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 Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

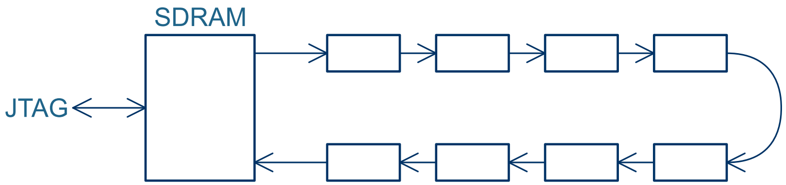
Day 5 – 21 July 2017

Outline

THE RADAR MASTERS COURSE

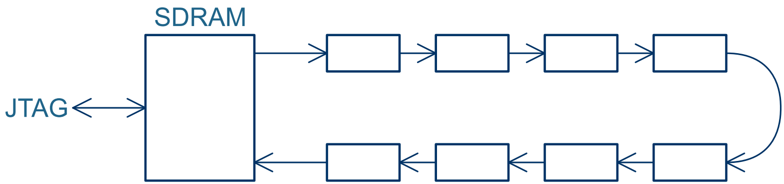
Outline

Sharing Resources



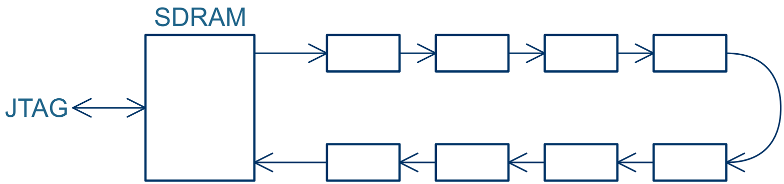
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^2C bus
  - etc.

Sharing Resources



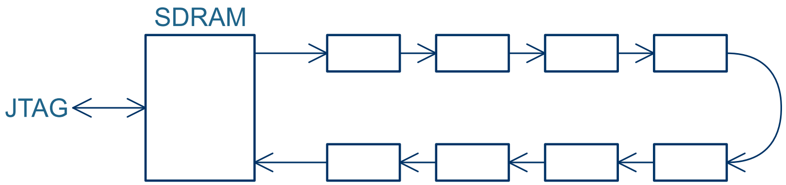
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^{2}C bus
  - etc.

Sharing Resources



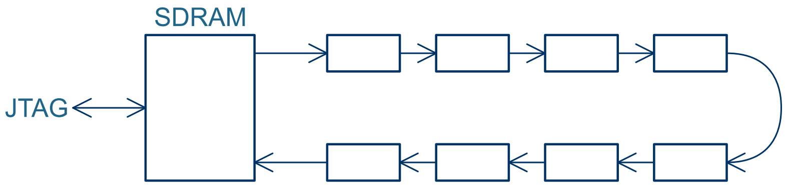
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^{2}C bus
  - etc.

Sharing Resources



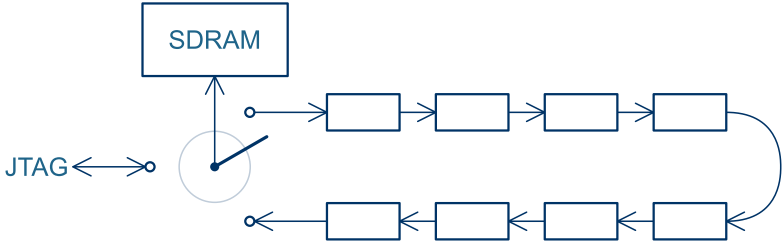
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^{2}C bus
  - etc.

Sharing Resources



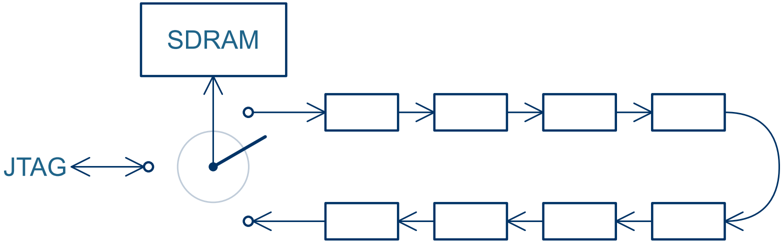
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^2C bus
  - etc.

Mutual Exclusion



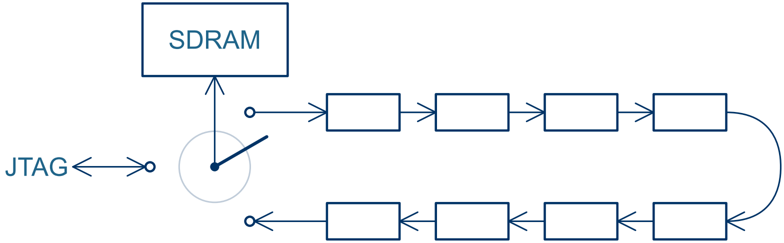
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



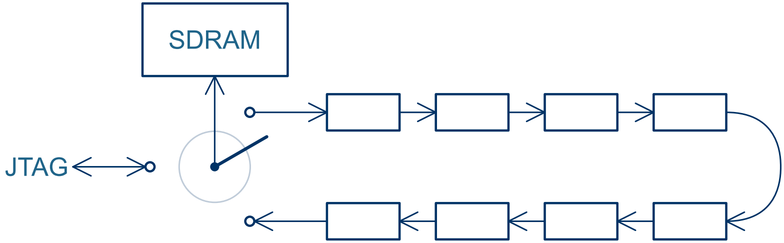
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



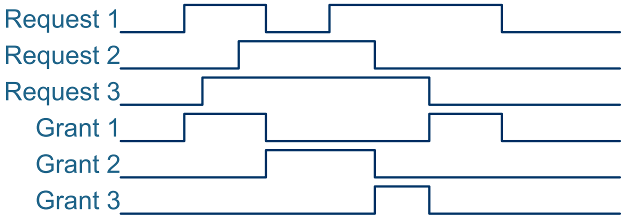
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



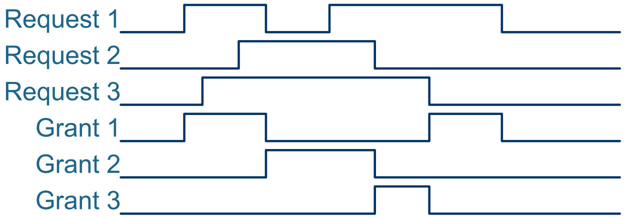
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Round-robin Mutual Exclusion



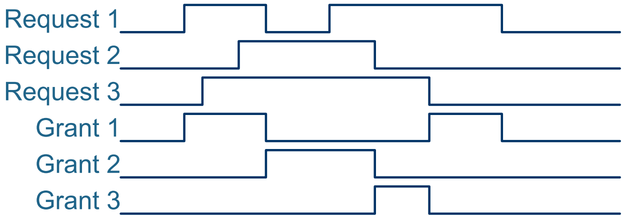
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module’s request line)

Round-robin Mutual Exclusion



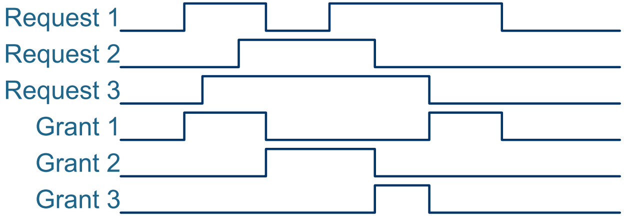
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module's request line)

Round-robin Mutual Exclusion



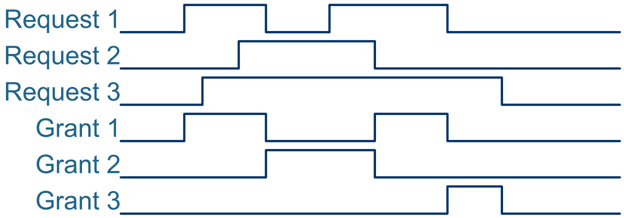
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module's request line)

Round-robin Mutual Exclusion



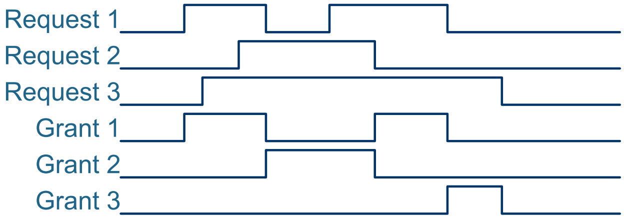
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module’s request line)

Priority-based Mutual Exclusion



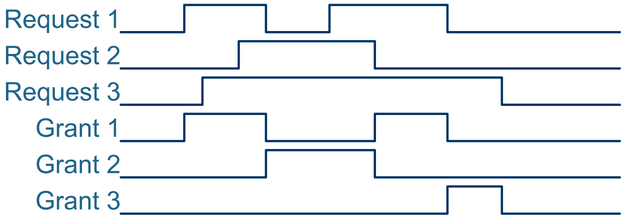
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion



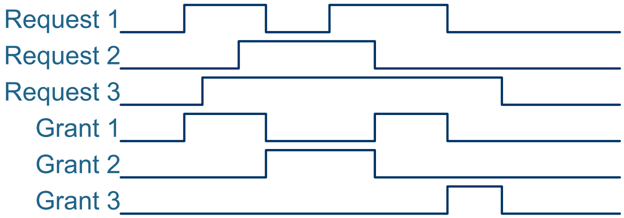
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion



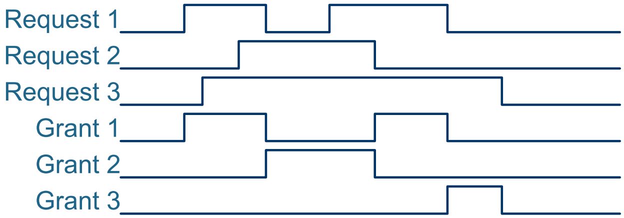
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion



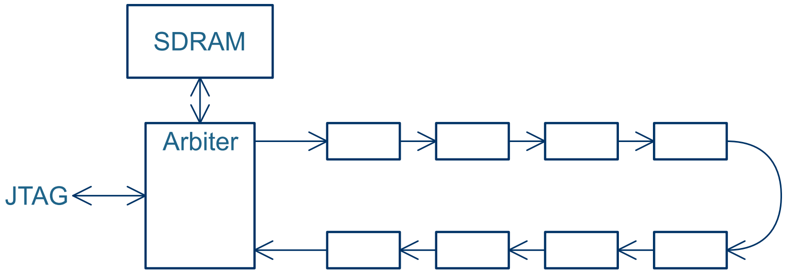
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion

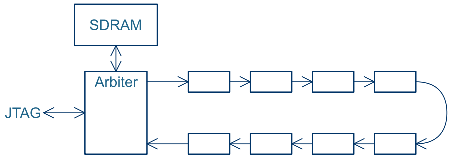


- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Arbitration

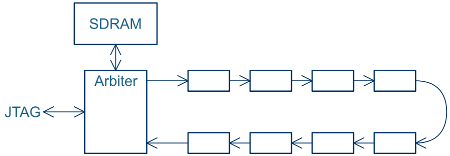


Arbitration



- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^{2}C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

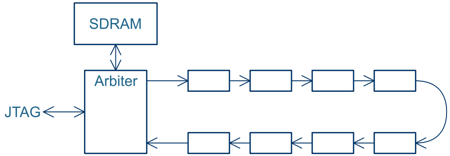
Arbitration



- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^2C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

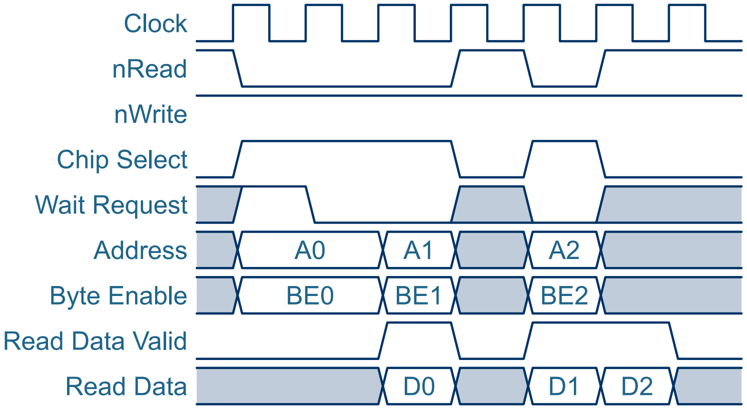
THE RADAR MASTERS COURSE

Arbitration



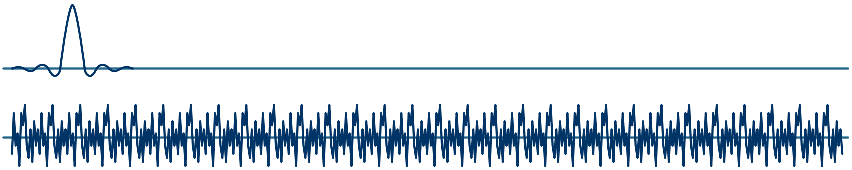
- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^2C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

Avalon Bus Arbitration



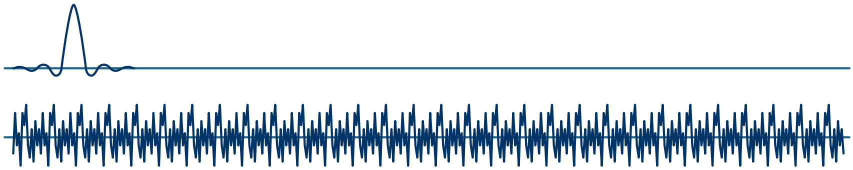
Outline

FIR Filter – Concept



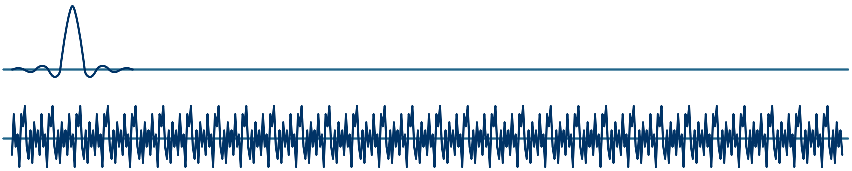
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



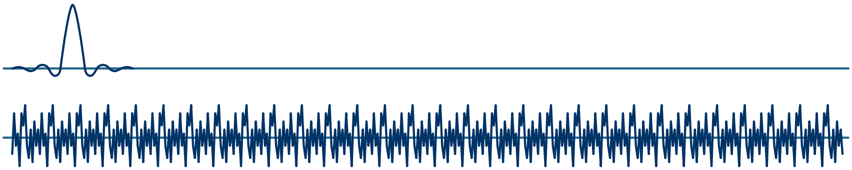
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



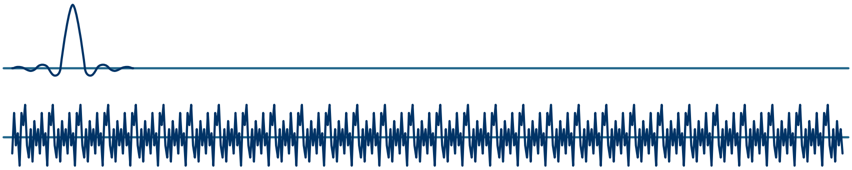
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



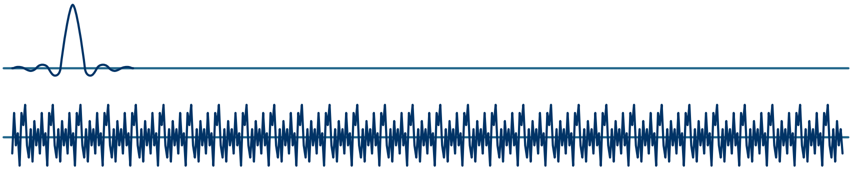
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



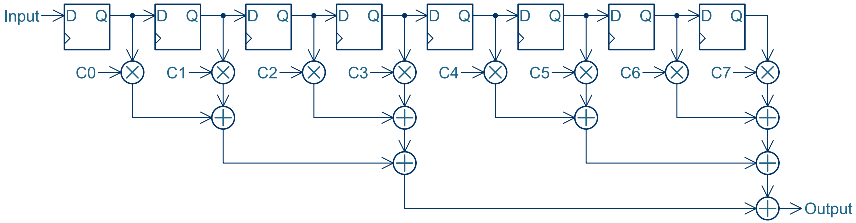
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Concept



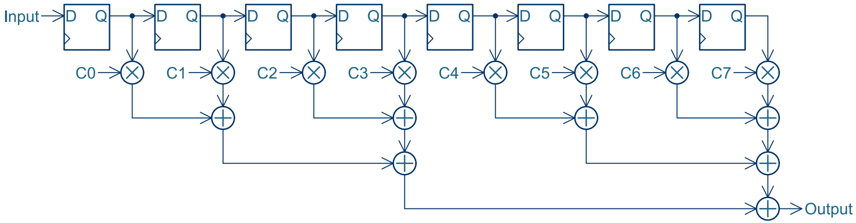
- Convolve the impulse-response with the signal:
  1. Time-reverse the impulse-response
  2. Within the FIR filter window, multiply the impulse-response sample by the signal sample
  3. Sum the products and output the result
  4. Move the impulse-response by one sample
  5. Repeat from step 2

FIR Filter – Implementation



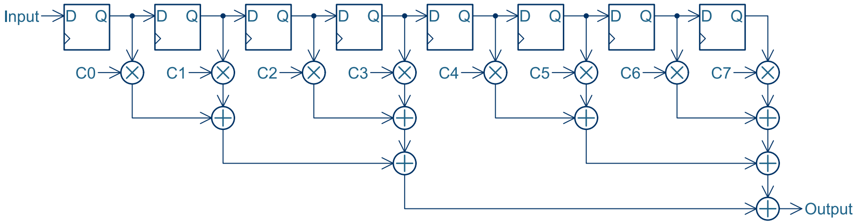
- Move the signal instead of the impulse-response
- To check the direction, inject an impulse (which should produce the impulse-response at the output)
- Pipeline the adder tree to increase the maximum clock frequency

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...

