FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
http://www.rrsg.uct.ac.za

Presented by Dr John-Philip Taylor

Convened by Dr Stephen Paine

Day 5 – 13 September 2024

Outline

THE RADAR MASTERS COURSE

Outline

Typical Workflow

1. Design the IIR filter in Matlab / Octave / Python
   (pay careful attention to potential rounding errors and the resolution required)
2. Implement the IIR filter on the sample clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25\ \text{kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

THE RADAR MASTERS COURSE

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

THE RADAR MASTERS COURSE

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + b y_{n-1} - c y_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + b y_{n-1} - c y_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25\ \text{kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

THE RADAR MASTERS COURSE

IIR Filter

$$y_n = \frac{Ax_n + By_{n-1} - Cy_{n-2}}{2^N}, \quad \begin{cases} A = (2^N/a) \cdot \omega_0^2 \\ B = (2^N/a) \cdot b \\ C = (2^N/a) \cdot c \\ N = 32 \end{cases}$$

f_0	A	B	C
10	28	8 589 446 092	4 294 478 824
100	2 775	8 585 049 594	4 290 085 073
1 000	274 663	8 541 087 701	4 246 395 068
10 000	24 799 428	8 104 255 079	3 834 087 211
100 000	997 792 625	4 839 800 553	1 542 625 882

IIR Filter

- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Be careful with the signedness of the operation
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

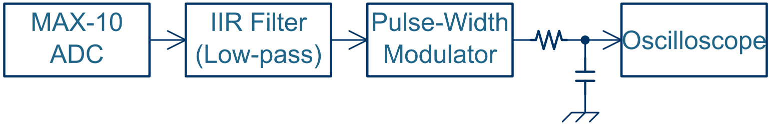
IIR Filter

- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Be careful with the signedness of the operation
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

