FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Dr Stephen Paine

Day 5 – 3 May 2022

Outline

Injection Testing

Advanced Simulation

IIR Filters

Practical – Spectrum Analyser

Energy Counter

Outline

Injection Testing

Advanced Simulation

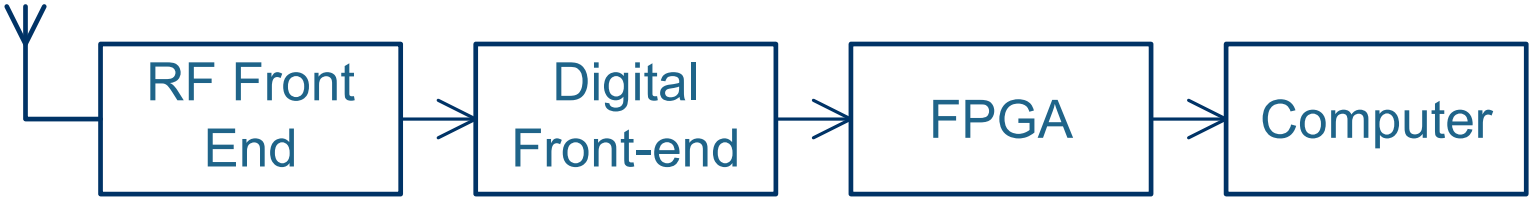
IIR Filters

Practical – Spectrum Analyser

Energy Counter

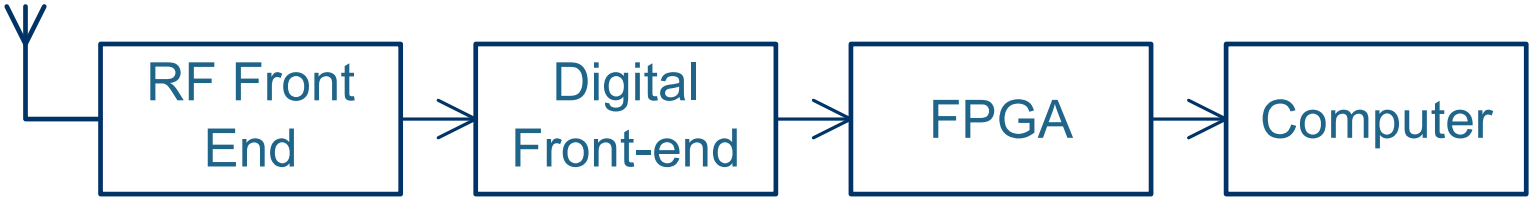
THE RADAR MASTERS COURSE

Typical Development Challenges



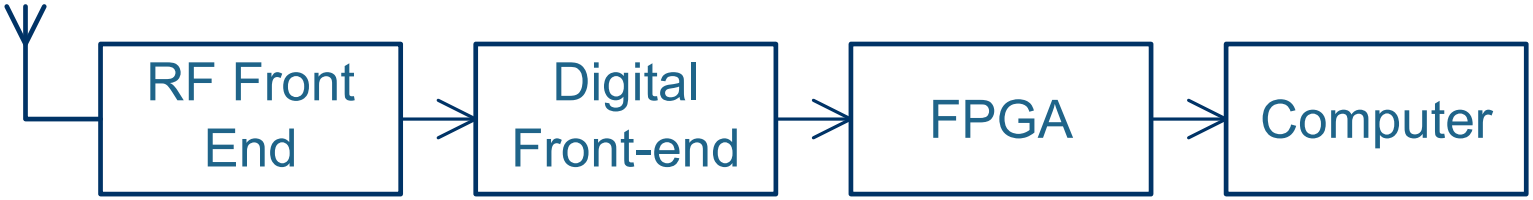
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



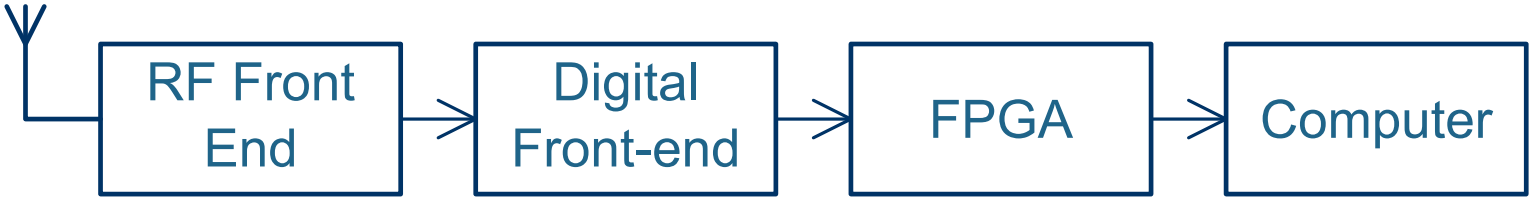
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



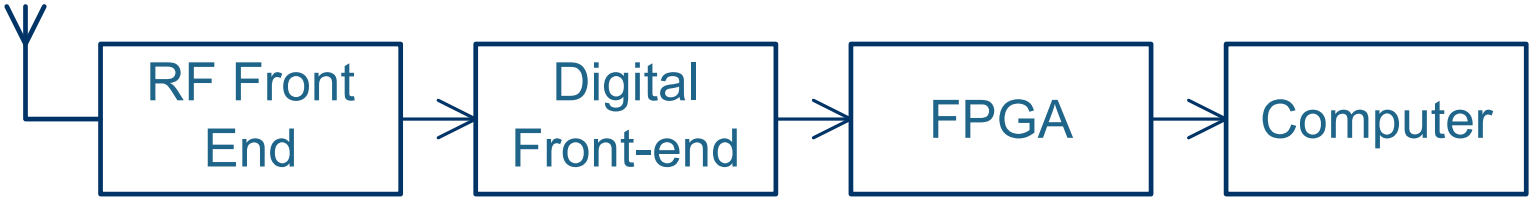
- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Typical Development Challenges



- The hardware (RF front-end and DFE) is not available at the time of FPGA firmware development
- There is no real-world data available
- Lab-generated signals are not suitable for DSP-chain testing
- Field tests are short-term experiments and expensive to run
- The DSP algorithms are experimental and uncertain

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

Early Development

- The system is simulated
- The simulated ADC data is injected into the FPGA
- The DSP-chain is developed on this simulated data
- At first, the computer-based DSP does most, if not all, of the processing (FPGA firmware consist only of supporting infrastructure)
- As the algorithms mature, the functionality is moved into the FPGA

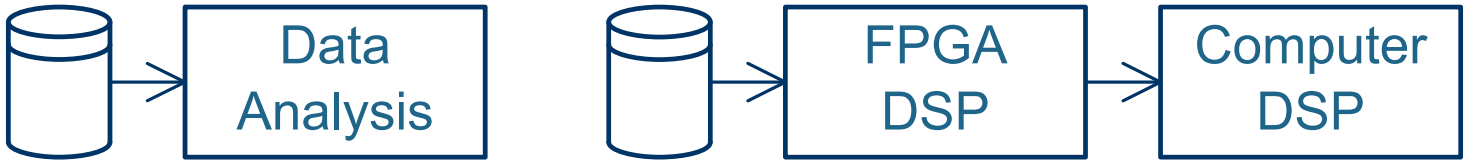
Late Development

- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development

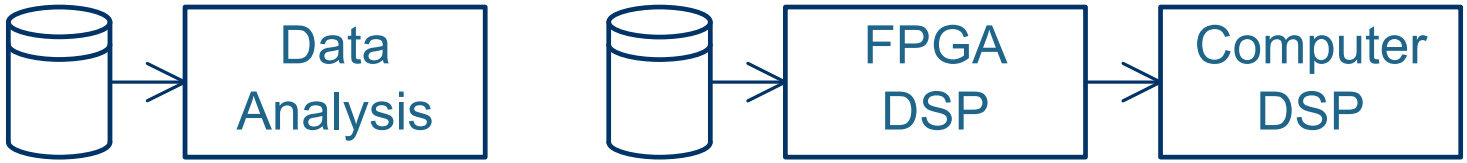
- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development



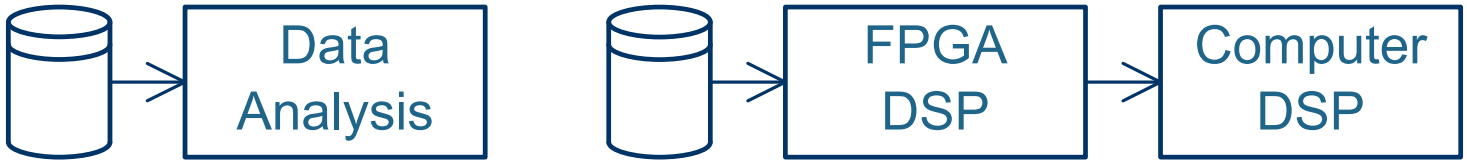
- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development