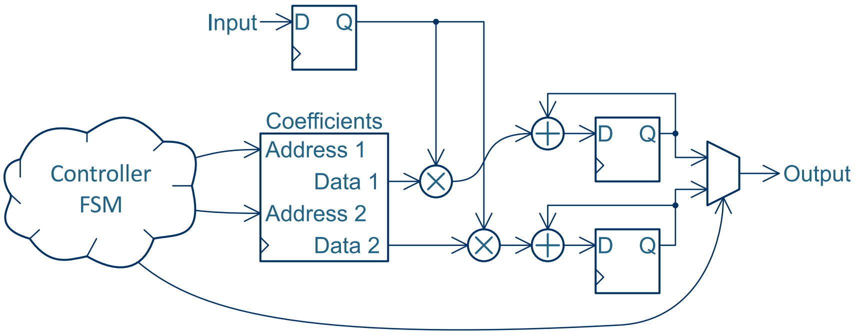
Architecture Design

- 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

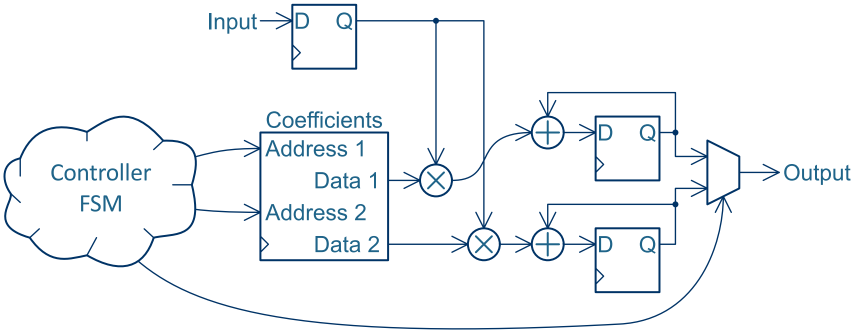
- Always take the design criteria into account when designing an architecture
- What happens when you add decimation (i.e. you don't need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the scenario: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

Decimation



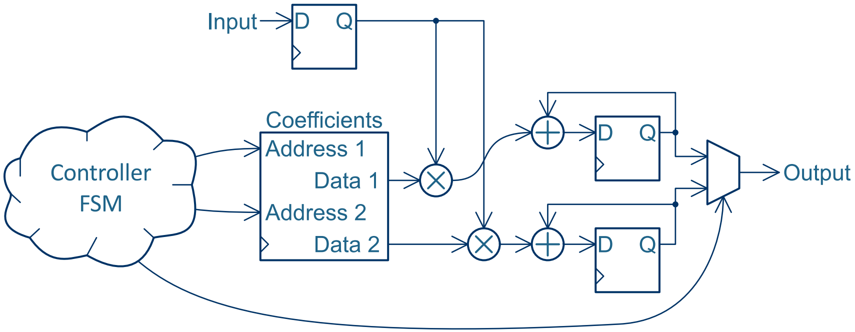
- This example decimates by $N/2$: a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

Decimation



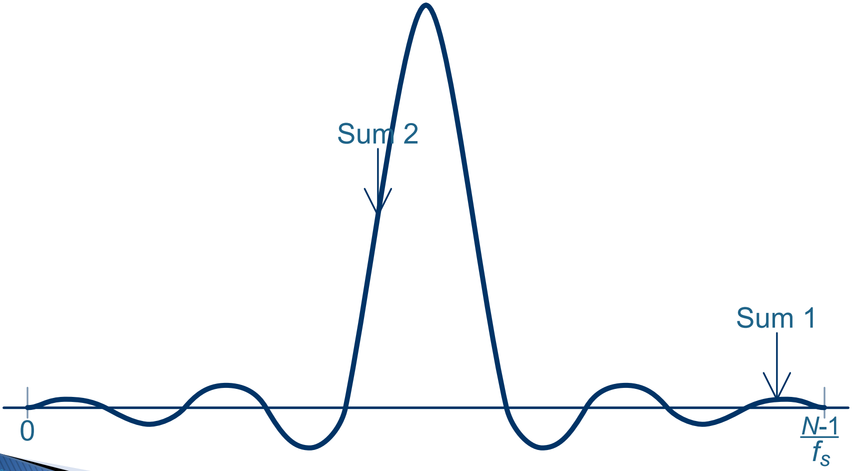
- This example decimates by $N/2$:
  a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

Decimation

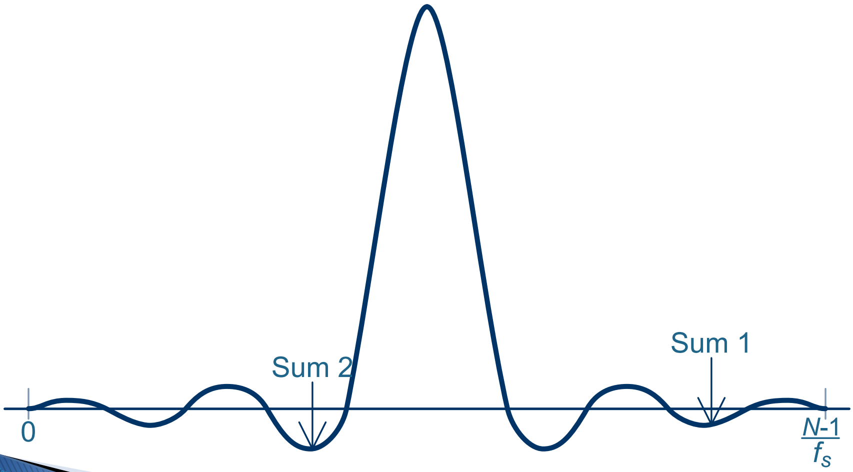


- This example decimates by $N/2$: a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