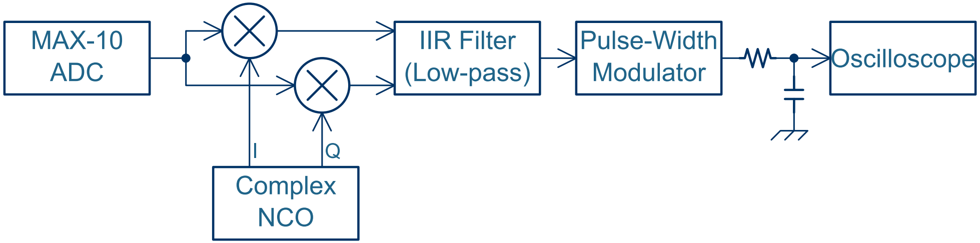
```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

THE RADAR MASTERS COURSE

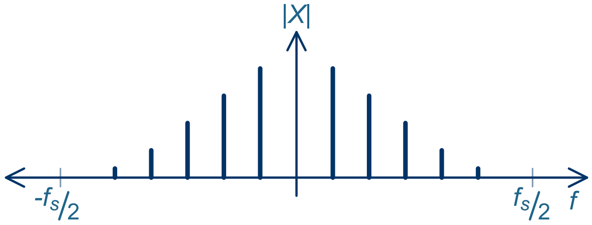
IIR Filter – ADC Test (Optional)



IIR Filter – ADC Test (Optional)



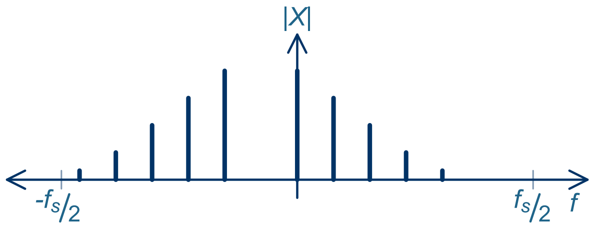
IIR Filter – Injection Test



$f_s = 44.1$ kHz
$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

IIR Filter – Injection Test

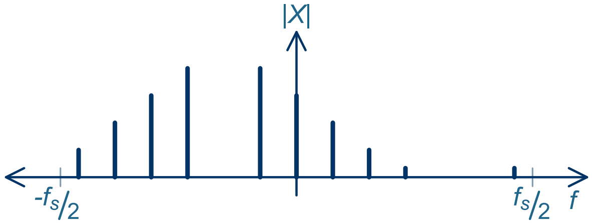


$f_s = 44.1$ kHz

$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

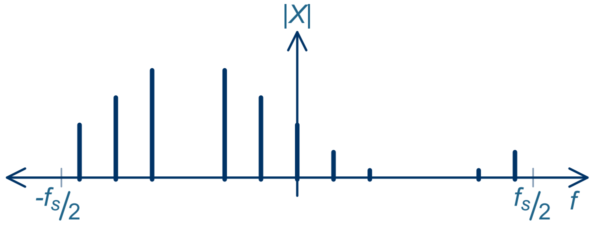
IIR Filter – Injection Test



$f_s = 44.1$ kHz
$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

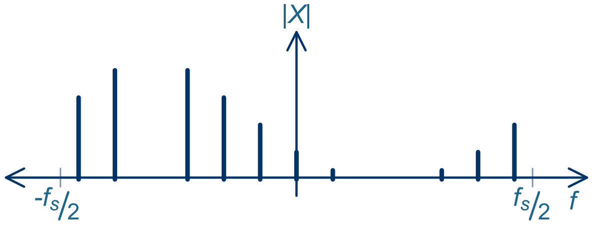
IIR Filter – Injection Test



$f_s = 44.1$ kHz
$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

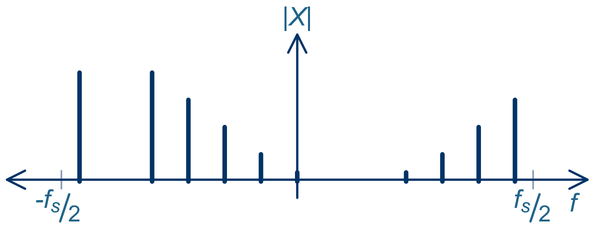
IIR Filter – Injection Test



$f_s = 44.1$ kHz
$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

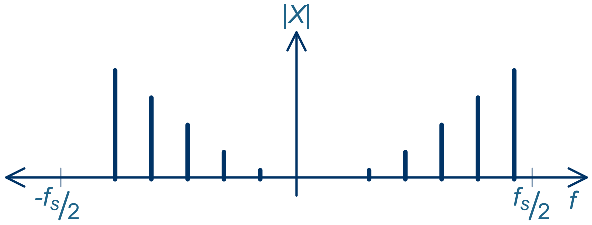
IIR Filter – Injection Test



$f_s = 44.1$ kHz
$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

IIR Filter – Injection Test



$f_s = 44.1$ kHz

$x = 3$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 200 Hz (100 Hz low-pass filter)
- Sweep the complex NCO from DC to 20 kHz

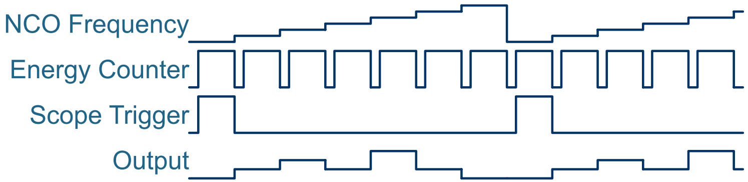
Outline

Energy Counter

$$\begin{aligned} P &= \frac{1}{T_p}\int_{T_p} |x(t)|^2 \, \mathrm{d}t \\ &= \frac{1}{N}\sum_{n=0}^{N-1} (x_n x_n^*) \end{aligned}$$

- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Spectrum Analyser Timing

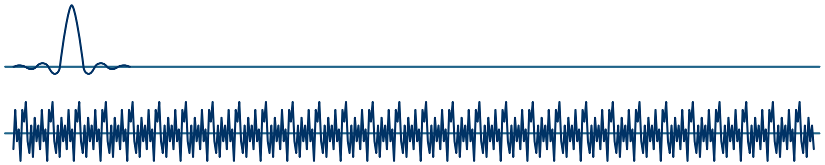


- Many parameters are involved...
- Set the parameters by means of registers:
  let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (e.g. resolution bandwidth selection by constants 1 $\rightarrow$ 5, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Coffee Break...

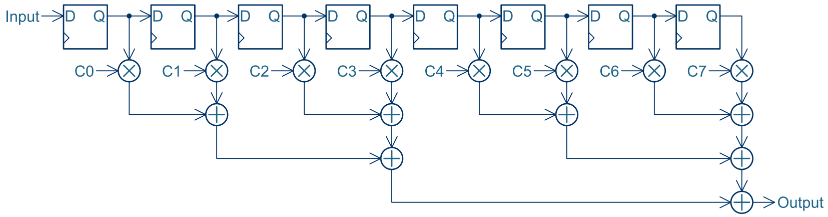
Outline

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 – Implementation

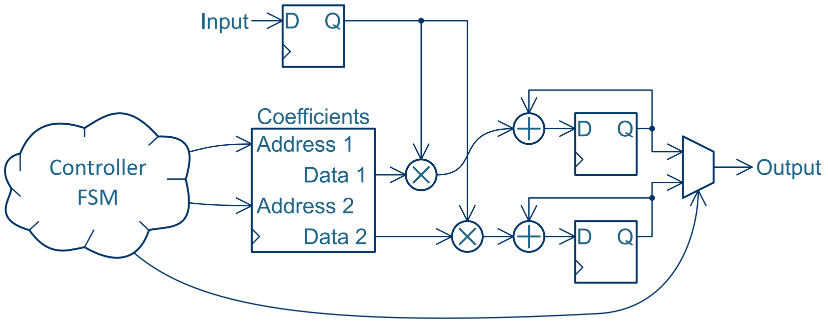


- Move the signal instead of the impulse-response
- To check the direction, inject an impulse (which should produce the impulse-response at the output)
- Pipeline the adder tree to increase the maximum clock frequency

Architecture Design

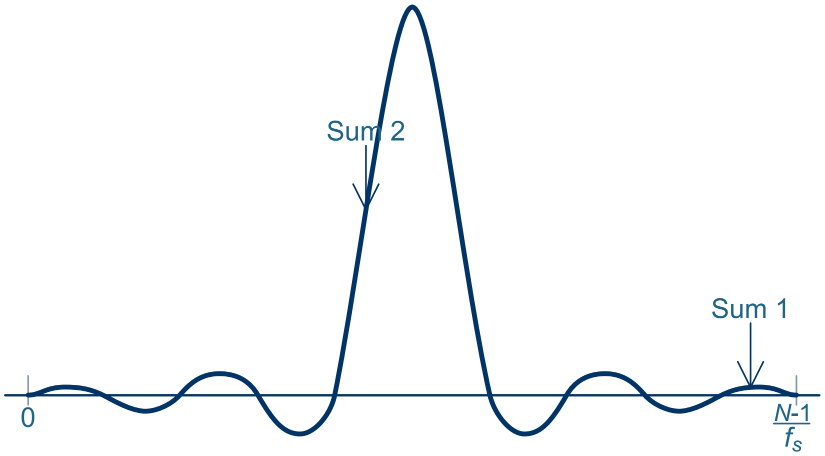
- Always take the design criteria into account when designing an architecture
- What happens when you add decimation
  (i.e. you don’t need an output sample every clock cycle)?
- What if the FPGA clock is much faster than the sample rate?
- What about a combination of the above?
- Always consider the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

Decimation

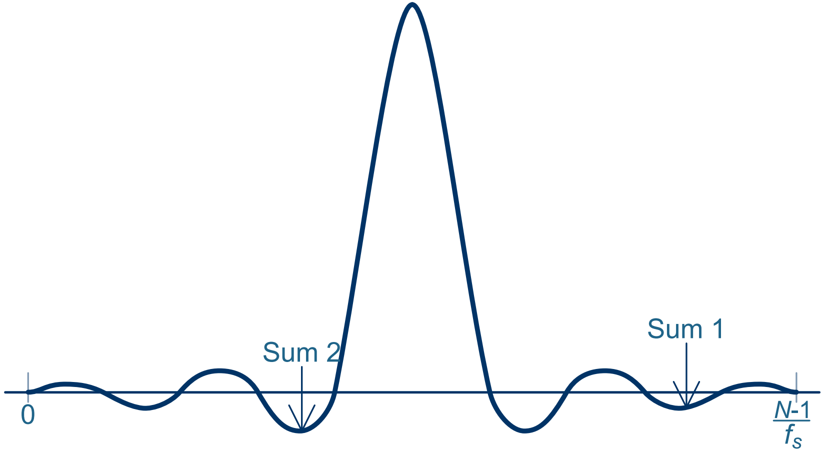


- This example decimates by $N/2$:
  a 512-point filter will decimate by 256
- The coefficient addresses are run $N/2$ out of phase

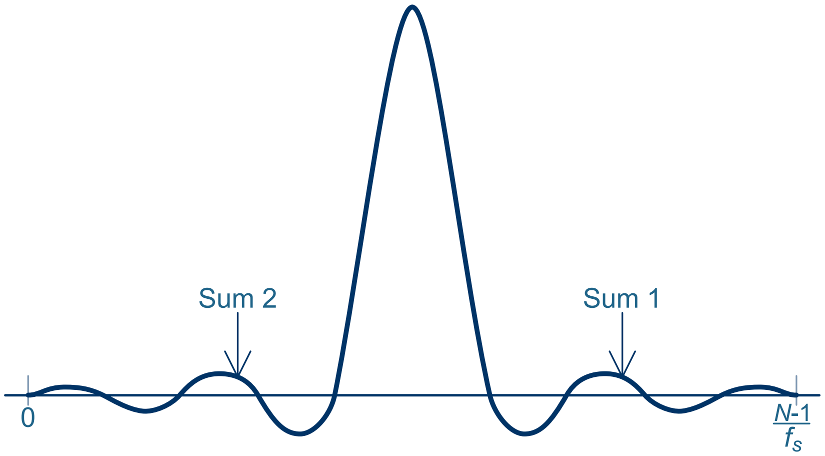
Decimation



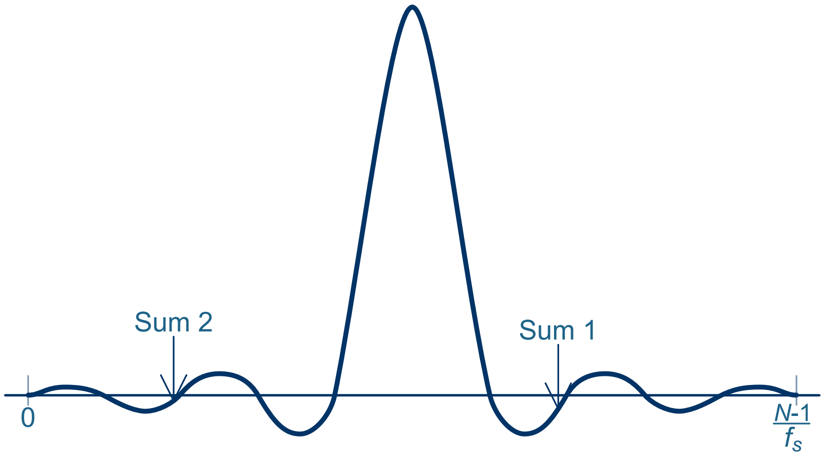
Decimation



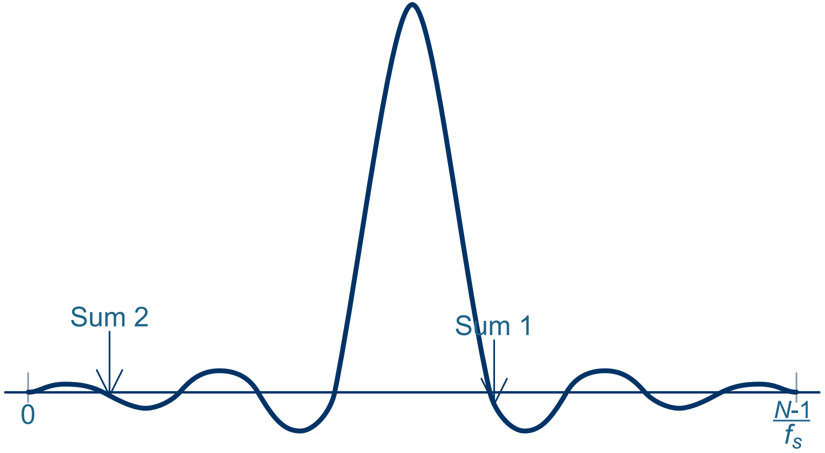
Decimation



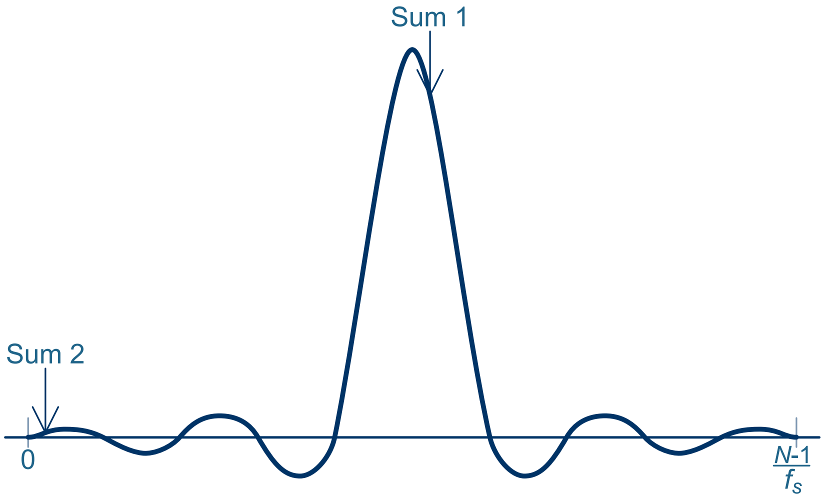
Decimation



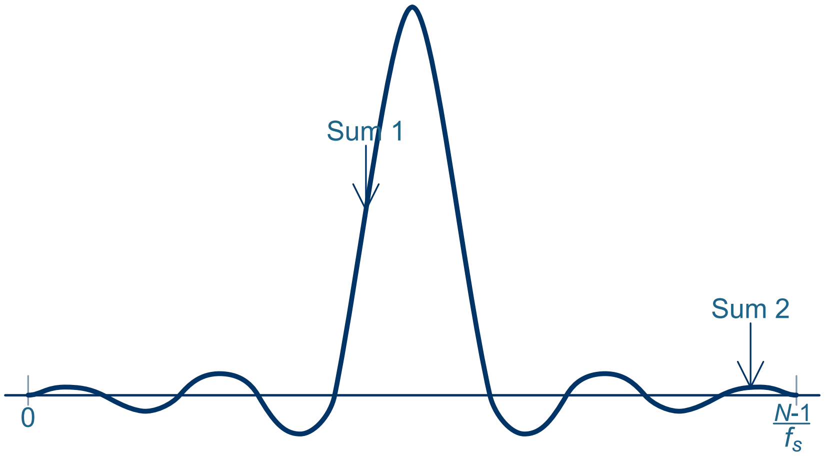
Decimation



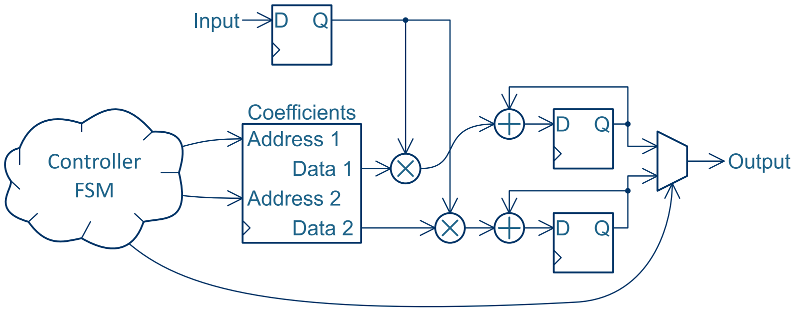
Decimation



Decimation

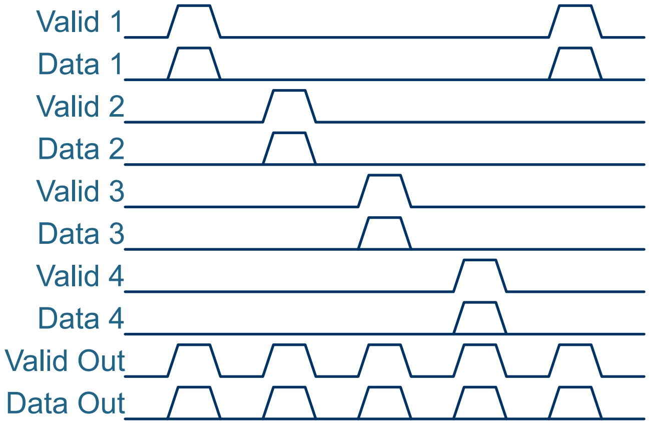


Faster System Clock

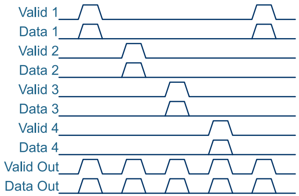


- If the sample clock is lower than the system clock, this same architecture can decimate by less
