Cover Page

CPE 324-02: Intro to Embedded Computer System

**Lab 6**

**FIFO Memory**

**Submitted by**: Nolan Anderson

**Date of Experiment**: 04/09/2021

**Report Deadline**: 04/13/2021

**1. Introduction:**

**1.1 - What is to be studied, what is the purpose, and how is this purpose accomplished?**

|  |
| --- |
| The purpose of Lab 6 is to expand our knowledge of FIFO memory by using newly developed code and code from previous labs. To do this, we first need to study FIFO memory, Fibonacci sequence, and a simple up counter (see section 2). We will accomplish this by writing code in varying difficulty, with phase 2 being the most in depth. We will also display the output of all the code (section 4) and go over phase 3 with a short demonstration video. Section 6 will contain all the new code developed in the lab. |

**2. Experiment Description**

**2.1 Theory, analysis, and purpose:**

**2.1.1 FIFO Memory**

|  |
| --- |
| FIFO (first in first out) memory chips are good for applications where the devices operate at different speeds. You can temporarily store values for future processing so that you cover all your data. In this lab, we build our FIFO memory by using Memory Building blocks:    This is simply the block ram portion, however, and more implementation needs to be created to store and extract this data. We do this by creating FIFO Micro-Architecture. This is implemented in more detail in the Appendix, but a general overview can be seen below. |

**2.1.2 Fibonacci Sequence**

|  |
| --- |
| The Fibonacci sequence is what we are storing in our FIFO ram. The general formula for this sequence is as follows: Fn = Fn-1 + Fn-2. As you can see this data will grow very quickly. Continually, we can see this implemented in the appendix. |

**2.1.3 Up Counters**

|  |
| --- |
| A general up counter will be used to increment through our Fibonacci sequence in phase 3. What is different about this up counter, however, is the fact that it will reset once the value gets to be too large for the DE10 lite to output, since we only have a 6-segment display. Even with displaying in hex, these values become very large. |

**2.1.4 Top Level Design**

|  |
| --- |
| The top-level design for this program can also be found in the appendix. This file allows for the DE10 Lite board to display all the first 36 values of the Fibonacci sequence. A general overview can be seen below.  As you can see, this top-level file will introduce and use much of the code that we have already developed throughout this lab including the display\_driver and meta from labs 3 and 4, and also the generated syspll file. |

**2.2 Design and implementation procedure:**

|  |
| --- |
| All the design models can be found in the section 2.1, and the implementation procedure can be found in the appendix. |

**3. Demonstration**

**3.1 Implementation method:**

|  |
| --- |
| **See the appendix for the code I used.** |

**3.2 Video Link:**

|  |
| --- |
| **Video is uploaded to provided link.** |

**4. Experimental Results**

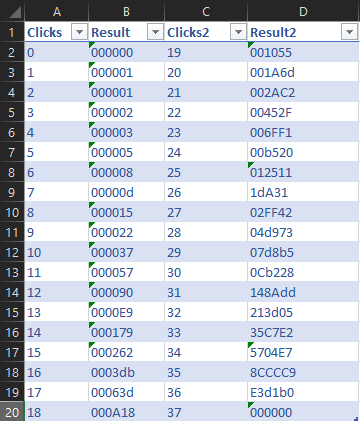
**4.1 Observations:**

|  |
| --- |
| The results for the 3 phases can be seen in section 4.3. Overall, the results are as expected when comparing to theoretical values and operations. |

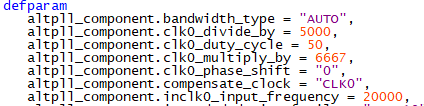
**4.2 Post lab questions:**

1. Run through the entire 36 entries of the Fibonacci sequence stored in the ROM and report the numbers on the lab report. What is the next value of the sequence? Will it fit in a 24-bit binary number space of the 6-digit hex display?

A graph of the outputs can be seen below. Yes, it can fit on the display but if you were to continue past 36 iterations, it would eventually not fit on the display.



* 1. 2. For generating the 66.667 MHz clock for the read-side of the FIFO:
  2. a. What frequency does the VCO oscillate at?
  3. 66.67 Mhz
  4. b. What is the clock division ratio on the 50 MHz input clock?
  5. 5000



* 1. c. What is the divider for the VCO feedback path?

20000

* 1. d. The answers to 1b and 1c above must lead to the same frequency for the PLL to lock – do they? What frequency is it?

Yes. They both go to 66.67 Mhz

e. What is the division ratio from the VCO to the 66.667 MHz output clock?

