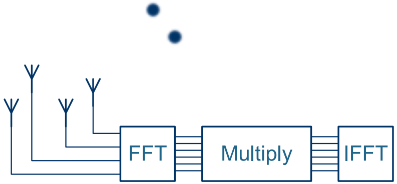
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't fit, do them sequentially

Project 10 – Astronomy Receiver



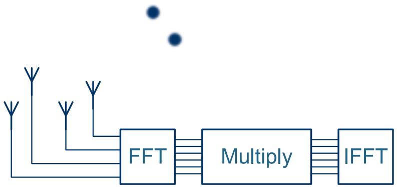
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't fit, do them sequentially

Project 10 – Astronomy Receiver



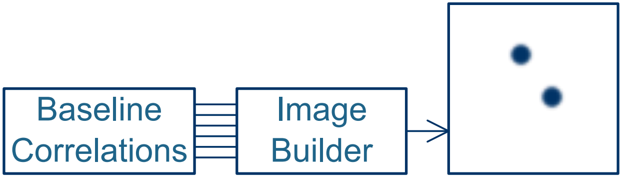
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't fit, do them sequentially

Project 10 – Astronomy Receiver



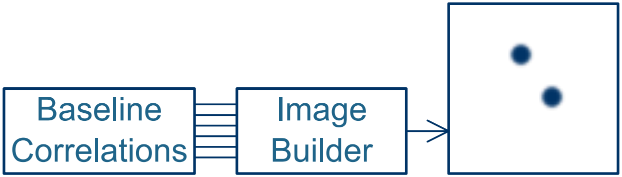
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't fit, do them sequentially

Project 11 – Astronomy Image



- Simulate the output of Project 10 and inject the anticipated result into the SDRAM
- Build an image from the correlation data
- Store the resulting image in SDRAM so that it can be viewed on the PC

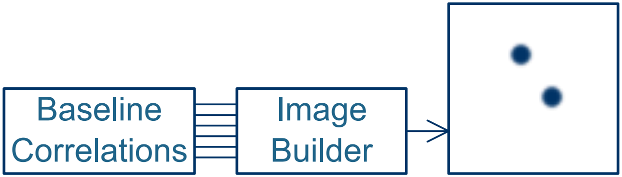
Project 11 – Astronomy Image



- Simulate the output of Project 10 and inject the anticipated result into the SDRAM
- Build an image from the correlation data
- Store the resulting image in SDRAM so that it can be viewed on the PC

THE RADAR MASTERS COURSE

Project 11 – Astronomy Image



- Simulate the output of Project 10 and inject the anticipated result into the SDRAM
- Build an image from the correlation data
- Store the resulting image in SDRAM so that it can be viewed on the PC

Outline

Design Once, Use Many

- Whenever possible, design your modules such that they can be re-used in other projects
- Use module parametrisation when appropriate
- Use standardised bus structures and interfaces, and the same interface family across all projects
- Use consistent naming conventions
- Clearly mark negative logic in the name

Design Once, Use Many

- Whenever possible, design your modules such that they can be re-used in other projects
- Use module parametrisation when appropriate
- Use standardised bus structures and interfaces, and the same interface family across all projects
- Use consistent naming conventions
- Clearly mark negative logic in the name

Design Once, Use Many

- Whenever possible, design your modules such that they can be re-used in other projects
- Use module parametrisation when appropriate
- Use standardised bus structures and interfaces, and the same interface family across all projects
- Use consistent naming conventions
- Clearly mark negative logic in the name

Design Once, Use Many

- Whenever possible, design your modules such that they can be re-used in other projects
- Use module parametrisation when appropriate
- Use standardised bus structures and interfaces, and the same interface family across all projects
- Use consistent naming conventions
- Clearly mark negative logic in the name

Design Once, Use Many

- Whenever possible, design your modules such that they can be re-used in other projects
- Use module parametrisation when appropriate
- Use standardised bus structures and interfaces, and the same interface family across all projects
- Use consistent naming conventions
- Clearly mark negative logic in the name

Libraries

- When possible, use the vendor-provided libraries
- Be careful: if the library function does not do quite what you want, it's generally easier and faster to design your own than to hack the existing library to do what you want
- Always use the correct tool for the job

Libraries

- When possible, use the vendor-provided libraries
- Be careful: if the library function does not do quite what you want, it’s generally easier and faster to design your own than to hack the existing library to do what you want
- Always use the correct tool for the job

Libraries

- When possible, use the vendor-provided libraries
- Be careful: if the library function does not do quite what you want, it's generally easier and faster to design your own than to hack the existing library to do what you want
- Always use the correct tool for the job

Testing and Debugging

- Always test as small a design as possible: in the ideal case, unit-test one module at a time
- In some cases, it’s faster to simulate
- Other times, its easier and faster to compile and test on real hardware than to write the test-bench
- The on-chip logic analyser (i.e. Diamond Reveal, Quartus Signal-tap, Vivado Chip-scope, etc.) is your friend!

Testing and Debugging

- Always test as small a design as possible: in the ideal case, unit-test one module at a time
- In some cases, it's faster to simulate
- Other times, its easier and faster to compile and test on real hardware than to write the test-bench
- The on-chip logic analyser (i.e. Diamond Reveal, Quartus Signal-tap, Vivado Chip-scope, etc.) is your friend!

Testing and Debugging

- Always test as small a design as possible: in the ideal case, unit-test one module at a time
- In some cases, it's faster to simulate
- Other times, its easier and faster to compile and test on real hardware than to write the test-bench
- The on-chip logic analyser (i.e. Diamond Reveal, Quartus Signal-tap, Vivado Chip-scope, etc.) is your friend!

Testing and Debugging

- Always test as small a design as possible: in the ideal case, unit-test one module at a time
- In some cases, it's faster to simulate
- Other times, its easier and faster to compile and test on real hardware than to write the test-bench
- The on-chip logic analyser (i.e. Diamond Reveal, Quartus Signal-tap, Vivado Chip-scope, etc.) is your friend!

Scripting

- Most FPGA IDEs, including Lattice Diamond, Xilinx (AMD) Vivado and Altera (Intel) Quartus have powerful TCL scripting support
- You can automate version control, a more complicated compile chain (e.g. compiling a software component before the FPGA hardware), etc.
- Some IDEs lets you call a script automatically when starting and / or finishing a compile.

Scripting

- Most FPGA IDEs, including Lattice Diamond, Xilinx (AMD) Vivado and Altera (Intel) Quartus have powerful TCL scripting support
- You can automate version control, a more complicated compile chain (e.g. compiling a software component before the FPGA hardware), etc.
- Some IDEs lets you call a script automatically when starting and / or finishing a compile.

Scripting

- Most FPGA IDEs, including Lattice Diamond, Xilinx (AMD) Vivado and Altera (Intel) Quartus have powerful TCL scripting support
- You can automate version control, a more complicated compile chain (e.g. compiling a software component before the FPGA hardware), etc.
- Some IDEs lets you call a script automatically when starting and / or finishing a compile.

Outline

Advanced Timing Constraints

FIR Filters

Practical – FIR Filter

Projects

Tips and Tricks

Conclusion

Course Conclusion

- We have covered a huge amount of work in a very short time
- It’s up to you to go home and go play with the board
- And if you have questions: ask

Course Conclusion

- We have covered a huge amount of work in a very short time
- It’s up to you to go home and go play with the board
- And if you have questions: ask

Course Conclusion

- We have covered a huge amount of work in a very short time
- It’s up to you to go home and go play with the board
- And if you have questions: ask

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 6 – 4 May 2022