- Keep more than 2 sums (sometimes more efficient to keep these in BRAM as well)

Combining FIR Filter Units



FIR Filter

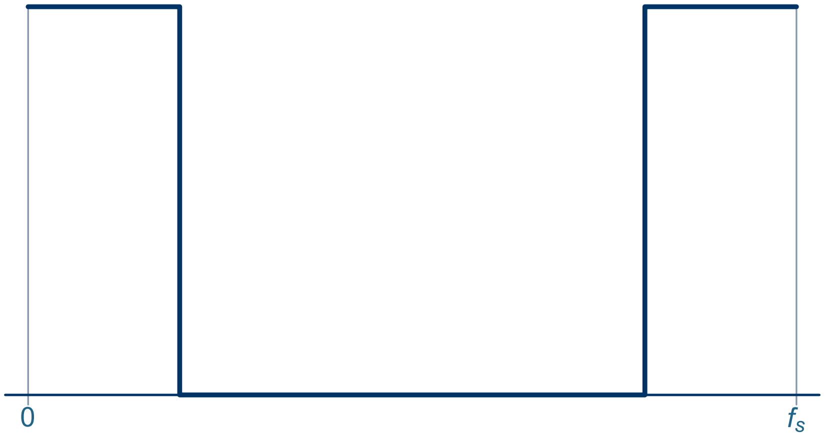


- Use multiple instances of a filter that decimates more than desired
- Reset the counters of the units out of phase
- Combine the outputs with a simple AND-OR circuit

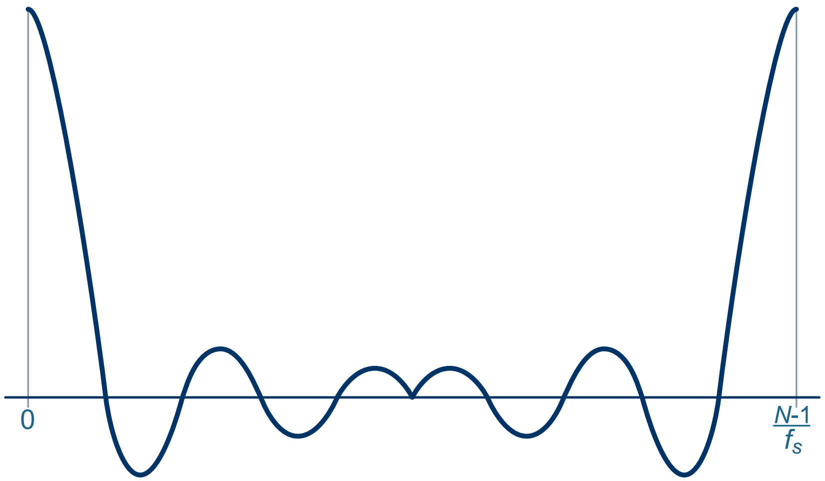
Typical Workflow

- 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

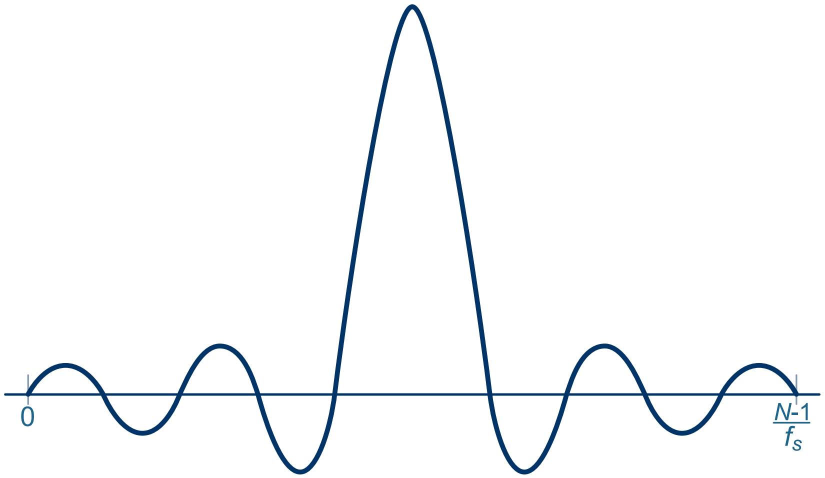
FIR Filter – Desired Response



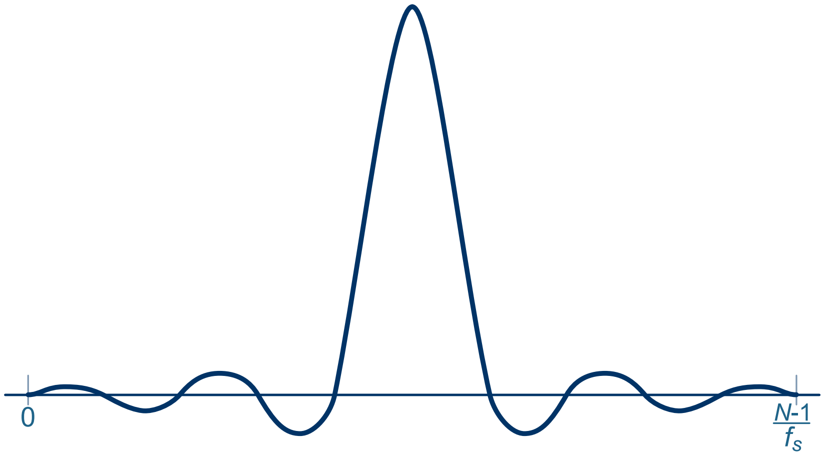
FIR Filter – IFFT (real part)



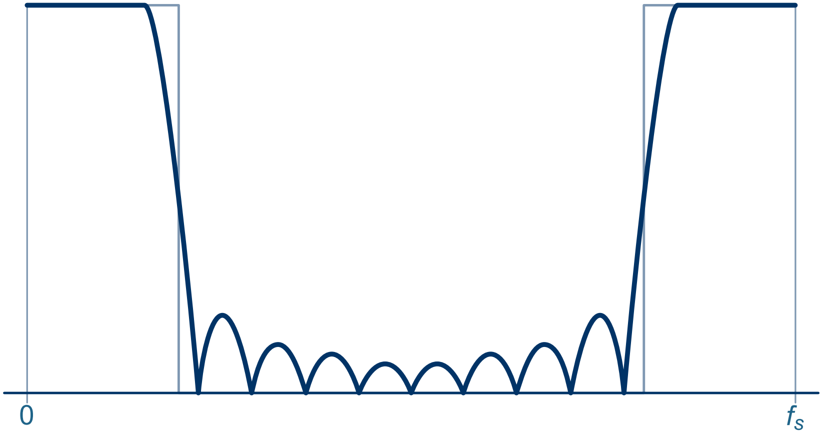
FIR Filter – Time-shifted



FIR Filter – Windowed



FIR Filter – Actual Response



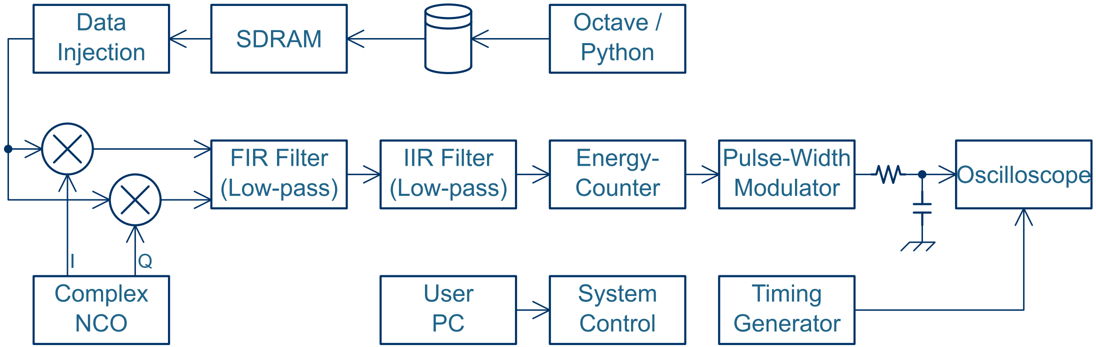
(Zero-pad to see the side-lobes)

FIR Filter Implementation

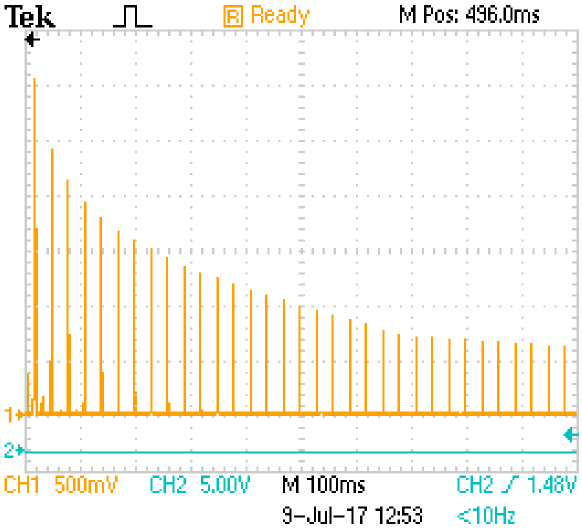
- Implement a N-point FIR filter that decimates by N (i.e. one sample output for every N samples input)
- Use an appropriate cut-off frequency and a Hann window $w(n) = \sin^2\left(\frac{\pi n}{N-1}\right)$
- Use Matlab / Octave / Python to generate the FIR filter constants and memory initialisation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

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

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 (typically over the JTAG)

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 JTAG)

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 JTAG)

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 JTAG)

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 JTAG)

The Report

Your report needs to:

- Describe the design overview