1

* 1. 3. The top-level module has two meta.v blocks drawn at the ~locked output of the system PLL. They are connected to the same inSig, so why are there two different ones? Where do the outputs of these blocks lead?
  2. There are two in the top level because one is connected to the clock and the other is not. The One not connected to the clock is used for the reset of the up counter and the reset for the write reset. The one with the clock input is sent to the read reset, essentially if it times out then we reset the reading.
  3. 4. What is the reported device utilization with the original sdp\_ram\_infer.v block? This must include the number of logic elements, flip-flops, and dedicated RAM blocks and/or bits.

11,232 / 1677312 (Memory Bits) 1838 / 49760 (Total Logic Elements)

* 1. a. In Phase 3, after recording the device utilization numbers, the student is asked to modify the sdp\_ram\_infer.v file to include a reset to set the RAM contents to 0 for every address. What effect does this have on device utilization numbers? Does it have an adverse impact on meeting timing?

The device utilization numbers were a bit less when implementing it this way.

* 1. b. The other modification to sdp\_ram\_infer (besides resetting all contents) is to remove the 1-cycle latency between the RAM data array storage and the read-data output. Again in this scenario, report and remark on the ability for Quartus to map the RAM into the large, efficient Block RAMs, and whether device utilization and timing were negatively impacted.
  2. Timing seems to be affected, but the device utilization is less. This is because it is not using as much of the device to implement the latency.
  3. 5. Try the following depths for the async\_fifo block, and report on device utilization numbers. For the case where the depth of the FIFO is not a power-of-two, simulate it to ensure it works properly and then demonstrate that it performs the same as any other (power-of-two) option.
  4. Output generated was the same
  5. a. 32 - 11136 / 1677312
  6. b. 128 - 13440 / 1677312
  7. c. 10 - 10608 / 1677312
  8. d. 512 - 22656 / 1677312
  9. e. 2048 – 59520 / 1677312

**4.3 Results**

**4.3.1 Phase 1 Output**

|  |
| --- |
|  |

**4.3.2 Phase 2 Output**

|  |
| --- |
|  |

**4.3.3 Phase 3 Output**

|  |
| --- |
| The output for phase 3 can be seen in the demonstration video. The output for the hex display will be the first 36 numbers of the Fibonacci sequence. |

**5. Conclusions**

**5.1 - Results and lessons learned:**

|  |
| --- |
| Overall, this lab was extremely difficult. Not due to the actual coding. The overall coding was not that bad, but the bug fixing, and wire checking took far too long to be acceptable for a 1 hour course. I did learn a lot, though. |

**6. Appendix**

**6.1 – Phase 1 Code:**

|  |
| --- |
| // Nolan Anderson  // CPE324  // Lab 6 Phase 1 Fibonacci ROM  module fibonacci\_rom (input  [5:0]  rdadr, output [23:0] rddat);    reg [23:0] rom [0:35];    integer i;    initial begin    rom[0] = 1; rom[1] = 1;    for(i = 2; i < 35; i = i + 1) begin        rom[i] = rom[i - 1] + rom[i - 2];    end    end      assign rddat = rom[rdadr];  endmodule  **---------------------------------------------------------------------------------------------**  // Nolan Anderson  // CPE324  // Lab 6 Phase 1 UP Counter  module up\_counter #(parameter WIDTH, parameter TERM\_CNT)  ( input clk, reset, en, output reg [WIDTH-1:0] count);  always @(posedge clk)begin  if(reset)begin count = 0; end  else if(en) begin      if(count == 2\*\*(WIDTH)-1)begin count = 0; end      else if (count == TERM\_CNT)begin count = 0; end      else begin count = count + 1; end      end  end  endmodule |