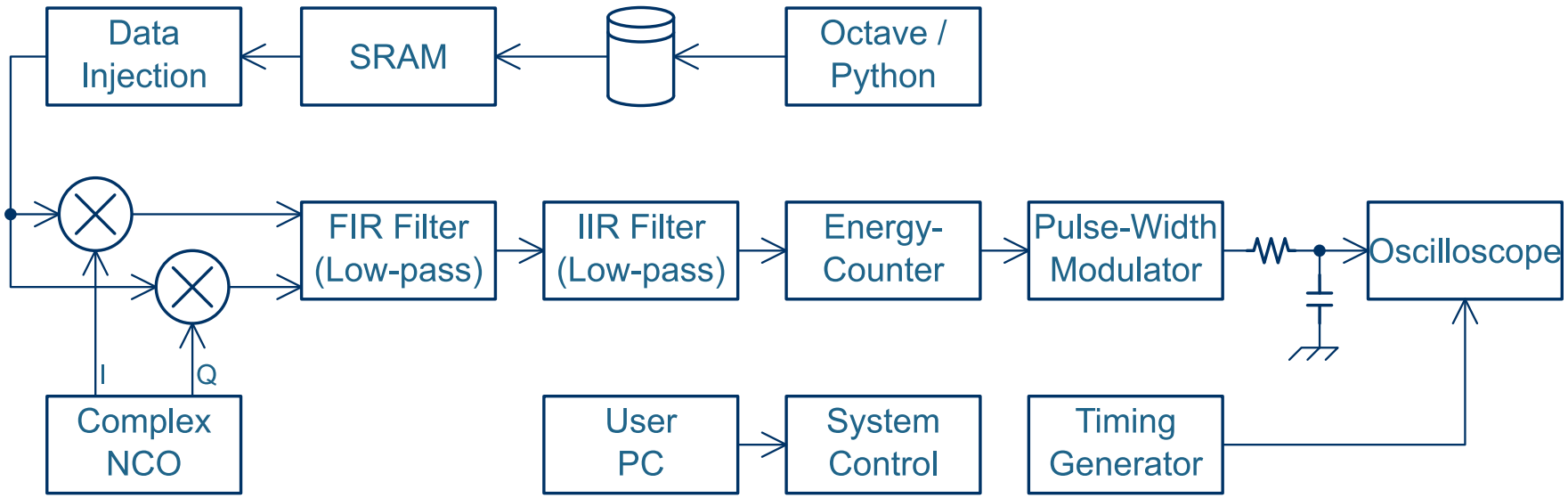
- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

Late Development

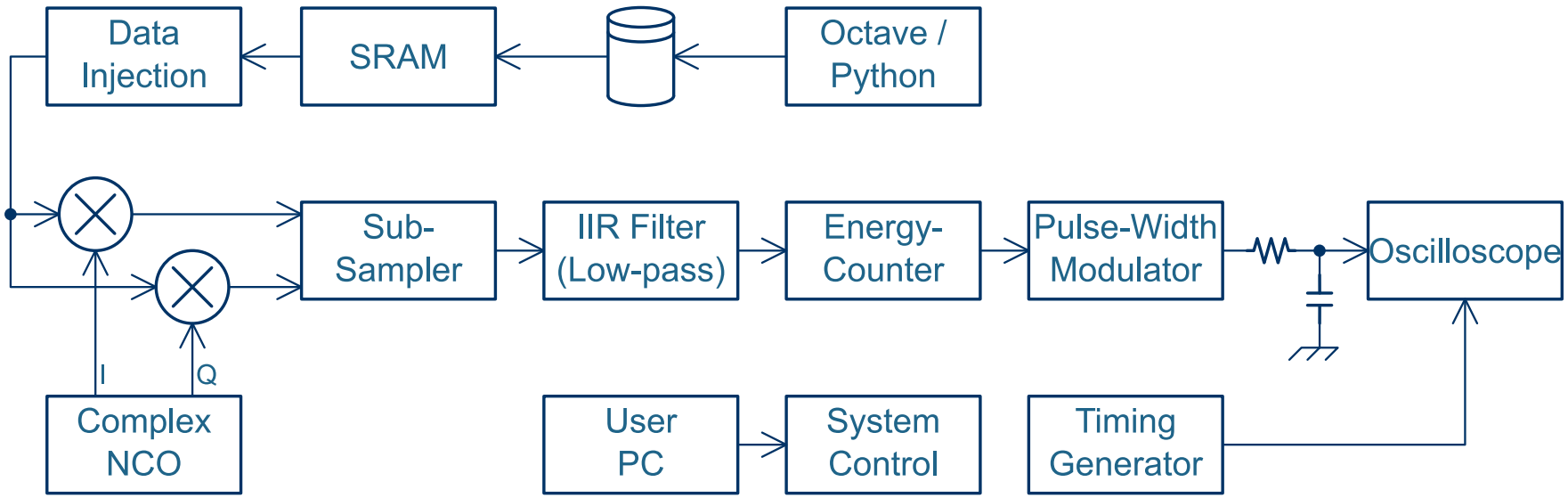


- The front-end hardware is built, but must be tested
- An FPGA-based logger is implemented that saves raw ADC data from a field trial
- This recorded data is analysed in order to test the hardware and injected into the FPGA to test the DSP-chain
- This same recorded data can be used in simulation...

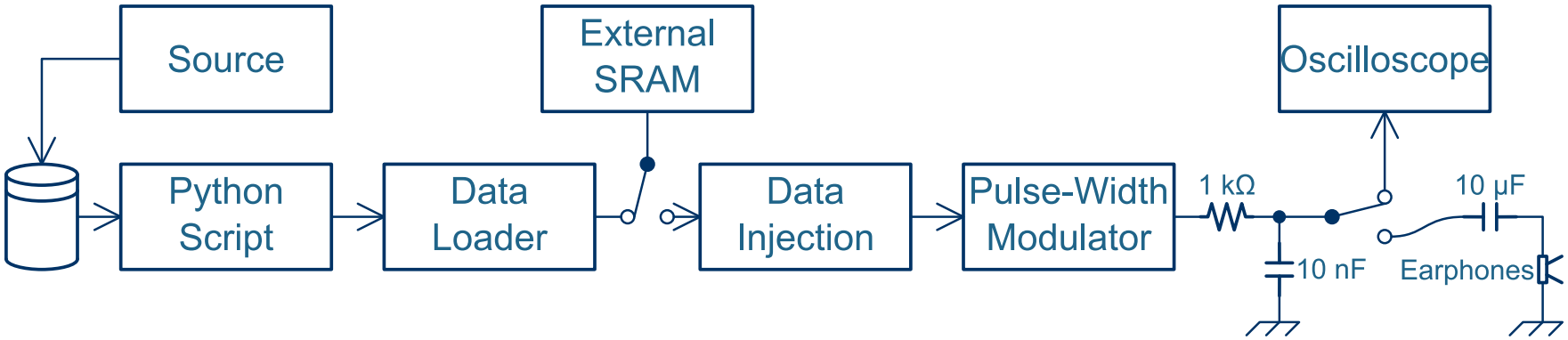
Practical 11 – External FIFO



Practical 11 – External FIFO



Data Injection



Outline

Injection Testing

Advanced Simulation

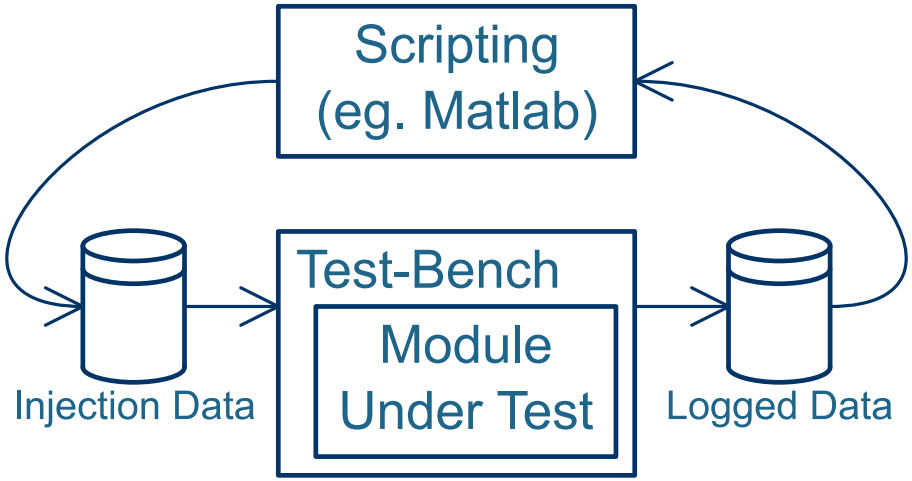
IIR Filters

Practical – Spectrum Analyser

Energy Counter

THE RADAR MASTERS COURSE

Advanced Simulation



File I/O


```
`timescale 1ns/1ns
module File_IO_TB;

integer Input_File;
integer Output_File;

initial begin
  Input_File  = $fopen("Test_File.txt", "rb");
  Output_File = $fopen("Output.txt"   , "wb");
  $fwrite(
    Output_File,
    "Time [ns], Data [Hex], Data [Binary]\n"
  );
end

reg ipClk = 0; always #5 ipClk <= !ipClk; // 100 MHz
```