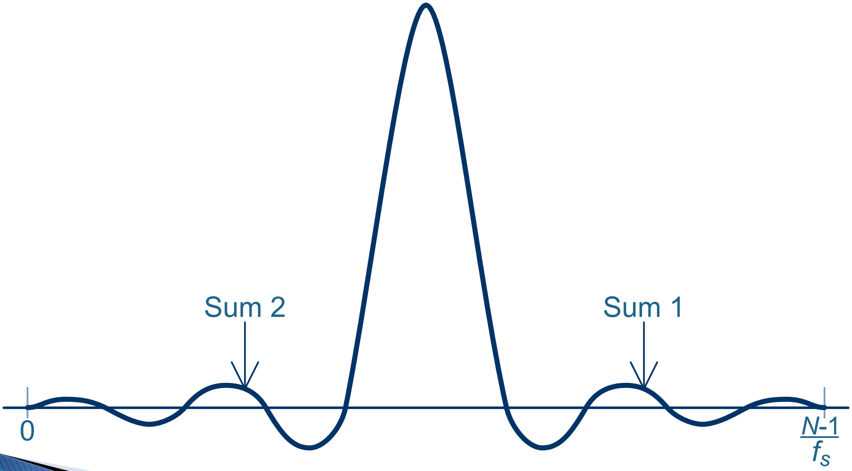
Decimation



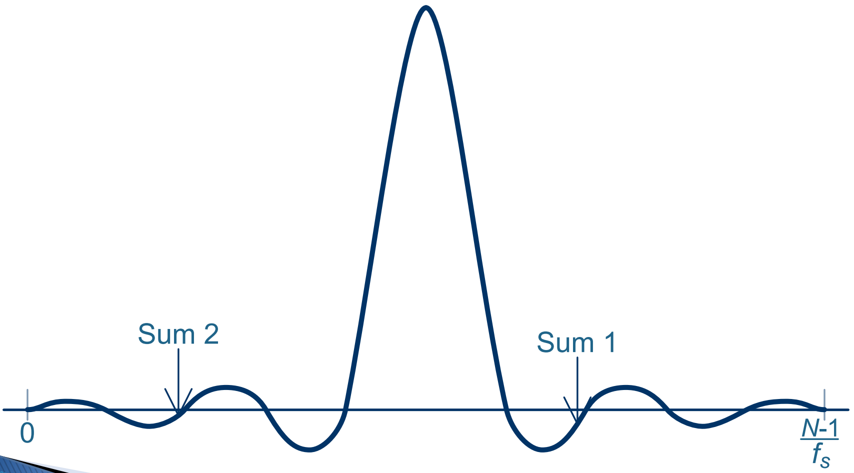
Decimation



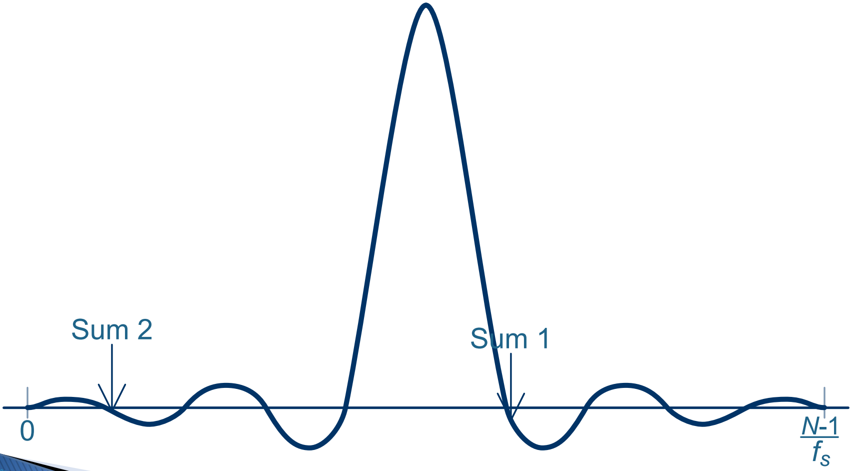
Decimation



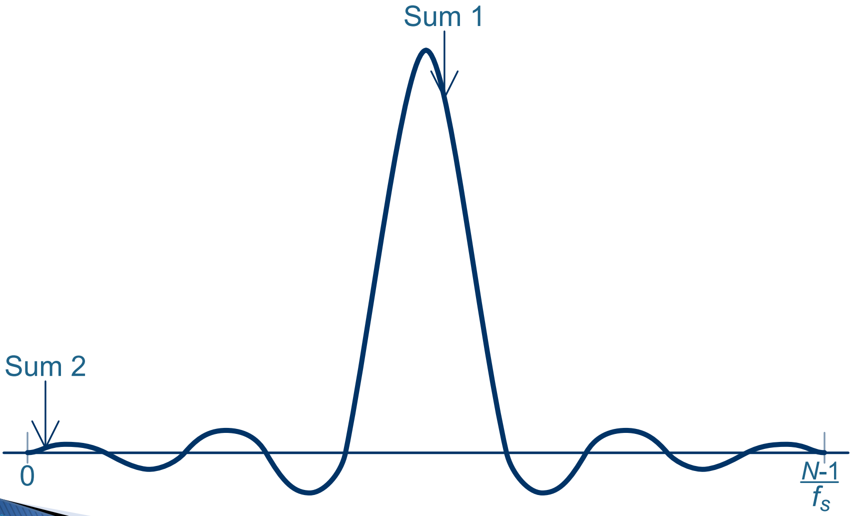
Decimation



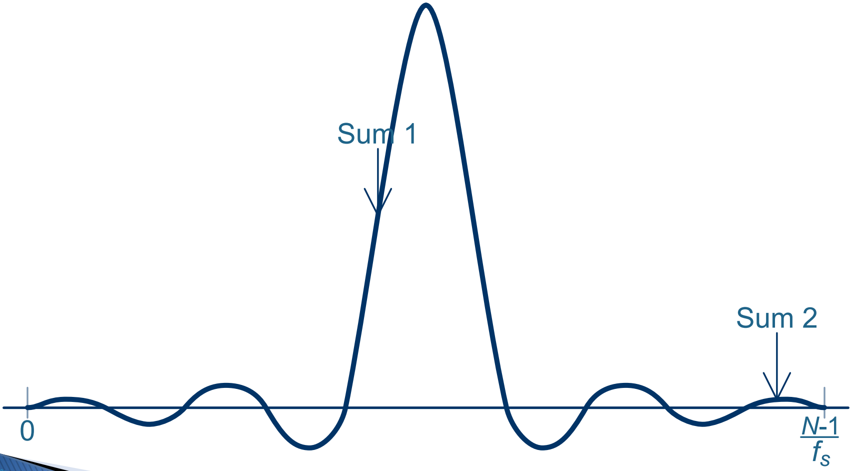
Decimation



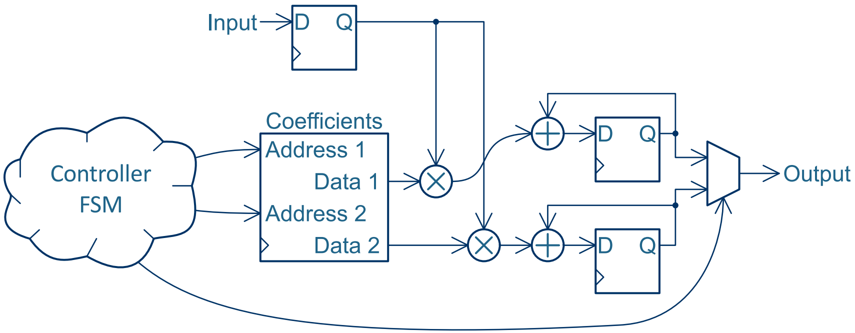
Decimation



Decimation

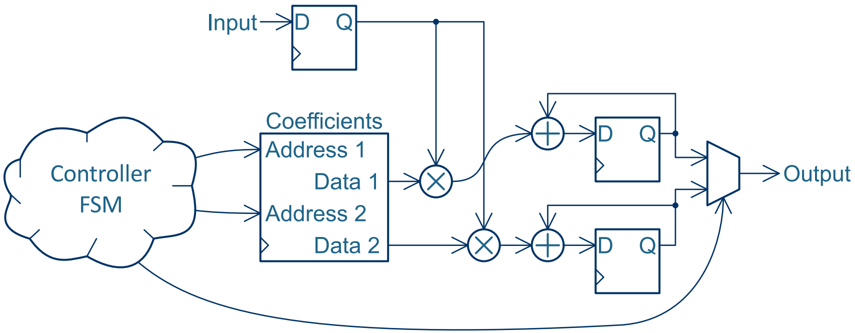


Faster System Clock



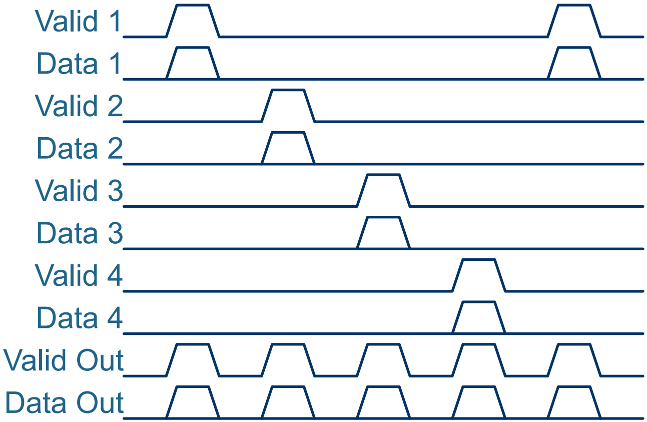
- If the sample clock is lower than the system clock, this same architecture can decimate by less:
- Keep more than 2 sums – sometimes more efficient to keep these in BRAM as well

Faster System Clock

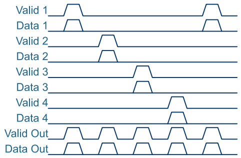


- If the sample clock is lower than the system clock, this same architecture can decimate by less:
- Keep more than 2 sums – sometimes more efficient to keep these in BRAM as well

Combining FIR Filter Units

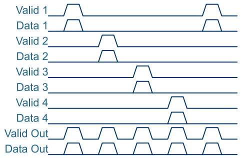


FIR Filter



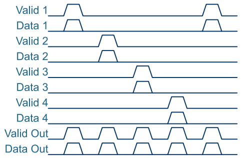
- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

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

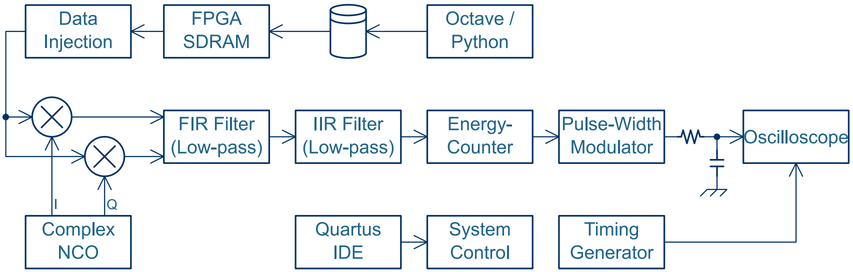


- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

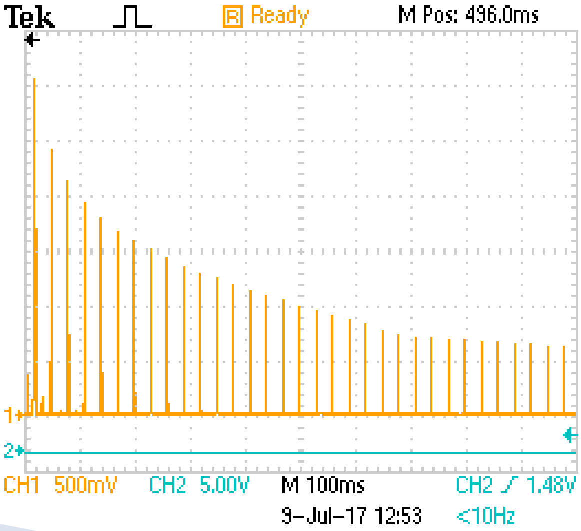
Coffee Break...