**6.2 – Phase 2 Code:**

|  |
| --- |
| // Nolan Anderson  // CPE324  // Lab 6 Phase 2 ASYNC Fifo  module async\_fifo #(parameter WIDTH, parameter DEPTH)  (    input              wrClk,    input              wrRst,    input              wrEn,    input  [WIDTH-1:0] wrData,    input              rdClk,    input              rdRst,    input              rdEn,    output             full,    output             empty,    output [WIDTH-1:0] rdData  );    localparam DEPTH\_LOG = $clog2(DEPTH);    localparam PTR\_WIDTH = (DEPTH == 2\*\*DEPTH\_LOG) ? DEPTH\_LOG + 1 : DEPTH\_LOG;    wire w0, w1;    and a0(w0, wrEn, ~full);    wire [PTR\_WIDTH - 1 : 0] WritePTR;    and a1(w1, rdEn, ~empty);    wire [PTR\_WIDTH - 1 : 0] ReadPTR; // Read Address    wire [DEPTH\_LOG - 1 : 0] w2;      // GC Out Write    wire [DEPTH\_LOG - 1 : 0] w3;      // FF Out Write    wire [DEPTH\_LOG - 1 : 0] w4;      // META Out Write    wire [DEPTH\_LOG - 1 : 0] w5;      // GD OutWrite    wire [DEPTH\_LOG - 1 : 0] w6;      // GC Out Read    wire [DEPTH\_LOG - 1 : 0] w7;      // FF Out Write Read    wire [DEPTH\_LOG - 1 : 0] w8;      // Meta Out Read    wire [DEPTH\_LOG - 1 : 0] w9;      // GD Out Read    up\_counter #(.WIDTH(PTR\_WIDTH), .TERM\_CNT(2\*\*(PTR\_WIDTH)-1))    Write\_Ptr(      .clk(wrClk),      .reset(wrRst),      .en(w0),      .count(WritePTR)    );    up\_counter #(.WIDTH(PTR\_WIDTH), .TERM\_CNT(2\*\*(PTR\_WIDTH)-1))    Read\_Ptr(      .clk(rdClk),      .reset(rdRst),      .en(w1),      .count(ReadPTR)    );    sdp\_ram\_infer #(.WIDTH(WIDTH), .DEPTH(DEPTH), .DEPTH\_LOG(DEPTH\_LOG))    RAM(      .wrClk(wrClk),      .rdClk(rdClk),      .rdRst(rdRst),      .wrEn(w0),      .wrDat(wrData),      .wrAdr(WritePTR),      .rdEn(w1),      .rdAdr(ReadPTR),      .rdDat(rdData)    );    gray\_coder #(.WIDTH(DEPTH\_LOG))    write\_coder(      .binIn(WritePTR),      .gcOut(w2)    );    DFF #(.WIDTH(DEPTH\_LOG))    WprtGC(      .clk(wrClk),      .D(w2),      .Q(w3)    );    meta #(.DATA\_WIDTH(DEPTH\_LOG), .DEPTH(2))    WptrGCX(      .clk(rdClk),      .in\_sig(w3),      .out\_sig(w4)    );    gray\_decoder #(.WIDTH(DEPTH\_LOG))    write\_decoder(      .gcIn(w4),      .binOut(w5)    );    gray\_coder #(.WIDTH(DEPTH\_LOG))    read\_coder(      .binIn(ReadPTR),      .gcOut(w6)    );    DFF #(.WIDTH(DEPTH\_LOG))    RptrGC(      .clk(rdClk),      .D(w6),      .Q(w7)    );    meta #(.DATA\_WIDTH(DEPTH\_LOG), .DEPTH(2))    RptrGCX(      .clk(wrClk),      .in\_sig(w7),      .out\_sig(w8)    );    gray\_decoder #(.WIDTH(DEPTH\_LOG))    read\_decoder(      .gcIn(w8),      .binOut(w9)    );      assign full = ((DEPTH) == (WritePTR - w9)) ? 1 : 0;    assign empty = (ReadPTR == w5) ? 1 : 0;  endmodule |