File I/O


```
reg [7:0]Byte; reg [15:0]Data; integer Result;

always @(posedge ipClk) begin
  Result = $fread(Byte, Input_File); Data[ 7:0] = Byte;
  Result = $fread(Byte, Input_File); Data[15:8] = Byte;

  if(Result < 1) begin
    $fclose(Input_File); $fclose(Output_File); $stop;
  end

  $fwrite(Output_File, "%d, %04X, %s\n", $time, Data, Data);
end

endmodule
```

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

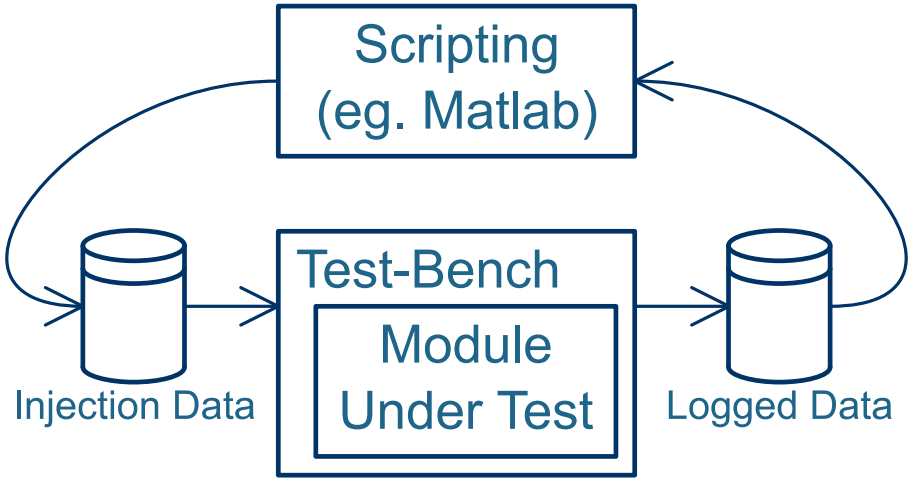
- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

File I/O

Consult the Verilog-2001 standard:

- Section 17.2.4: “Reading data from a file”
- Section 17.2.8: “Loading memory data from a file”
- `$fread` reads any size register, but in big-endian
- `$fscanf` reads formatted data, including raw binary
- `$fwrite` works similar to `fprintf` from C, except that `%u` writes raw binary data
- `$fmonitor` logs to file whenever the signals change

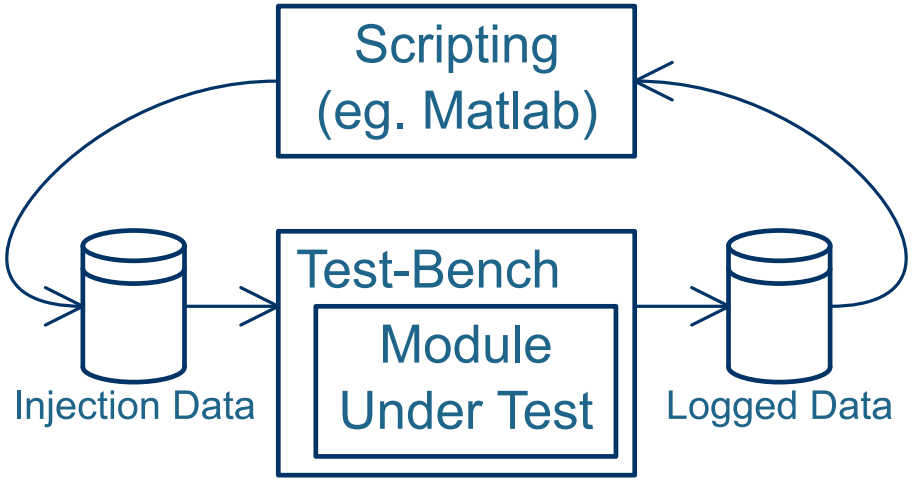
Data Injection (ADXL345_TB)



- Create a test file “ADXL345_TB.dat”:

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Data Injection (ADXL345_TB)



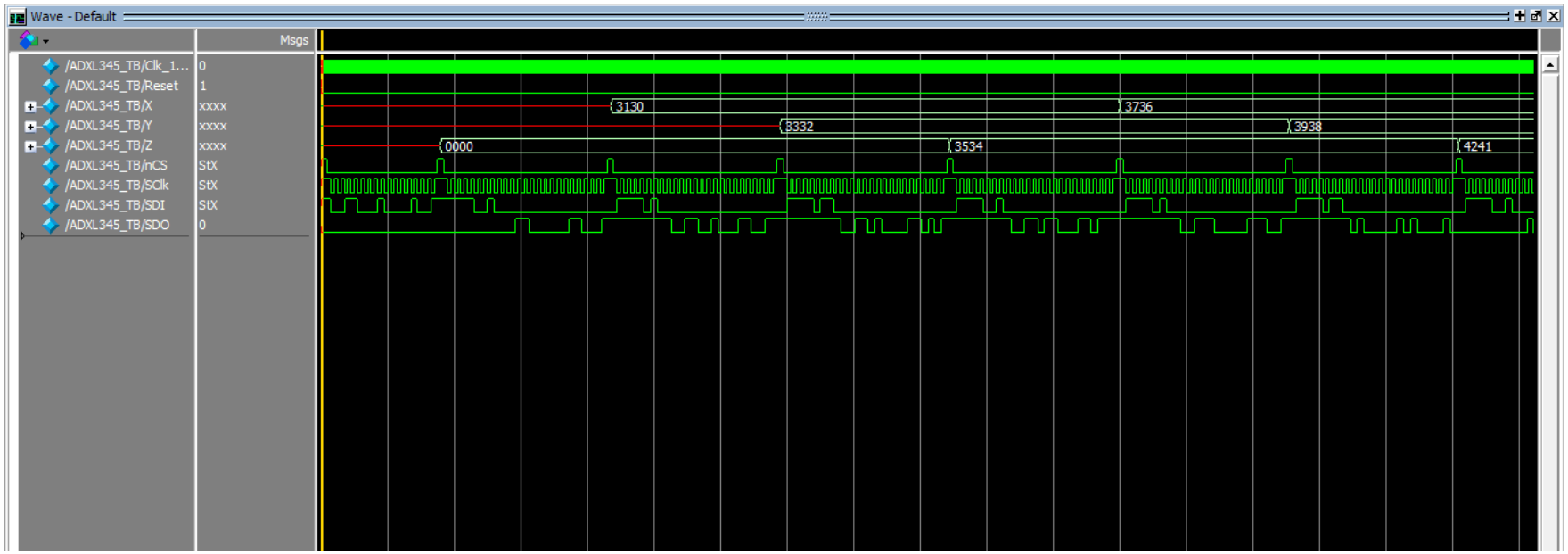
- Create a test file “ADXL345_TB.dat”:

```
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

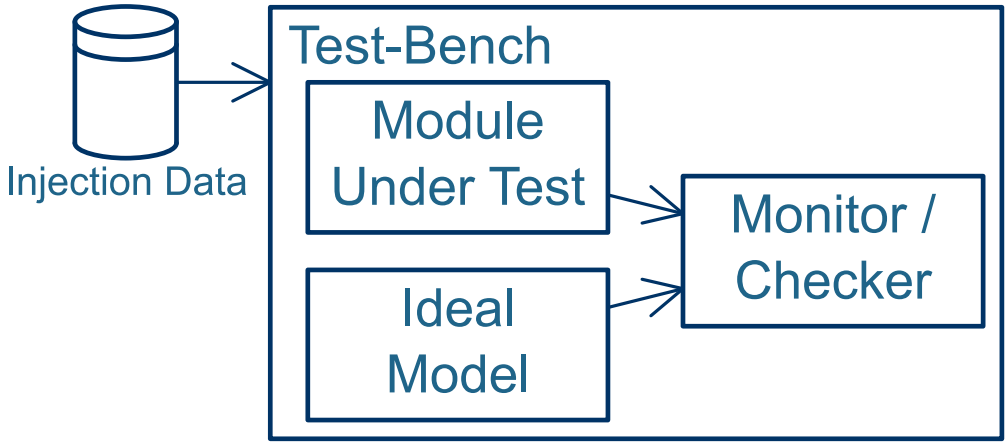
Data Injection (ADXL345_TB)