Outline

Practical


Data Injection
FPGA SDRAM
Octave / Python
FIR Filter (Low-pass)
IIR Filter (Low-pass)
Energy-Counter
Pulse-Width Modulator
Oscilloscope
I
Q
Complex NCO
Quartus IDE
System Control
Timing Generator

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

Agenda

- Design the FIR filter in Matlab and test using integer values
- Check for overflows, rounding problems, etc.
- Design what bit-widths to use, given the native RAM and DSP elements of the FPGA in question
- Implement and integrate the FIR filter into the design, and test the system as a whole
- Optional: change the JTAG vs. Injection selection from external switch to internal arbitration

Agenda

- Design the FIR filter in Matlab and test using integer values
- Check for overflows, rounding problems, etc.
- Design what bit-widths to use, given the native RAM and DSP elements of the FPGA in question
- Implement and integrate the FIR filter into the design, and test the system as a whole
- Optional: change the JTAG vs. Injection selection from external switch to internal arbitration

Agenda

- Design the FIR filter in Matlab and test using integer values
- Check for overflows, rounding problems, etc.
- Design what bit-widths to use, given the native RAM and DSP elements of the FPGA in question
- Implement and integrate the FIR filter into the design, and test the system as a whole
- Optional: change the JTAG vs. Injection selection from external switch to internal arbitration

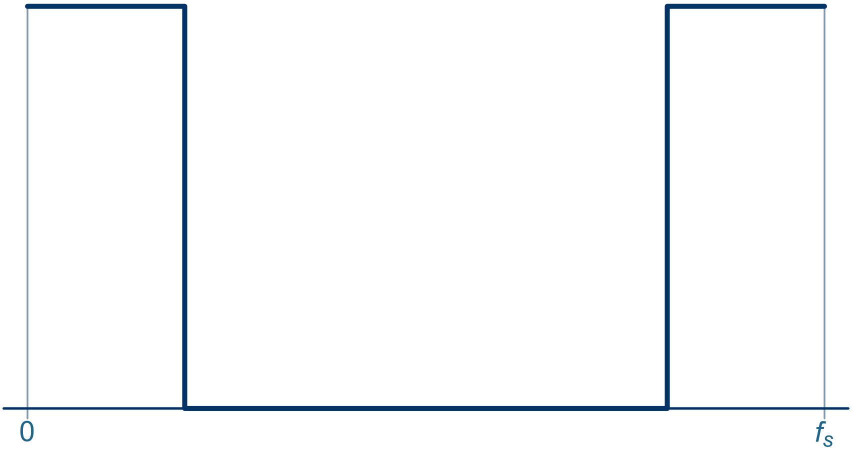
Agenda

- Design the FIR filter in Matlab and test using integer values
- Check for overflows, rounding problems, etc.
- Design what bit-widths to use, given the native RAM and DSP elements of the FPGA in question
- Implement and integrate the FIR filter into the design, and test the system as a whole
- Optional: change the JTAG vs. Injection selection from external switch to internal arbitration

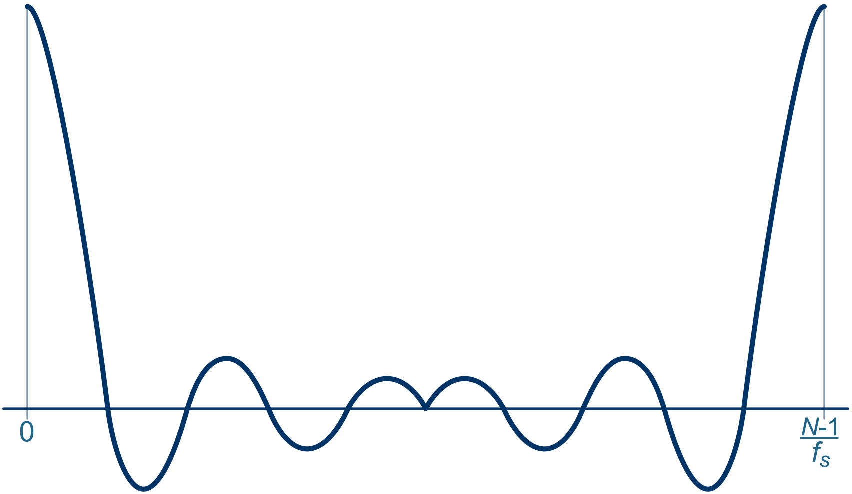
Agenda

- Design the FIR filter in Matlab and test using integer values
- Check for overflows, rounding problems, etc.
- Design what bit-widths to use, given the native RAM and DSP elements of the FPGA in question
- Implement and integrate the FIR filter into the design, and test the system as a whole
- Optional: change the JTAG vs. Injection selection from external switch to internal arbitration

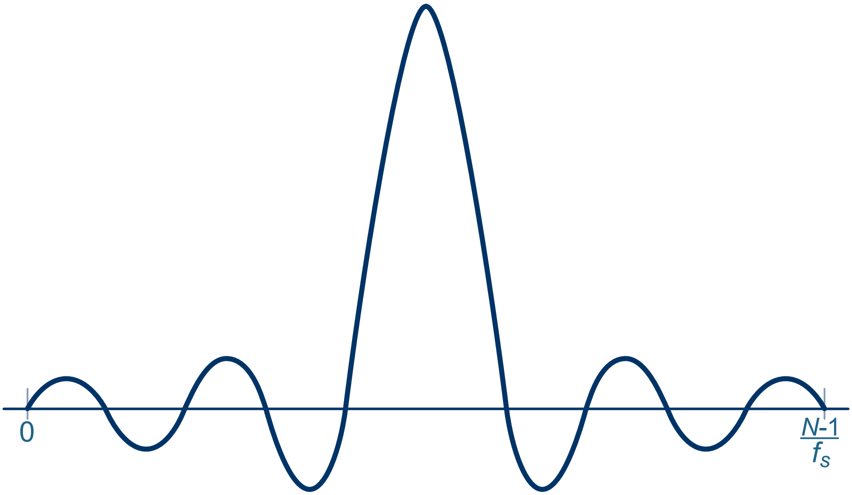
FIR Filter – Desired Reponse



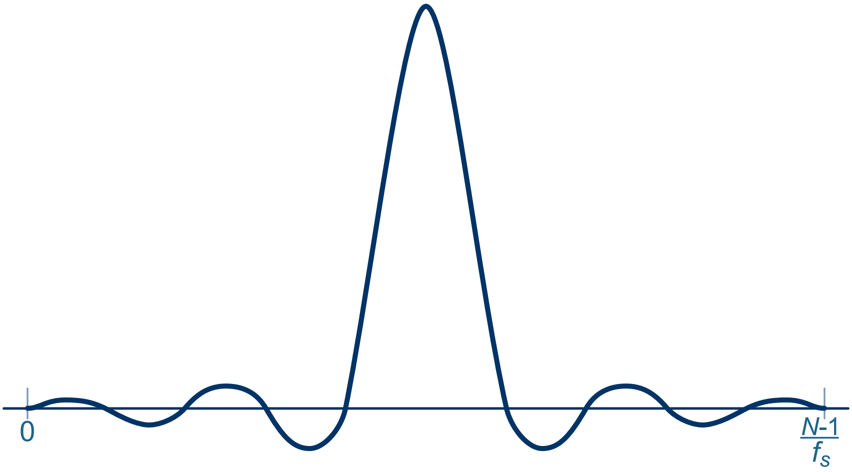
FIR Filter – IFFT (real part)