**6.3 – Phase 3 Code:**

|  |
| --- |
| // Nolan Anderson  // CPE324  // Lab 6 Phase 3 Top Level File  `define ENABLE\_CLOCK1  `define ENABLE\_HEX0  `define ENABLE\_HEX1  `define ENABLE\_HEX2  `define ENABLE\_HEX3  `define ENABLE\_HEX4  `define ENABLE\_HEX5  `define ENABLE\_KEY  `define ENABLE\_LED  `define ENABLE\_SW  `define ENABLE\_GPIO  module DE10\_LITE\_Golden\_Top(      //////////// ADC CLOCK: 3.3-V LVTTL //////////  `ifdef ENABLE\_ADC\_CLOCK      input                       ADC\_CLK\_10,  `endif      //////////// CLOCK 1: 3.3-V LVTTL //////////  `ifdef ENABLE\_CLOCK1      input                       MAX10\_CLK1\_50,  `endif      //////////// CLOCK 2: 3.3-V LVTTL //////////  `ifdef ENABLE\_CLOCK2      input                       MAX10\_CLK2\_50,  `endif      //////////// SDRAM: 3.3-V LVTTL //////////  `ifdef ENABLE\_SDRAM      output          [12:0]      DRAM\_ADDR,      output           [1:0]      DRAM\_BA,      output                      DRAM\_CAS\_N,      output                      DRAM\_CKE,      output                      DRAM\_CLK,      output                      DRAM\_CS\_N,      inout           [15:0]      DRAM\_DQ,      output                      DRAM\_LDQM,      output                      DRAM\_RAS\_N,      output                      DRAM\_UDQM,      output                      DRAM\_WE\_N,  `endif      //////////// SEG7: 3.3-V LVTTL //////////  `ifdef ENABLE\_HEX0      output           [7:0]      HEX0,  `endif  `ifdef ENABLE\_HEX1      output           [7:0]      HEX1,  `endif  `ifdef ENABLE\_HEX2      output           [7:0]      HEX2,  `endif  `ifdef ENABLE\_HEX3      output           [7:0]      HEX3,  `endif  `ifdef ENABLE\_HEX4      output           [7:0]      HEX4,  `endif  `ifdef ENABLE\_HEX5      output           [7:0]      HEX5,  `endif      //////////// KEY: 3.3 V SCHMITT TRIGGER //////////  `ifdef ENABLE\_KEY      input            [1:0]      KEY,  `endif      //////////// LED: 3.3-V LVTTL //////////  `ifdef ENABLE\_LED      output           [9:0]      LEDR,  `endif      //////////// SW: 3.3-V LVTTL //////////  `ifdef ENABLE\_SW      input            [9:0]      SW,  `endif      //////////// VGA: 3.3-V LVTTL //////////  `ifdef ENABLE\_VGA      output           [3:0]      VGA\_B,      output           [3:0]      VGA\_G,      output                      VGA\_HS,      output           [3:0]      VGA\_R,      output                      VGA\_VS,  `endif      //////////// Accelerometer: 3.3-V LVTTL //////////  `ifdef ENABLE\_ACCELEROMETER      output                      GSENSOR\_CS\_N,      input            [2:1]      GSENSOR\_INT,      output                      GSENSOR\_SCLK,      inout                       GSENSOR\_SDI,      inout                       GSENSOR\_SDO,  `endif      //////////// Arduino: 3.3-V LVTTL //////////  `ifdef ENABLE\_ARDUINO      inout           [15:0]      ARDUINO\_IO,      inout                       ARDUINO\_RESET\_N,  `endif      //////////// GPIO, GPIO connect to GPIO Default: 3.3-V LVTTL //////////  `ifdef ENABLE\_GPIO      inout           [35:0]      GPIO  `endif  );  //=======================================================  //  REG/WIRE declarations  //=======================================================  wire notButton0, clk67, a1, w0, w1, w2, w3, w6, w8, lock, buttonPress;  assign notButton0 = ~KEY[0];  localparam WIDTH = 24;  localparam DEPTH = 36;  wire [WIDTH-1:0] w4;  wire [WIDTH-1:0] w5;  wire [WIDTH-1:0] w7;  assign LEDR[0] = lock;  //=======================================================  //  Structural coding  //=======================================================  meta #(.DATA\_WIDTH (1), .DEPTH (2))  meta\_1(    .clk (clk67),    .in\_sig (KEY[1]),    .out\_sig (w0)  );  D\_Flip\_Flop #(  .WIDTH(WIDTH))  RptrGC(      .clk(clk67),      .D(w0),      .Q(w1)  );  and a0(a1,~w0,w1);  syspll pll66 (     .areset(notButton0),      .inclk0 (MAX10\_CLK1\_50),      .c0     (clk67),      .locked (lock)   );    meta #(.DATA\_WIDTH (1),.DEPTH (2))  meta\_sys1(    .clk (clk67),    .in\_sig (~lock),    .out\_sig (w2)  );  meta #(.DATA\_WIDTH (1), .DEPTH (2))  meta\_sys2(    .clk (clk67),    .in\_sig (~lock),    .out\_sig (w3)  );    up\_counter #(.WIDTH(WIDTH), .TERM\_CNT(DEPTH))  Write\_Ptr(      .clk(MAX10\_CLK1\_50),      .reset(w2),      .en(~w6),      .count(w4)  );  fibonacci\_rom fib\_rom(      .rdadr(w4),      .rddat(w5)  );  async\_fifo #(.WIDTH(WIDTH),.DEPTH(DEPTH))  a\_fifo(    .wrClk(MAX10\_CLK1\_50),    .wrRst(w2),    .wrEn(~w6),    .wrData(w5),    .rdClk(clk67),    .rdRst(w3),    .rdEn(a1),    .full(w6),    .empty(LEDR[1]),    .rdData(w7)  );  display\_driver hex\_leds (    .clk        (clk67),    .dispMode   (1'b1),    .oneMsPulse (1'b1),    .OpReg      (1'b0),    .ShowOpReg  (1'b0),    .OpCode     (1'b0),    .ShowOpCode (1'b0),    .OpResult   (w7),    .HEX0       (HEX0),    .HEX1       (HEX1),    .HEX2       (HEX2),    .HEX3       (HEX3),    .HEX4       (HEX4),    .HEX5       (HEX5)  );  endmodule |