```
integer   File, n;
reg [15:0]Data;

initial begin
  File = $fopen("../Peripherals/ADXL345_TB.dat", "rb");

  for(n = 0; n < 16; n = n + 1) @(negedge SClk);
  forever begin
    for(n = 0; n <  8; n = n + 1) @(negedge SClk);
    $fread(Data, File);
    for(n = 0; n < 16; n = n + 1) begin
      @(negedge SClk);
      #40 {SDO, Data[15:1]} <= Data;
    end
  end
end
```

Data Injection (ADXL345_TB)

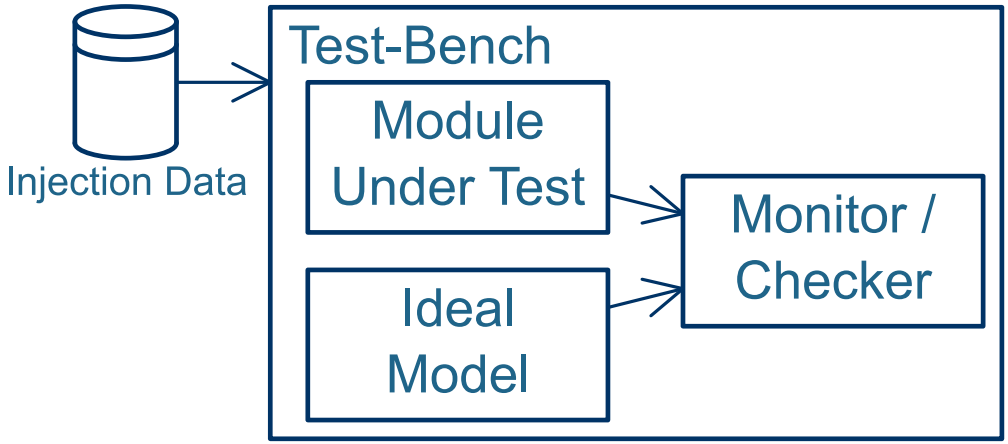


Automated Test-benches



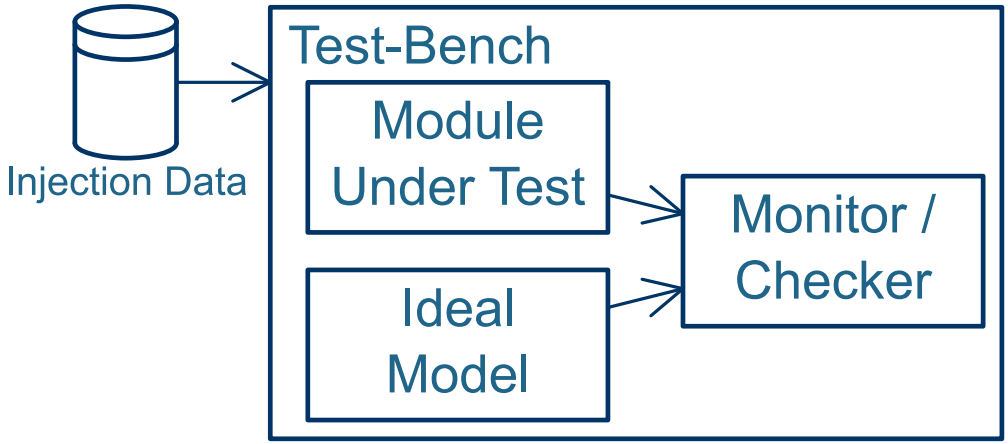
- Very useful, but time-consuming to write...
- The “Ideal Model” can be read from a file, or calculated in the test-bench
- The “Monitor” issues errors (with `$error`), warnings (with `$warning`) or information (with `$info`) as appropriate

Automated Test-benches



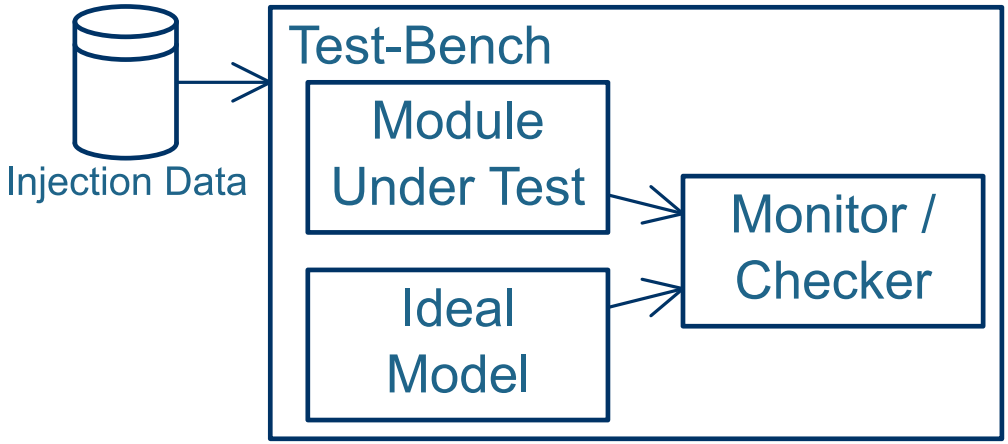
- Very useful, but time-consuming to write...
- The “Ideal Model” can be read from a file, or calculated in the test-bench
- The “Monitor” issues errors (with `$error`), warnings (with `$warning`) or information (with `$info`) as appropriate

Automated Test-benches



- Very useful, but time-consuming to write...
- The “Ideal Model” can be read from a file, or calculated in the test-bench
- The “Monitor” issues errors (with `$error`), warnings (with `$warning`) or information (with `$info`) as appropriate

Automated Test-benches



- Very useful, but time-consuming to write...
- The “Ideal Model” can be read from a file, or calculated in the test-bench
- The “Monitor” issues errors (with `$error`), warnings (with `$warning`) or information (with `$info`) as appropriate

Coffee Break...

Outline

Injection Testing

Advanced Simulation

IIR Filters

Practical – Spectrum Analyser

Energy Counter

THE RADAR MASTERS COURSE

IIR Filter

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

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

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

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

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

IIR Filter

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

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

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

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

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

IIR Filter

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

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

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

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

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

IIR Filter

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

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

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

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

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

IIR Filter

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

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

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

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

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

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'd_8_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'd_8_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:

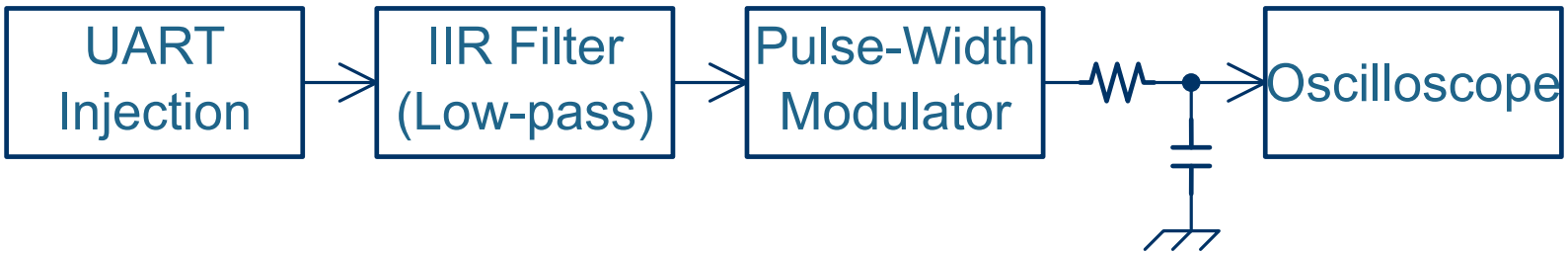
```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