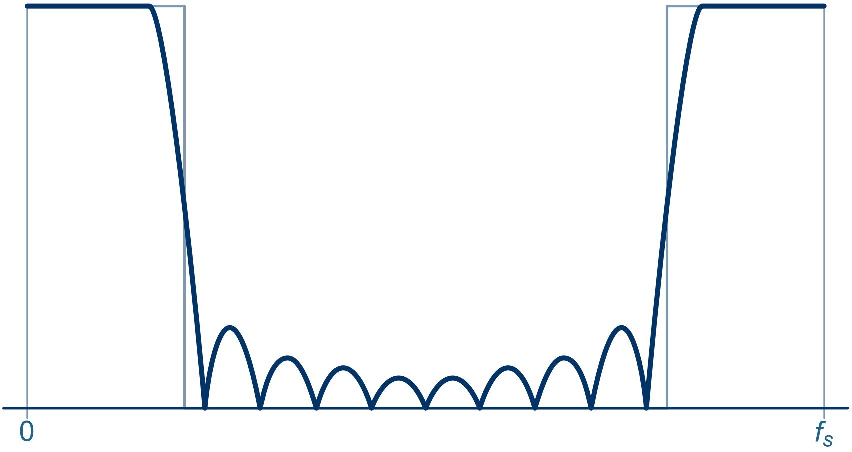
FIR Filter – Time-shifted



FIR Filter – Windowed



FIR Filter – Actual Reponse



(Zero-pad to see the side-lobes)

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

FIR Filter Implementation

- Implement a 1024-point FIR filter that decimates by 1024 (i.e. one sample output for every 1024 samples input)
- Use a cut-off frequency of 390 kHz 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 MIF file
- Verify through simulation
- Verify on FPGA
- Combine eight filter units to drop the decimation to 128
- Verify on FPGA

Outline

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 (sources and probes / virtual JTAG registers / GUI if you feel really ambitious / etc.)

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 (sources and probes / virtual JTAG registers / GUI if you feel really ambitious / etc.)

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 (sources and probes / virtual JTAG registers / GUI if you feel really ambitious / etc.)

THE RADAR MASTERS COURSE

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 (sources and probes / virtual JTAG registers / GUI if you feel really ambitious / etc.)

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 (sources and probes / virtual JTAG registers / GUI if you feel really ambitious / etc.)

The Demonstration

- The demonstrations are scheduled for the week of 4 September – 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 RADAR MASTERS COURSE

The Demonstration

- The demonstrations are scheduled for the week of 4 September – 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 4 September – 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 4 September – 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 4 September – 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 4 September – 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 relatively small FPGA
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for instance: always assume that targets are slow-moving, and that there are no multipath effects)
- 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 relatively small FPGA
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for instance: always assume that targets are slow-moving, and that there are no multipath effects)
- 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 relatively small FPGA
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for instance: always assume that targets are slow-moving, and that there are no multipath effects)
- 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 relatively small FPGA
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for instance: always assume that targets are slow-moving, and that there are no multipath effects)
- 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 relatively small FPGA
- Keep things simple – choose system parameters that favour easy implementation, not good system performance (for instance: always assume that targets are slow-moving, and that there are no multipath effects)
- 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

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

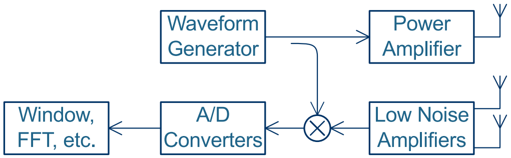
- I'm not on campus, but you can reach me via email
- Clearly explain what you're trying to do, and what you're struggling with
- Include your source code and, when applicable, some pictures of the architecture you're trying to implement and a test-bench to highlight the problem
- You'll find some good 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

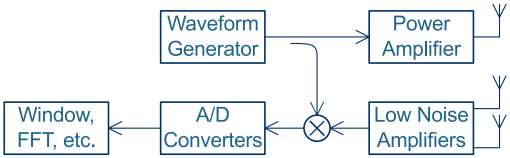
Project Ideas

Project 1 – FMCW Front-end



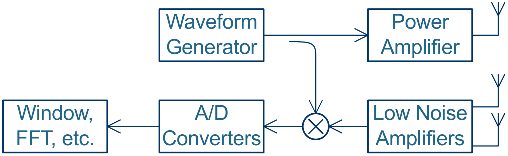
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



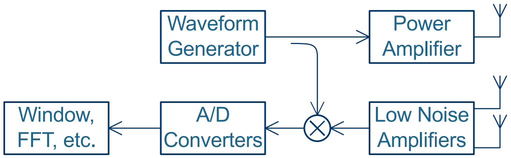
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



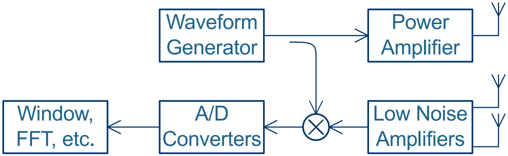
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



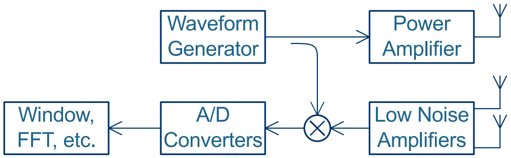
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



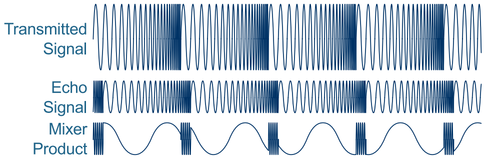
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



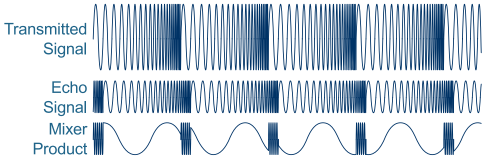
- Sparse-array FMCW RADAR (see Project 4)
- Choose system parameters appropriate for a practical radar – typical parameters include:
  - Sweep time of about 1 ms (sweep faster for fast-moving targets, and sweep slower for more range)
  - RF bandwidth of 500 MHz
  - 256 samples per sweep
  - 256 sweeps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

Project 1 – FMCW Front-end



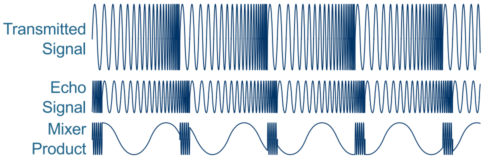
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially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

Project 1 – FMCW Front-end



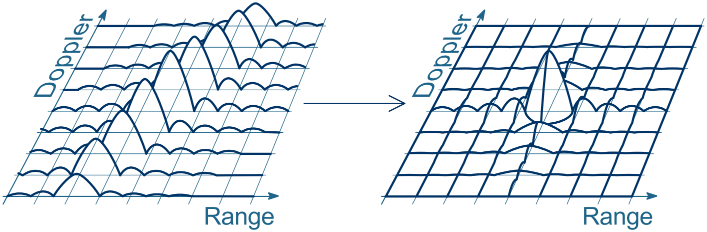
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially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

Project 1 – FMCW Front-end



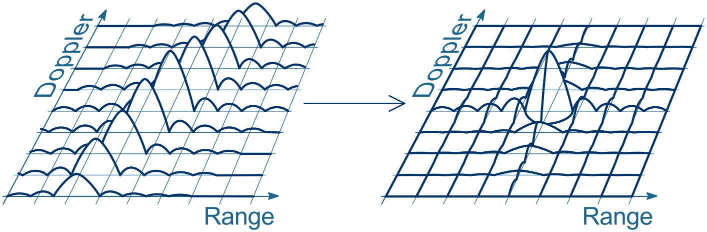
- You can assume that the FPGA generates the transmit sweep triangle, so you can use the SDRAM address to determine where in the sweep you are
- Take the FFT of each sweep (range FFT), organise them into bursts and store the results in SDRAM
- This is the end of this project – another project could potentially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

Project 2 – Doppler FFTs



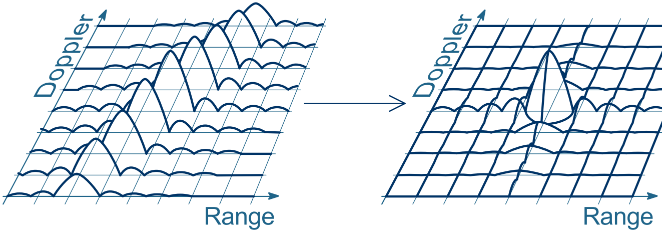
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 2 – Doppler FFTs



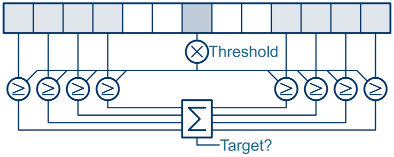
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 2 – Doppler FFTs



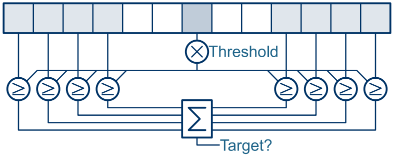
- Simulate the data output of Project 1 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SDRAM for the next step