  (use a block diagram with some accompanying text)
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show that you have a working FPGA-based DSP chain
  (use photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to your source code on an accessible Git server (typically GitHub)

The Report

Your report needs to:

- Describe the design overview
  (use a block diagram with some accompanying text)
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show that you have a working FPGA-based DSP chain
  (use photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to your source code on an accessible Git server (typically GitHub)

The Report

Your report needs to:

- Describe the design overview
  (use a block diagram with some accompanying text)
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show that you have a working FPGA-based DSP chain
  (use photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to your source code on an accessible Git server (typically GitHub)

The Report

Your report needs to:

- Describe the design overview
  (use a block diagram with some accompanying text)
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show that you have a working FPGA-based DSP chain
  (use photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to your source code on an accessible Git server (typically GitHub)

The Report

Your report needs to:

- Describe the design overview
  (use a block diagram with some accompanying text)
- Provide minimal detail of the various modules that compose the system
- Provide practical results: performance figures, etc.
- Show that you have a working FPGA-based DSP chain
  (use photos, oscilloscope screenshots, etc. as evidence)
- Provide a link to your source code on an accessible Git server (typically GitHub)

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
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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

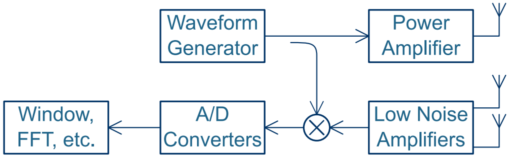
- I'm not on campus, but you can reach me via email
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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
- Alternatively, open an issue on your GitHub fork and at-mention me (jpt13653903)
- Clearly explain what you're trying to do, and what you're struggling with
- Make sure that I can access your source code (typically by means of your Git repo)
- Where applicable, also include 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

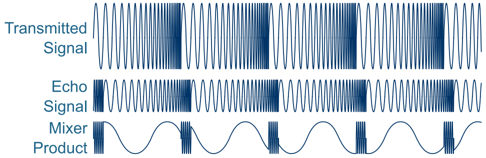
Project 1 – FMCW Front-end



- 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)

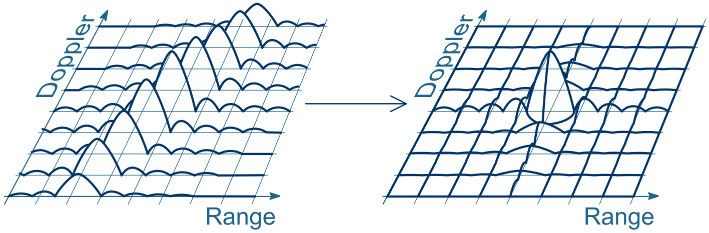
THE RADAR MASTERS COURSE

Project 1 – FMCW Front-end



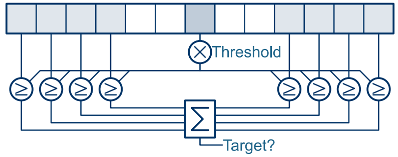
- You can assume that the FPGA generates the transmit sweep triangle
- Take the FFT of each sweep (range FFT),
  organise them into bursts and store the results in SRAM
- Do the FFT in-place instead of streaming,
  which saves on multipliers and RAM blocks
- This is the end of this project – another 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 SRAM
- Play back the corner-turned data and take the Doppler FFTs
- Store the resulting range-Doppler maps in SRAM for the next step

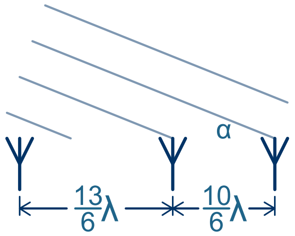
Project 3 – Range CFAR



- Simulate the data output of Project 2
  and inject the anticipated result into the SRAM