IIR Filter

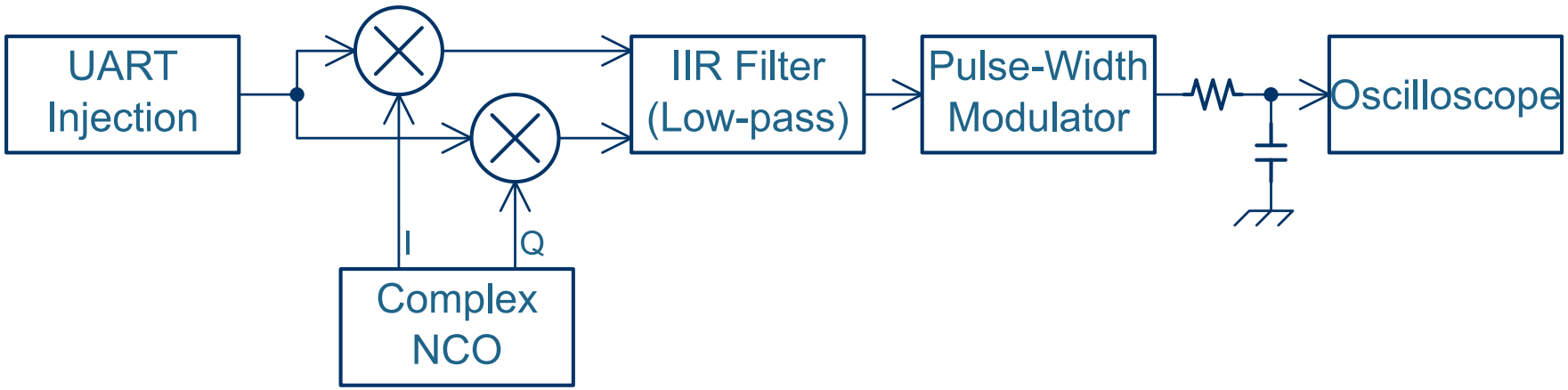
- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Be careful with the signedness of the operation
- To help you keep track of what bit represents what in fixed-point representations, use negative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

```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_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;
```

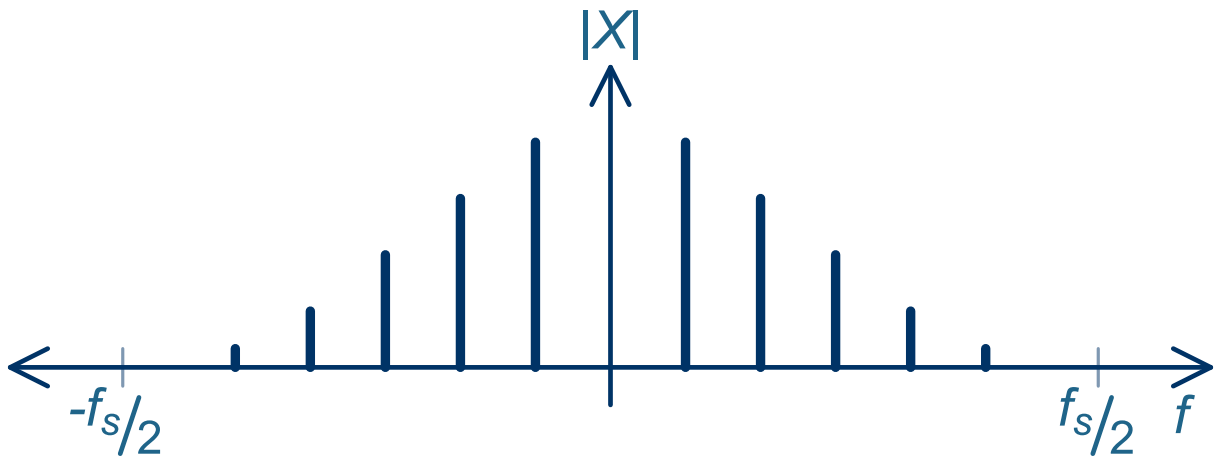
IIR Filter – ADC Test (Optional)



IIR Filter – ADC Test (Optional)