Project 3 – Range CFAR



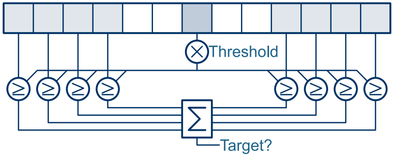
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly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

Project 3 – Range CFAR



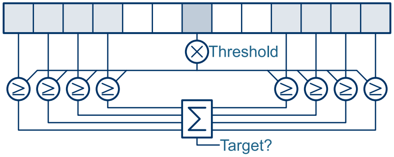
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly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

Project 3 – Range CFAR



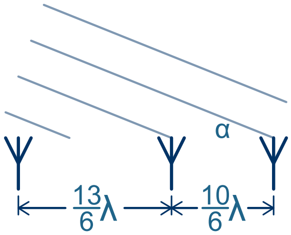
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly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

Project 3 – Range CFAR



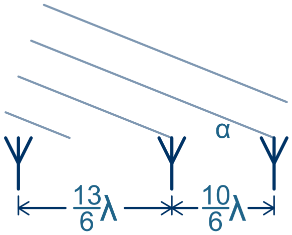
- Simulate the data output of Project 2 and inject the anticipated result into the SDRAM
- Play back the corner-turned data and process the range CFAR
- Create a stream of detected targets (store the range, Doppler and phase of the two incoming channels
- In a real system, this stream would go directly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

Project 4 – Angle Extraction



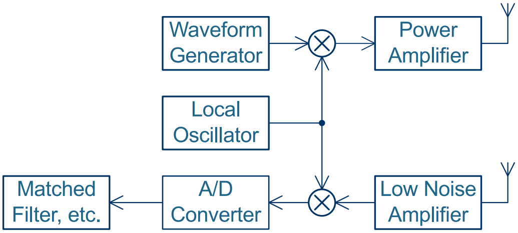
- Simulate the data output of Project 3 and inject the anticipated result into the SDRAM
- Play back the target stream and perform angle extraction for the sparse array

Project 4 – Angle Extraction



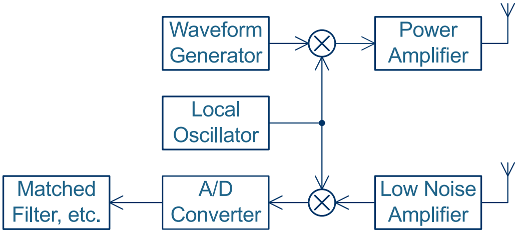
- Simulate the data output of Project 3 and inject the anticipated result into the SDRAM
- Play back the target stream and perform angle extraction for the sparse array

Project 5 – Pulsed Front-end



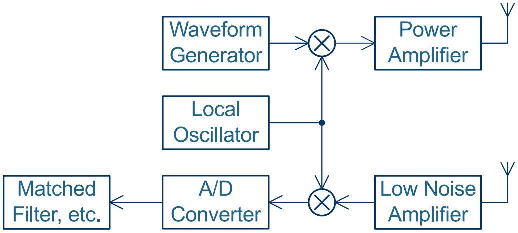
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope

Project 5 – Pulsed Front-end



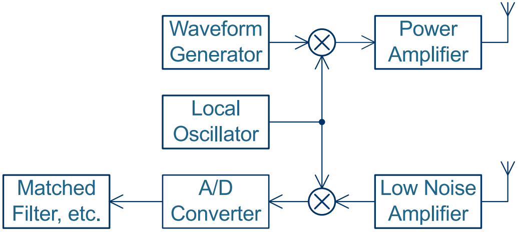
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope

Project 5 – Pulsed Front-end



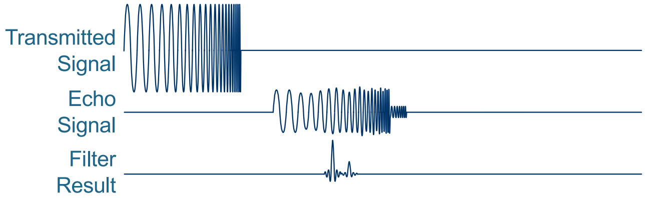
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope (Range is about 6.7 µs/km, so it is practical to output the signal from a long-range RADAR (350 km or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

Project 5 – Pulsed Front-end



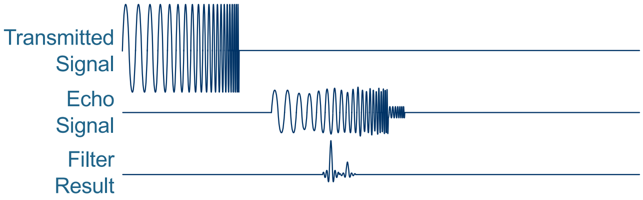
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI’s on the oscilloscope
- The rest of a typical processing chain is conceptually similar to projects 2 to 4, which can be adapted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

Project 5 – Pulsed Front-end



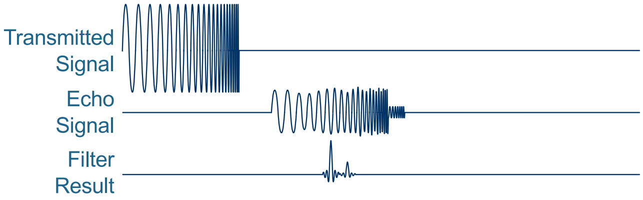
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially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

Project 5 – Pulsed Front-end



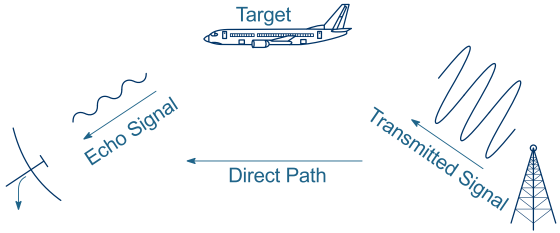
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially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

Project 5 – Pulsed Front-end



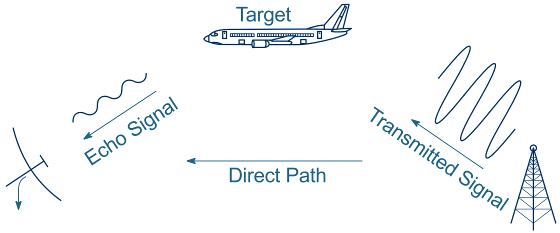
- You can assume that the FPGA generates the transmit timing control, so you can use the SDRAM address to determine where you are in the current PRI
- After doing matched filtering, organise the results into bursts and store them in SDRAM
- This is the end of this project – another project could potentially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

Project 6 – Commensal RADAR



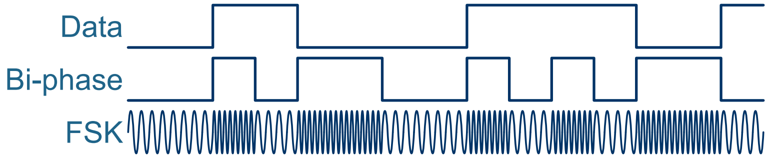
- Design and implement a commensal RADAR front-end
- Inject raw ADC data and implement range extraction (correlate the direct path with the echo signal)

Project 6 – Commensal RADAR



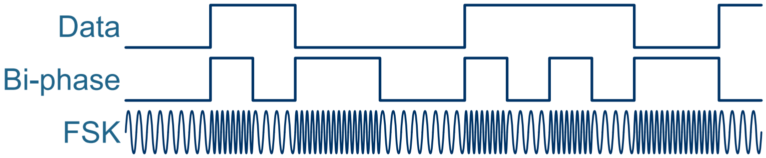
- Design and implement a commensal RADAR front-end
- Inject raw ADC data and implement range extraction (correlate the direct path with the echo signal)

Project 7 – FSK Communication



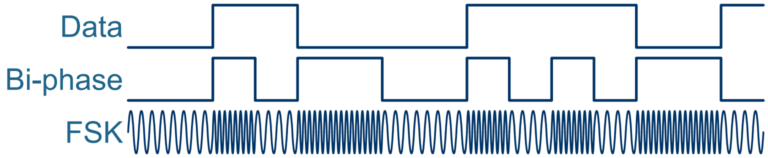
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 7 – FSK Communication



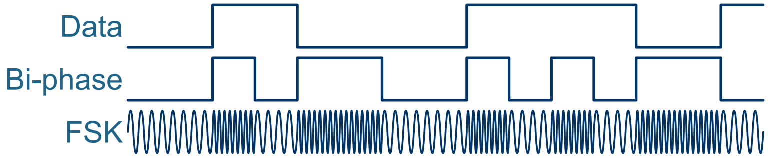
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 7 – FSK Communication