- 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 SRAM

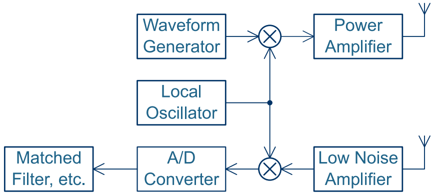
THE RADAR MASTERS COURSE

Project 4 – Angle Extraction



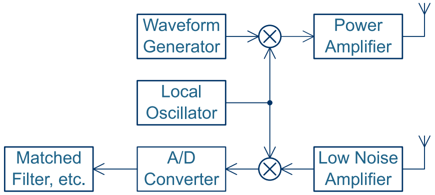
- Simulate the data output of Project 3
  and inject the anticipated result into the SRAM
- Play back the target stream and perform angle extraction for the sparse array

Project 5 – Pulsed Front-end



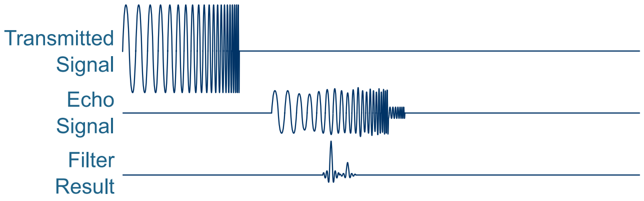
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI's on the oscilloscope (Range is about 6.7 µs/km, so it is practical to output the signal from a long-range RADAR (350 km or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

Project 5 – Pulsed Front-end



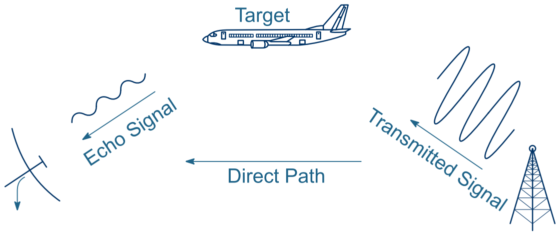
- Design and implement a chirped pulse RADAR front-end
- Inject raw ADC data and implement a matched filter by means of convolution (essentially a FIR filter)
- Display the PRI’s on the oscilloscope
- The rest of a typical processing chain is conceptually similar to projects 2 to 4, which can be adapted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

Project 5 – Pulsed Front-end



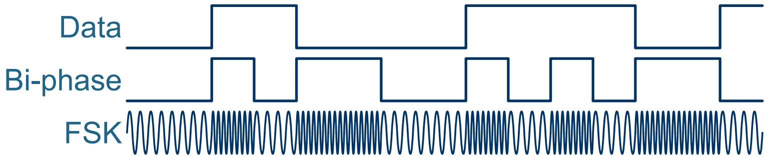
- You can assume that the FPGA generates the transmit timing control, so you can use the SRAM address to determine where you are in the current PRI
- After doing matched filtering,
  organise the results into bursts and store them in SRAM
- This is the end of this project – another project could potentially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

Project 6 – Commensal RADAR



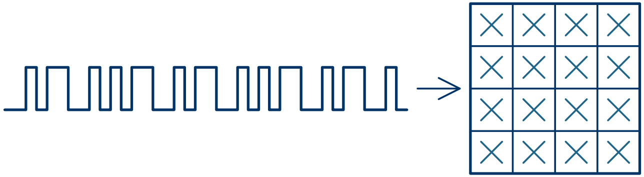
- Design and implement a commensal RADAR front-end