IIR Filter – Injection Test

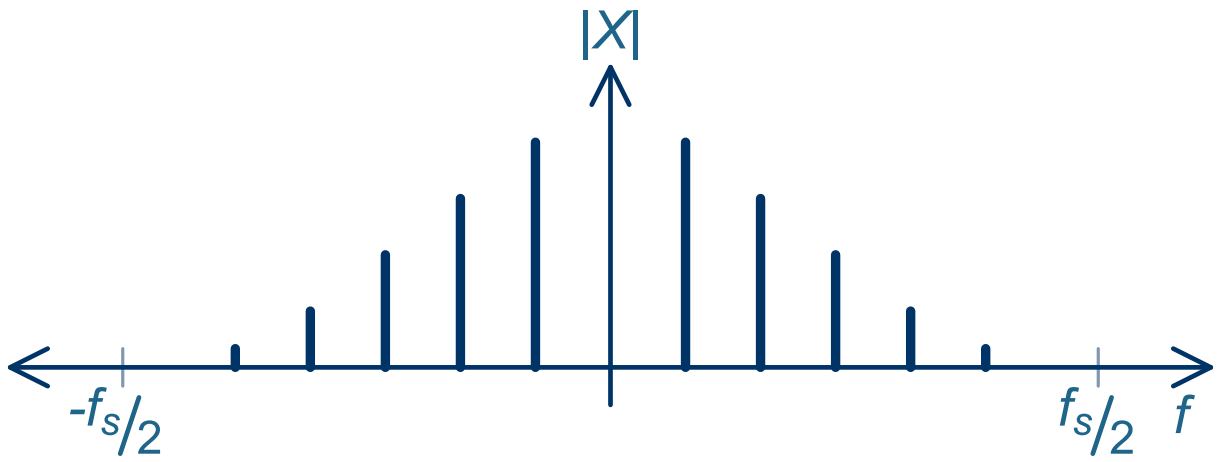


$f_s = 44.1$ kHz

$x = 3$ 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

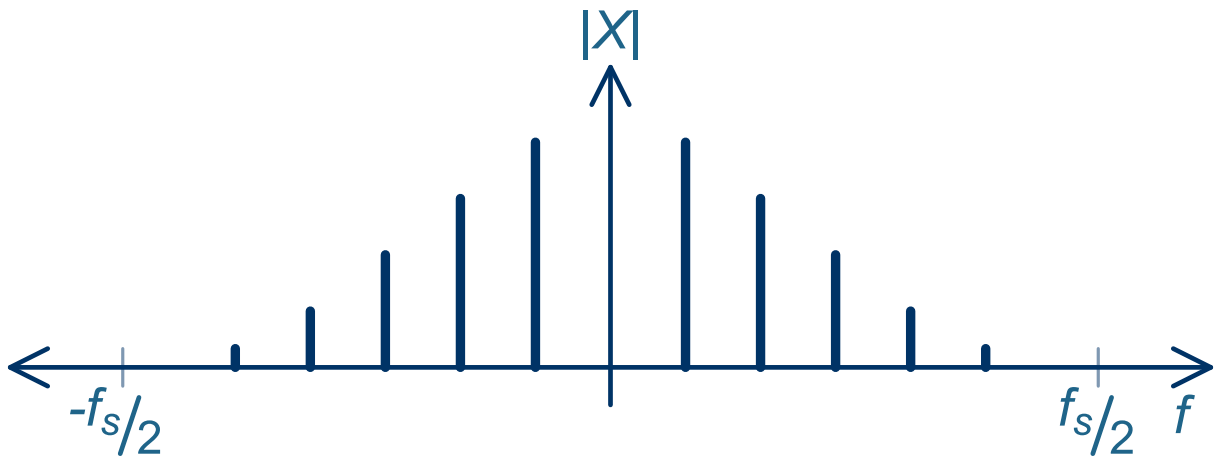


$f_s = 44.1$ kHz

$x = 3$ 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

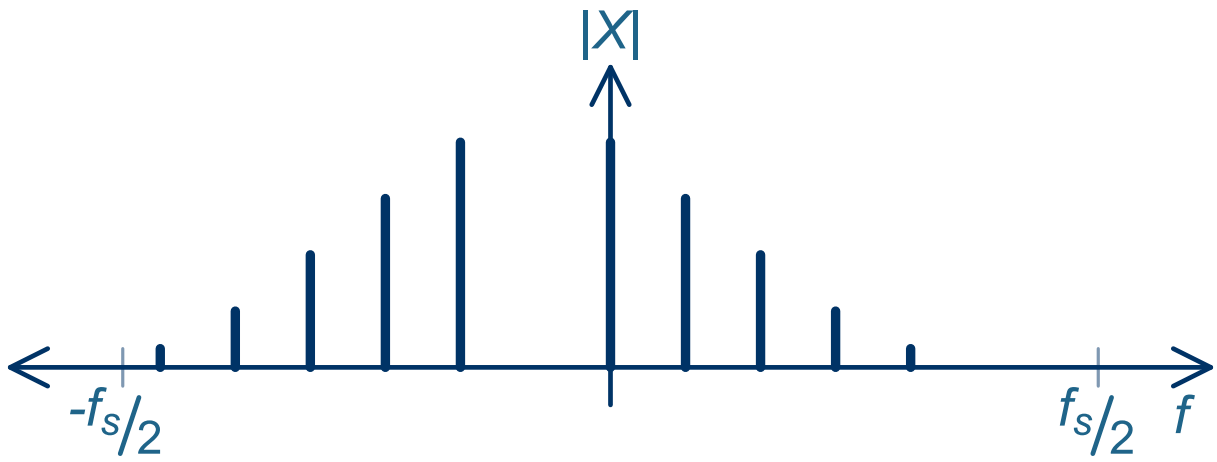


$f_s = 44.1$ kHz

$x = 3$ 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

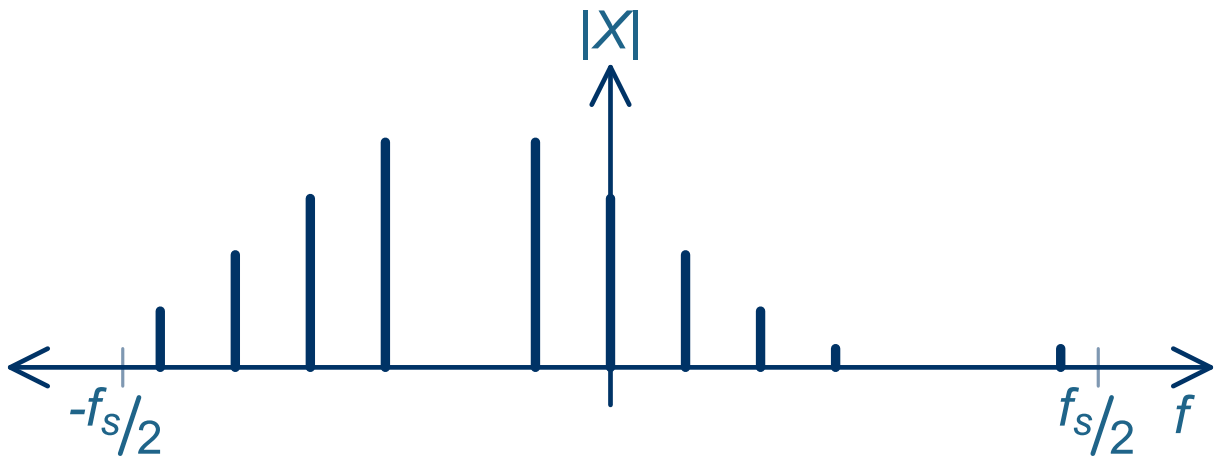


$f_s = 44.1$ kHz

$x = 3$ 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

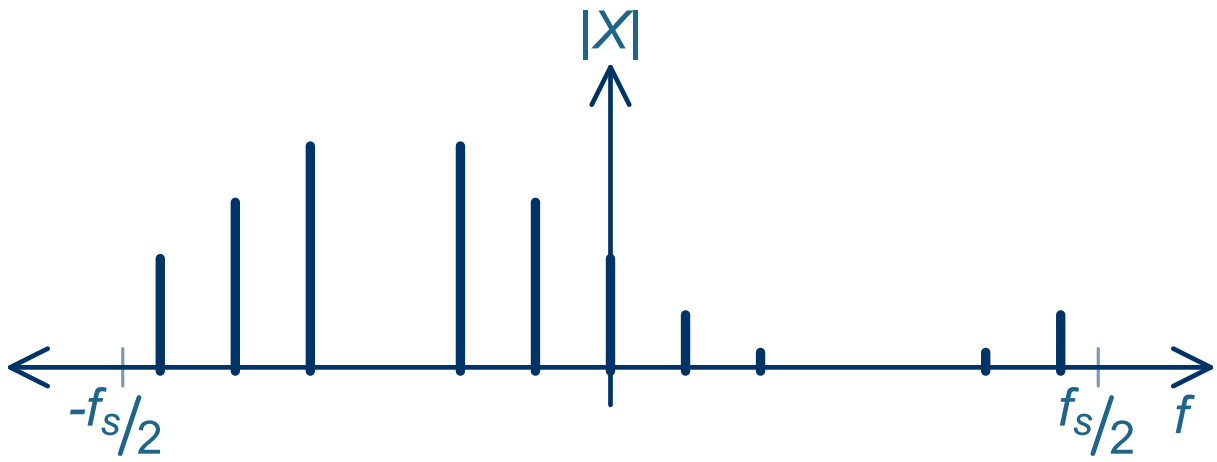


$f_s = 44.1$ kHz

$x = 3$ 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

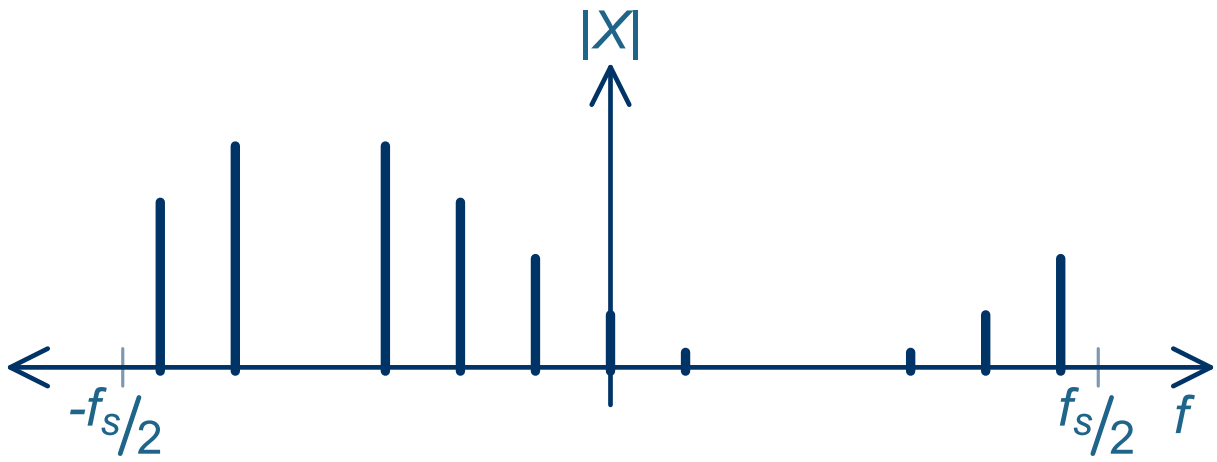


$f_s = 44.1$ kHz

$x = 3$ 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

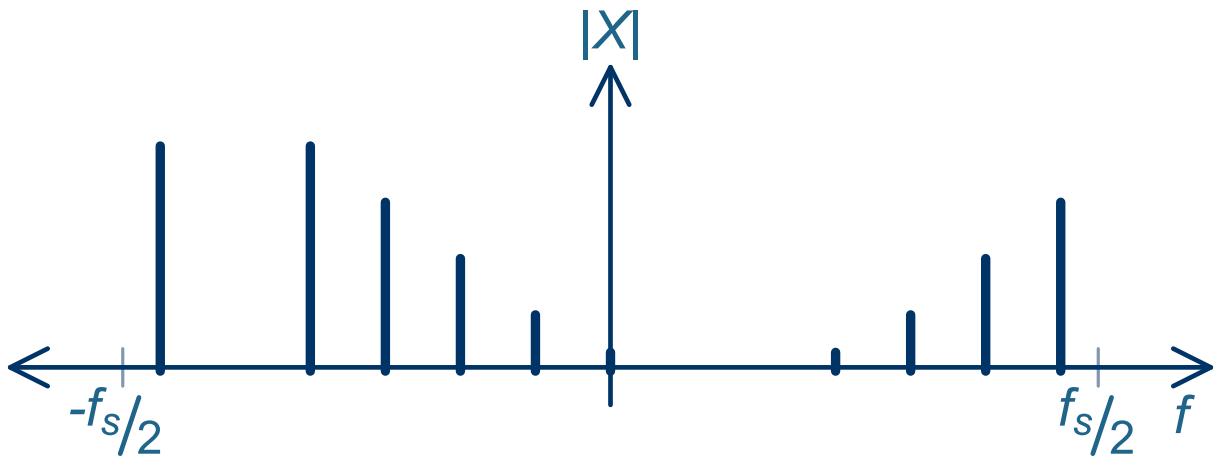


$f_s = 44.1$ kHz

$x = 3$ 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

IIR Filter – Injection Test

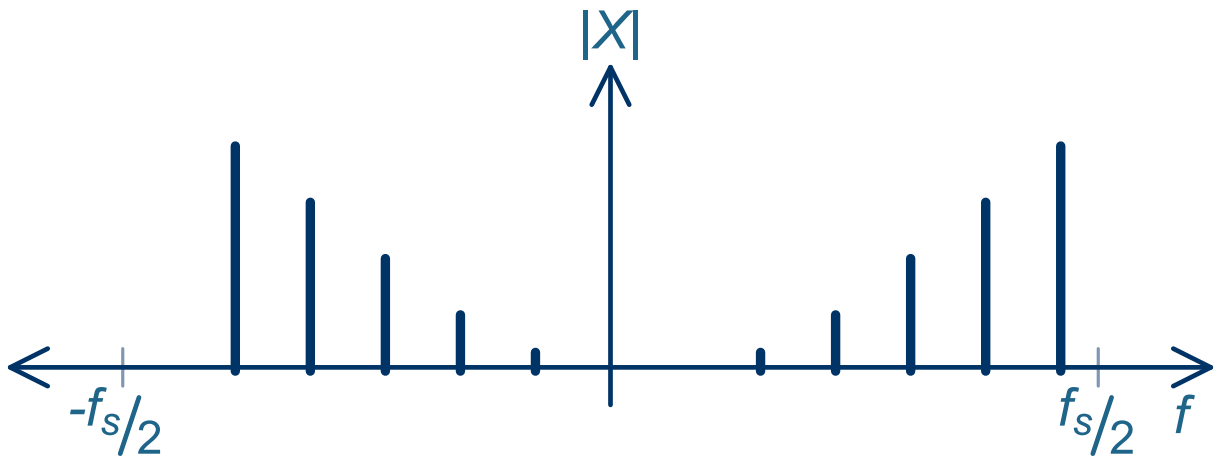


$f_s = 44.1$ kHz

$x = 3$ 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

Injection Testing

Advanced Simulation