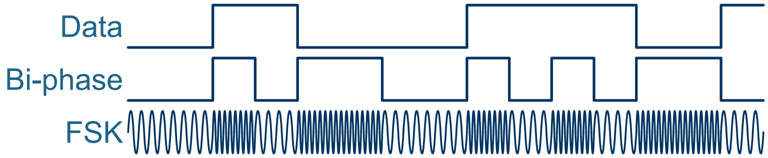
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 7 – FSK Communication



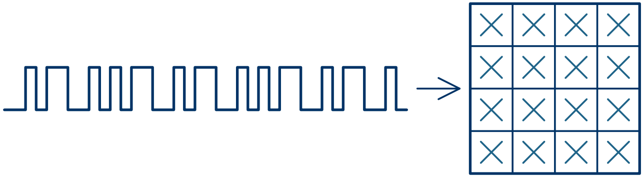
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 7 – FSK Communication



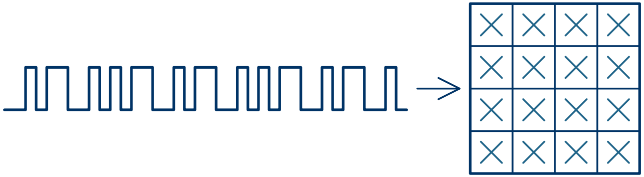
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 8 – QAM Modulator



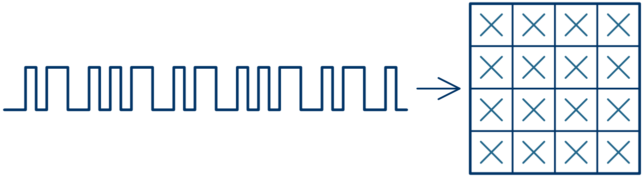
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



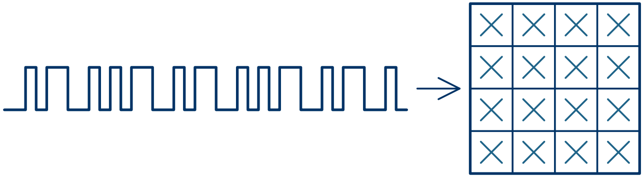
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



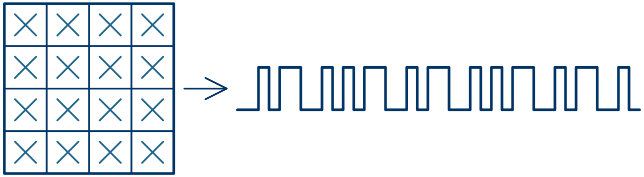
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 8 – QAM Modulator



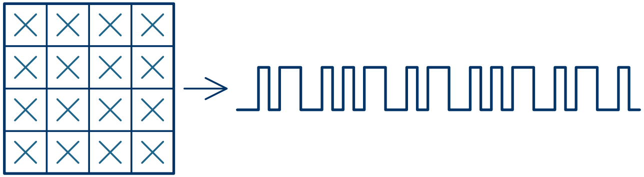
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SDRAM

Project 9 – QAM Demodulator



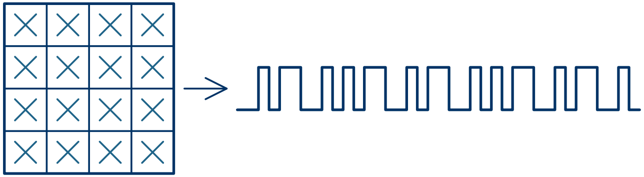
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



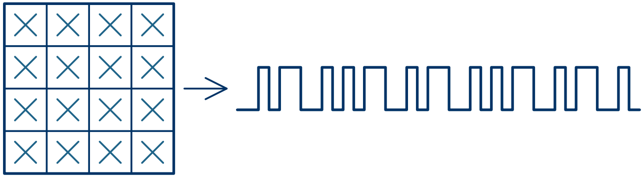
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end
  (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



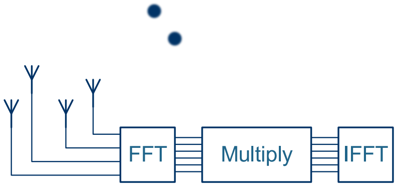
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end
  (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 9 – QAM Demodulator



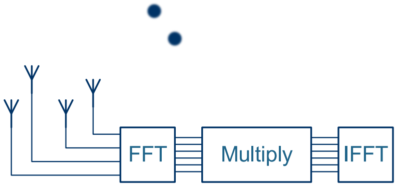
- Simulate the data output of Project 8 and inject the anticipated result into the SDRAM
- Assume an ideal RF front-end (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

Project 10 – Astronomy Receiver



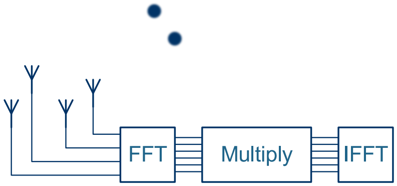
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't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

Project 10 – Astronomy Receiver



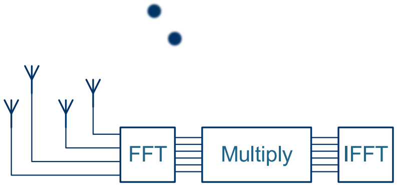
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't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

Project 10 – Astronomy Receiver



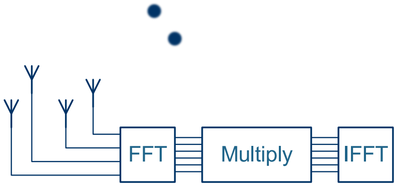
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't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

Project 10 – Astronomy Receiver



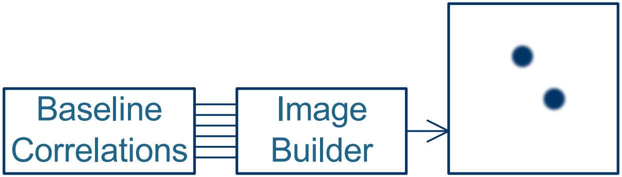
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't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

Project 10 – Astronomy Receiver



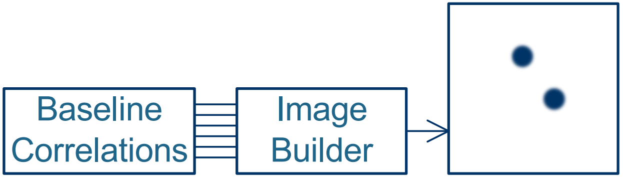
- Simulate a sky with at least two sources and inject ADC data from 4 receivers (4-bit per sample each)
- Perform correlation between all combinations of input channels (FFT, multiply, IFFT)
- Store the result in SDRAM
- Keep things simple: the earth is flat and stationary, etc.
- If parallel FFTs don't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

Project 11 – Astronomy Image



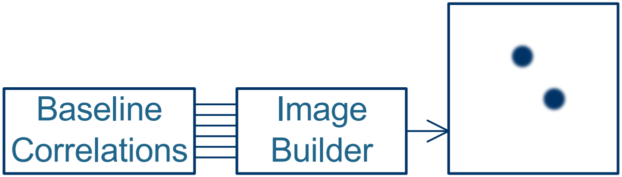
- Simulate the output of Project 10 and inject the anticipated result into the SDRAM
- Build an image from the correlation data
- Store the resulting image in SDRAM so that it can be viewed on the PC

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

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

Mutual Exclusion and Arbitration

Architecture Design

Practical

Projects

Tips and Tricks

Conclusion

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 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 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 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
- Signal-tap (Chip-scope in Xilinx) 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
- Signal-tap (Chip-scope in Xilinx) 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
- Signal-tap (Chip-scope in Xilinx) 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
- Signal-tap (Chip-scope in Xilinx) is your friend!

Scripting

- Both Xilinx and Altera have powerful TCL scripting support
- You can make scripts run automatically during the compilation process:

```
# In the QSF...

set_global_assignment        \
 -name PRE_FLOW_SCRIPT_FILE \
 "quartus_sh:Version Control/FirmwareVersion.tcl"

set_global_assignment         \
 -name POST_FLOW_SCRIPT_FILE \
 "quartus_sh:output_files/ProgramFPGA.tcl"
```

Scripting

- Both Xilinx and Altera have powerful TCL scripting support
- You can make scripts run automatically during the compilation process:

```
# In the QSF...

set_global_assignment       \
 -name PRE_FLOW_SCRIPT_FILE \
 "quartus_sh:Version Control/FirmwareVersion.tcl"

set_global_assignment        \
 -name POST_FLOW_SCRIPT_FILE \
 "quartus_sh:output_files/ProgramFPGA.tcl"
```

Outline

Mutual Exclusion and Arbitration

Architecture Design

Practical

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

THE RADAR MASTERS COURSE

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 Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

Day 5 – 21 July 2017