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 4 – 12 September 2024

Outline

Injection Testing

Advanced Simulation

Mutual Exclusion and Arbitration

Caching Systems

Numerically-controlled Oscillator

Pulse-width Modulation

Outline

Injection Testing

Advanced Simulation

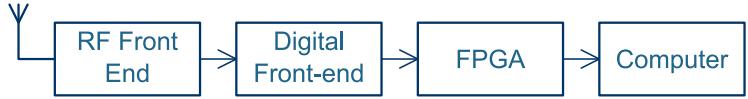
Mutual Exclusion and Arbitration

Caching Systems

Numerically-controlled Oscillator

Pulse-width Modulation

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

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...

Outline

Injection Testing

Advanced Simulation

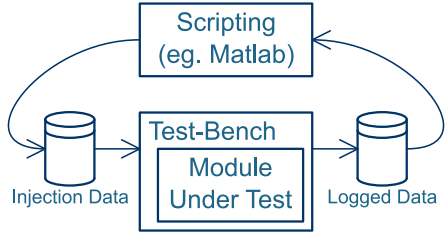
Mutual Exclusion and Arbitration

Caching Systems

Numerically-controlled Oscillator

Pulse-width Modulation

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

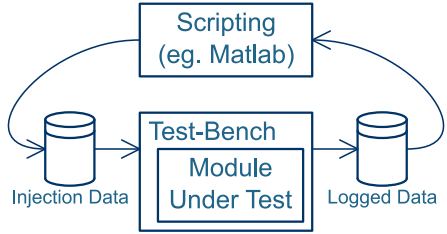
```
reg [7:0]Byte; reg [15:0]Data; integer Result;

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

Data Injection (ADXL345_TB)

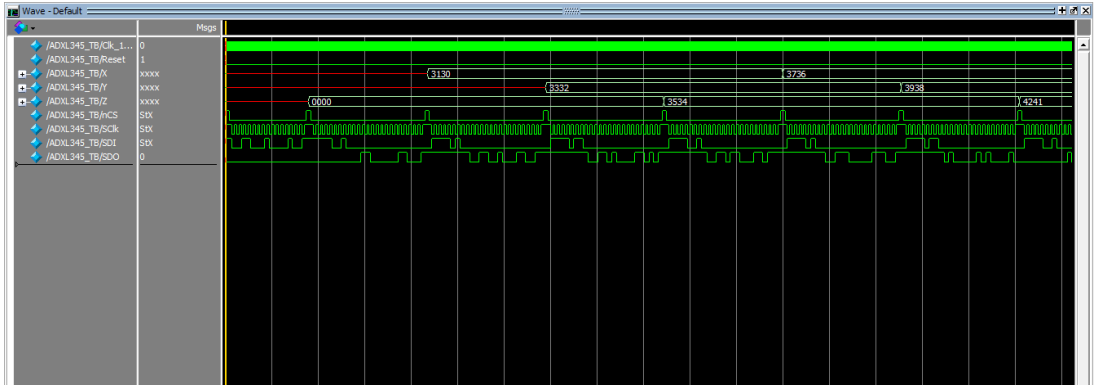


- Create a test file “ADXL345_TB.dat”:
  `0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ`

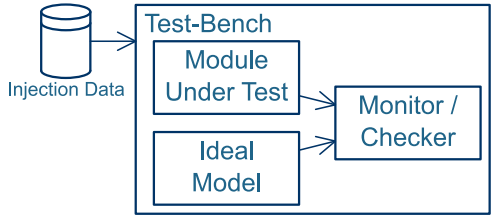
Data Injection (ADXL345_TB)

```
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

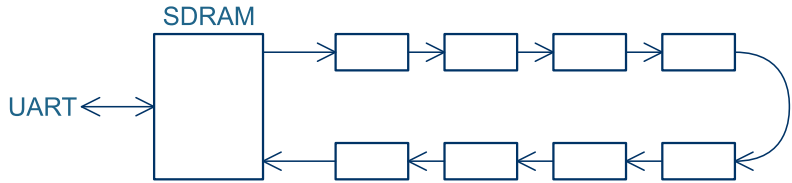


- Very useful, but time-consuming to write...
- The “Ideal Model” can be read from a file, or calculated in the test-bench
- The “Monitor” issues errors (with `$error`), warnings (with `$warning`) or information (with `$info`) as appropriate
- Refer to the Universal Verification Methodology for more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

Coffee Break...