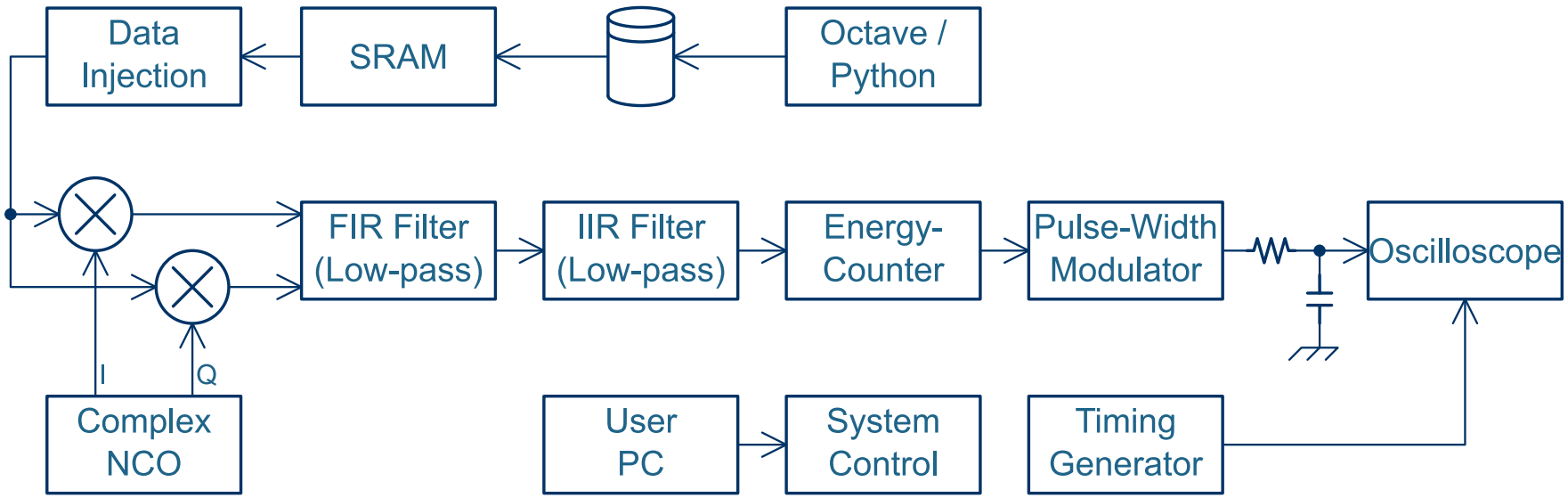
IIR Filters

Practical – Spectrum Analyser

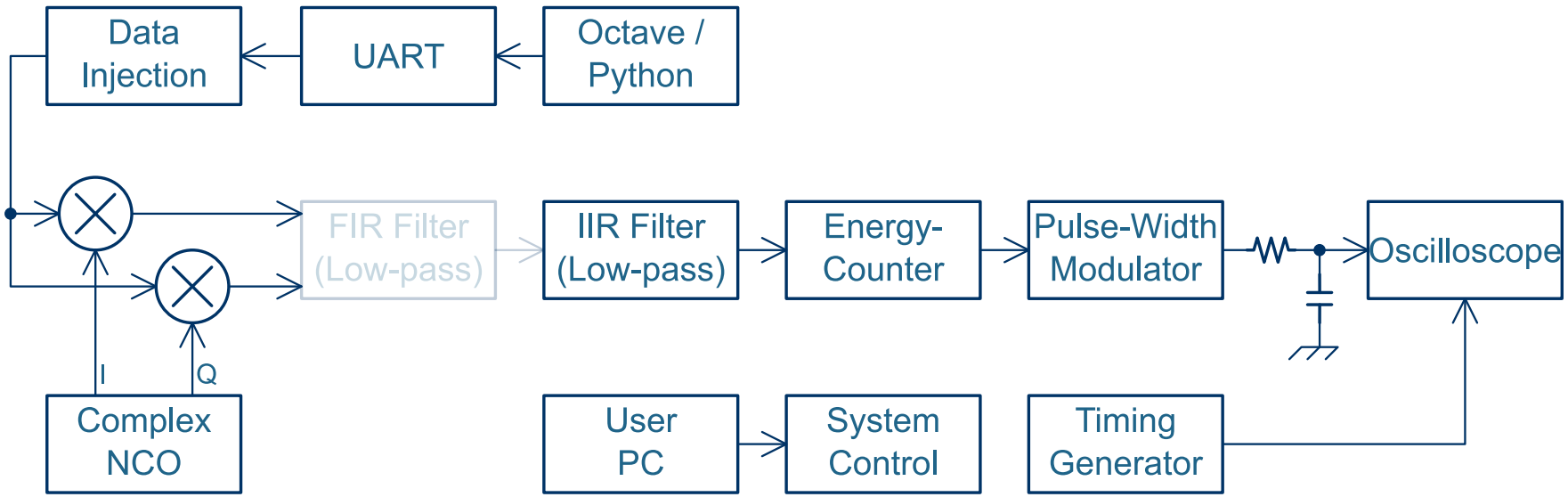
Energy Counter

THE RADAR MASTERS COURSE

Practical – Spectrum Analyser



Practical – Spectrum Analyser



Outline

Injection Testing

Advanced Simulation

IIR Filters

Practical – Spectrum Analyser

Energy Counter

THE RADAR MASTERS COURSE

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

Energy Counter

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

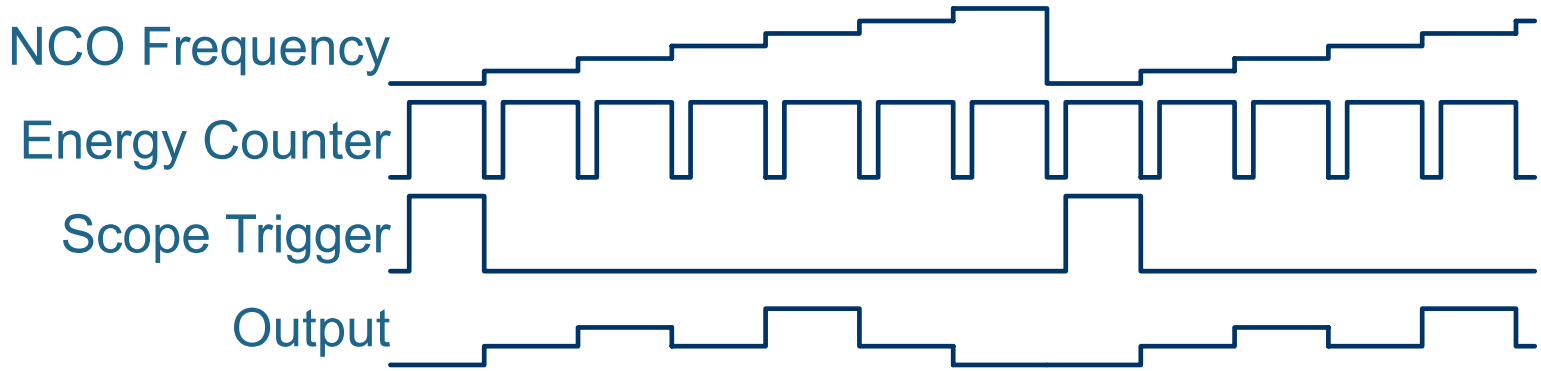
- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Energy Counter

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

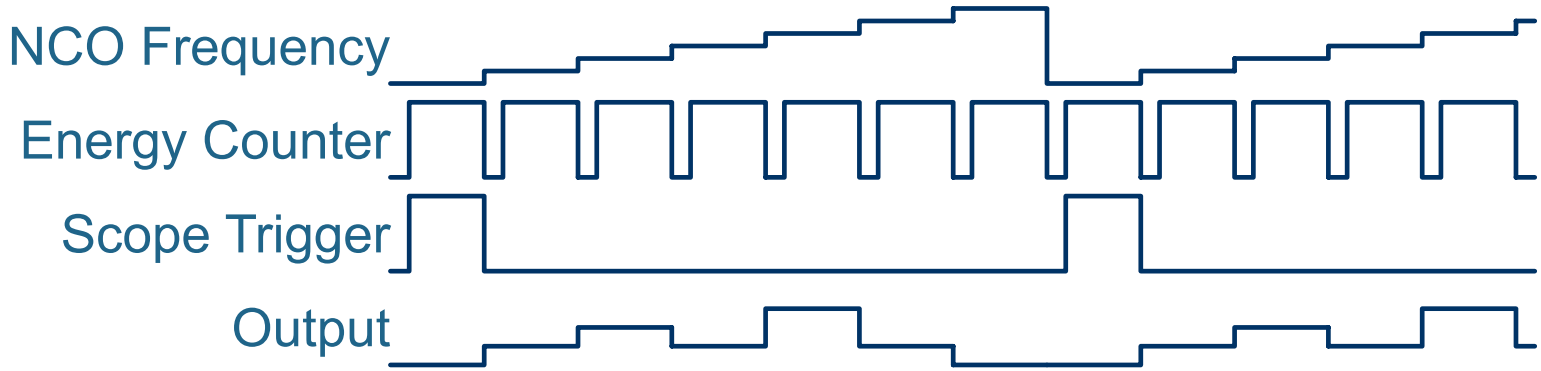
- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Timing Generator



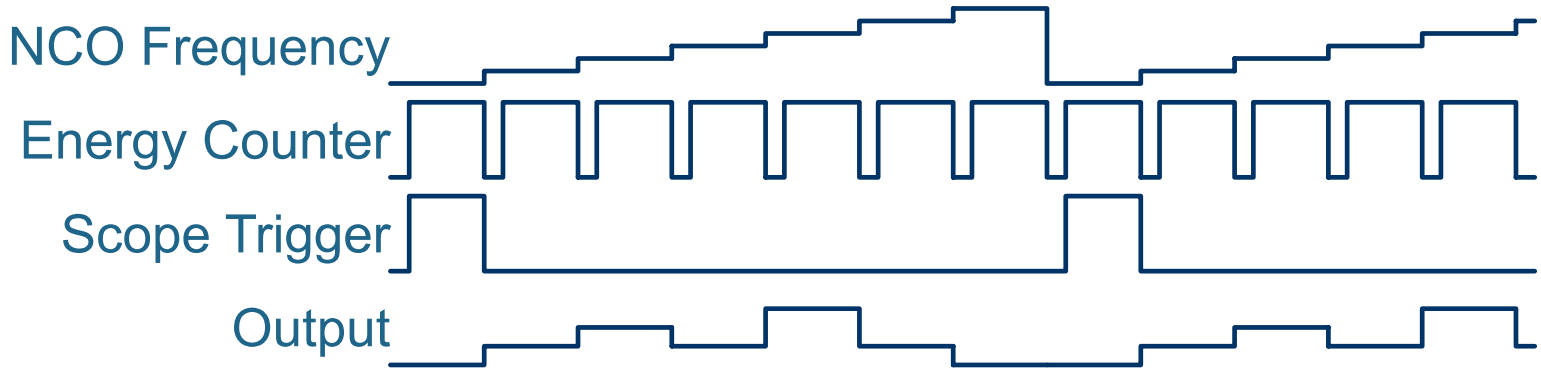
- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants $1 \rightarrow 5$, translated to the filter constants, energy-counter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

Timing Generator



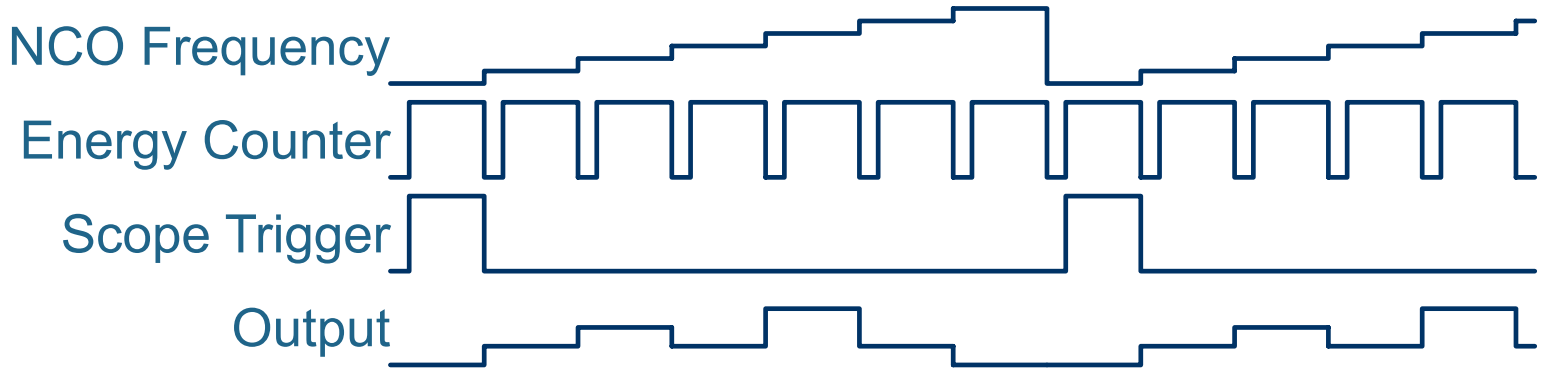
- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants $1 \rightarrow 5$, translated to the filter constants, energy-counter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

Timing Generator



- 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 (eg. resolution bandwidth selection by constants 1 $\rightarrow$ 5, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Timing Generator



- 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 (eg. resolution bandwidth selection by constants 1 $\rightarrow$ 5, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Select References

Stephen Brown and Zvonko Vranesic
Fundamentals of Digital Logic with Verilog Design, 2nd Edition
ISBN 978-0-07-721164-6

Merrill L Skolnik
Introduction to RADAR Systems
ISBN 978-0-07-288138-7

Mark A. Richards and James A. Scheer
Principles of Modern Radar: Basic Principles
ISBN 978-1-89-112152-4

Deepak Kumar Tala
World of ASIC
http://www.asic-world.com/

Jean P. Nicolle
FPGA 4 Fun
http://www.fpga4fun.com/

FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Dr Stephen Paine

Day 5 – 3 May 2022