- Inject raw ADC data and implement range extraction (correlate the direct path with the echo signal)

Project 7 – FSK Communication



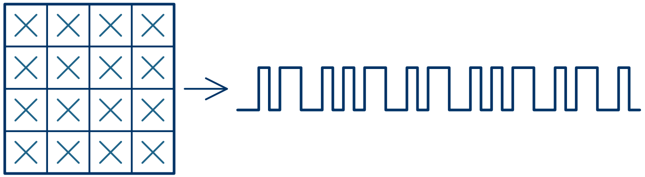
- Implement an FSK-based, bi-phase-coded communication channel
- Use S/PDIF as inspiration, with synchronisation word, etc.
- Inject simulated ADC data and demodulate by whatever means is convenient (matched filter, most likely)
- Display the received bit-stream on the oscilloscope
- Analyse channel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

Project 8 – QAM Modulator



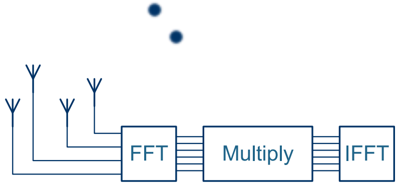
- Design and implement a 16-QAM modulator
- Inject a data stream and produce a 16-QAM output stream
- Including a synchronisation header and run-length limit
- Store the resulting modulated signal in SRAM

Project 9 – QAM Demodulator



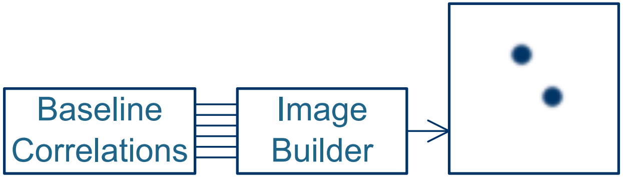
- Simulate the data output of Project 8
  and inject the anticipated result into the SRAM
- Assume an ideal RF front-end
  (i.e. no need to perform carrier-recovery)
- Play back the data stream and recover synchronisation
- Demodulate the data stream to obtain the original data

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) and 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

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

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. Quartus Signal-tap, Vivado Chip-scope, Diamond Reveal, 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 vendor tool-chains 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

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 Dr John-Philip Taylor

Convened by Dr Stephen Paine

Day 5 – 13 September 2024