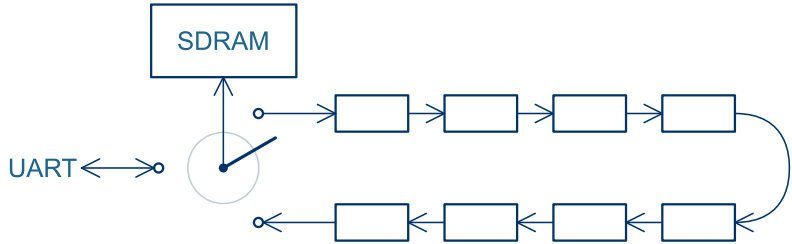
Outline

Sharing Resources



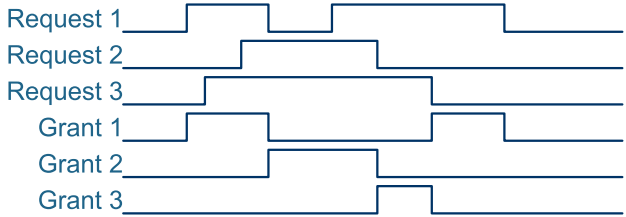
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^2C bus
  - etc.

Mutual Exclusion



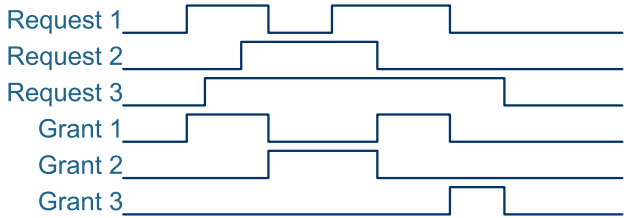
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Round-robin Mutual Exclusion



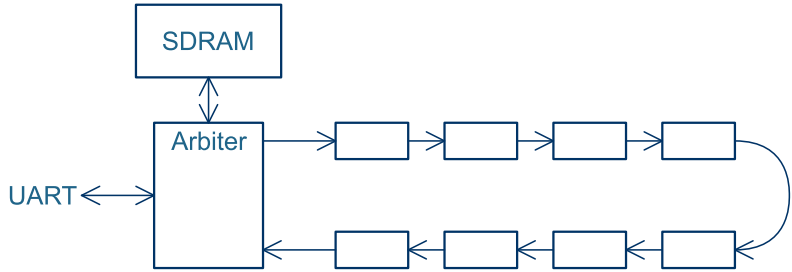
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module's request line)

Priority-based Mutual Exclusion

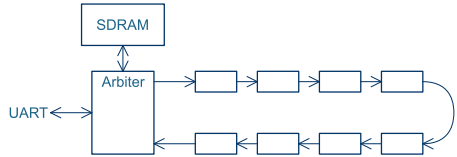


- When more than one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (same-cycle response by means of a combinational circuit)
- Scales well

Arbitration

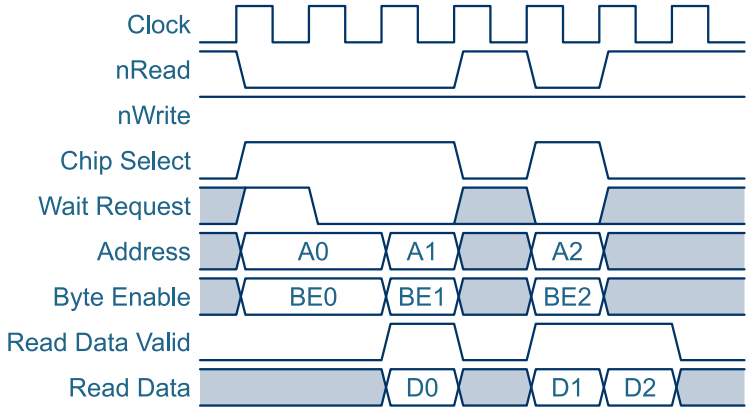


Arbitration



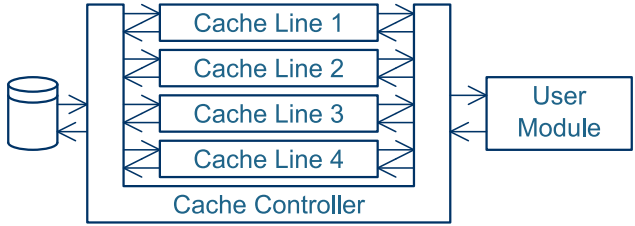
- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^2C standard implements arbitration by collision-detection 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

General Cache

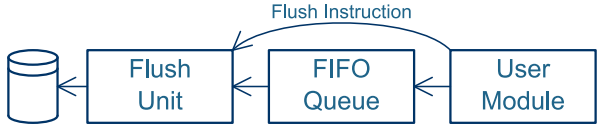


- Typical cache line sizes are 32, 64, 128 or 256 bytes
- Controller does address translation between external and cache line addresses
- Read on-demand, flush the least recently used line and load the desired content
- Controller blocks the user module when a cache miss occurs

Pre-fetch Read Cache

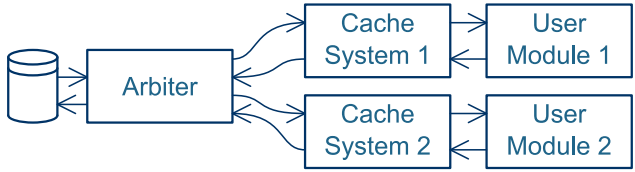
- Continuous reading: the user module is only blocked when the queue is empty
- The queue is typically multiple pages in size
- New data is fetched as soon as there is one page open on the queue

Write Cache



- Writing is continuous, and the user module is only blocked when the queue is full
- Alternatively, the user module is never blocked (data dropped on overflow)
- The flush unit automatically flushes the queue when there is one page of data
- The user module can also manually flush the queue if required

Cache Coherence



- Two independent cache system can refer to the same place in memory
- When the one user module reads what the other is writing, the two cache system must remain up to date with each other
- When the two user modules write to different places in the same page, a flush must not overwrite the other's data
- Many schemes exist to mitigate these and other issues, which are outside the scope of this course

Outline

Injection Testing

Advanced Simulation

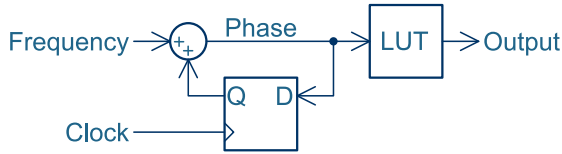
Mutual Exclusion and Arbitration

Caching Systems

Numerically-controlled Oscillator

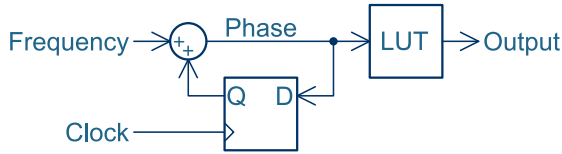
Pulse-width Modulation

Numerically-controlled Oscillator



- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

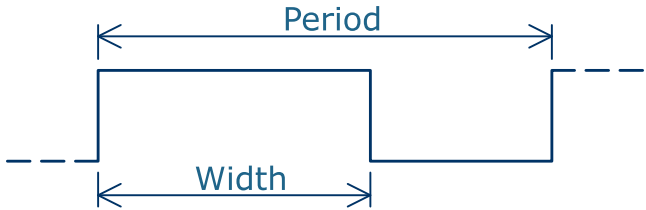
Numerically-controlled Oscillator



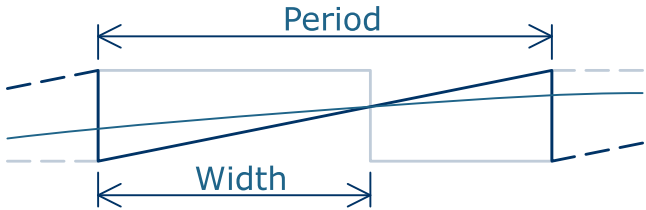
$$f_{out} = f_s \cdot \frac{f_{in}}{2^N}, \quad N = 32, \quad f_s = 100\ \text{MHz}$$

Outline

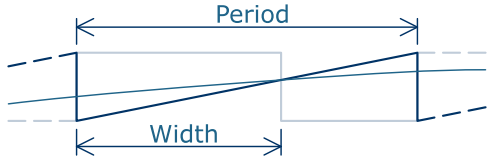
Pulse-width Modulation



Pulse-width Modulation

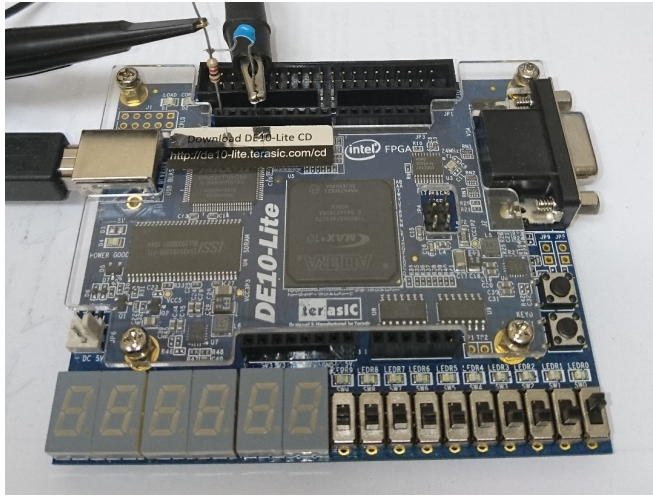


Pulse-width Modulation



- Typically, the signal is in 2's complement
- The PWM module requires an unsigned input from 0 to $(2^N - 1)$ – also known as offset-binary
- To convert from one to the other, invert the most-significant bit

Pulse-width Modulation



High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

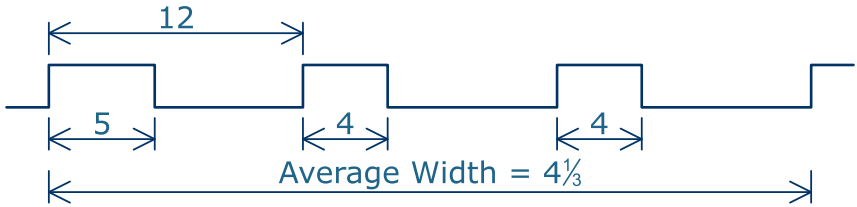
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

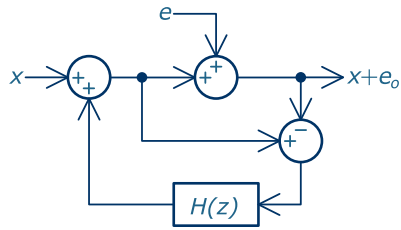
High Resolution PWM

- Assume that your signal is audio (20 kHz bandwidth)
- You want the PWM frequency at least 10 times the signal bandwidth $\Rightarrow$ 200 kHz
- Say you want 16-bit output resolution...
- You have to run the saw-tooth counter at 200 kHz $\times$ $2^{16} = 13.1072$ GHz
- That is a FAST clock!
- The solution...

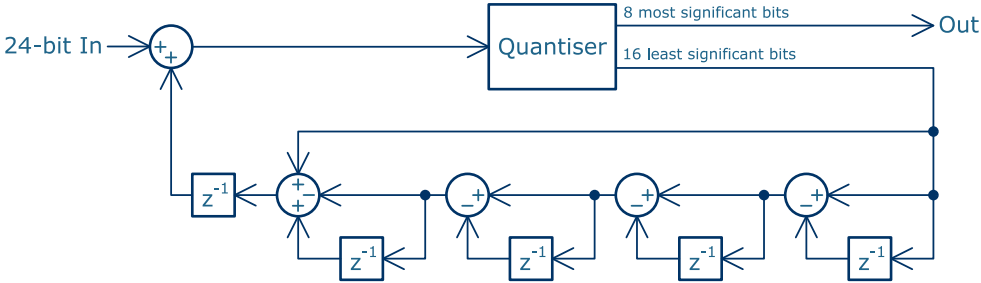
Noise Shaping



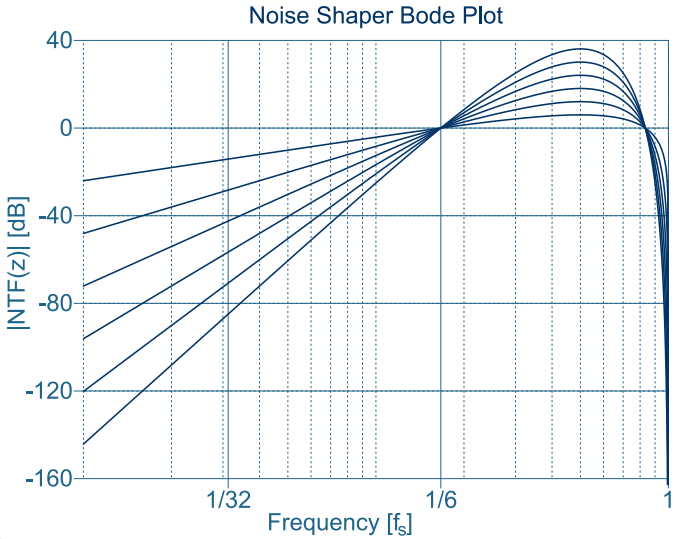
Noise Shaping



Noise Shaping



Noise Shaping



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 4 – 12 September 2024