

**Purpose**

In this lab, we will be reviewing the principles of system-level design by designing a digital system that performs factorial computation. We will go through each step in the design process: designing, functionally verifying, and FPGA prototyping the digital system. Each step is further broken up into smaller tasks to make the design and verification process simpler. The assignment requires that a factorial be calculated with input n. Input n must not be greater than 12. The digital system must also output a “Done” or “Error” signal depending on the status of the calculation. For example, if the input n is greater than 12, the “Error” signal must be output. An input “Go” signal must start the computation. Therefore, our design will incorporate a state machine that drives the control signals for a datapath where calculations of the factorial machine are performed. Finally, the design must be verified and tested.

**Design Methodology**

Based on the given algorithm in Assignment 1, a datapath was designed using the given building blocks. This design can be seen in Figure R. The building blocks are described in Table B and drawn in Figures A-F. The design takes into account several constraints, such as input n must be less than or equal to 12. In the datapath design, an input is fed into a down counter and a comparator. The comparator checks to ensure that input n is less than or equal to 12. The comparator generates an output error signal that is sent to the control unit that will later determine the next state transition in the control unit. Assuming the input n is less than or equal to 12, the input is stored in the down counter successfully. The down counter may begin to count down when enabled by its enable signal input. The output of the down counter is then fed into a multiplier module. This module multiplies the output of the down counter with the input from a register module. The register module is initially loaded with a 1. After the first multiplication has occurred, the data is written back into the register. A multiplexor at the input of the register controls whether a 1 or the result of the calculation is loaded into the register. At the same time, the down counter output feeds into another comparator that ensures the output is greater than or equal to 1. If the down counter outputs a 0, the result of the next multiplication will result in a 0, thereby nullifying the entire process. The comparator produces a signal that influences the state transitions in the control unit, similar to the error signal. Finally, the output of the register shows the final computation result of the machine. The output of the register is controlled by another multiplexor which selects the register’s output when necessary. The design is visible in Figure R. Below are diagrams describing all building blocks used.

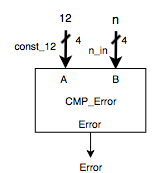


Figure A. A comparator with two 4 bit inputs: CMP\_Error

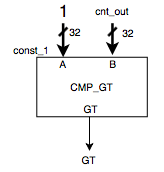


Figure B. A comparator with two 32 bit inputs: CMP\_GT

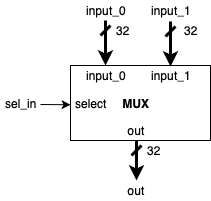


Figure C. A multiplexor with two 32 bit inputs: MUX. Since MUX1 is identical to MUX2, only one MUX is shown here.

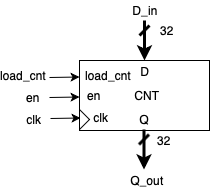


Figure D. A down counter module with one 32 bit input: CNT

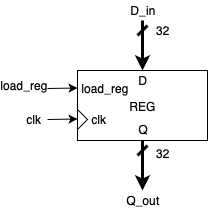


Figure E. A register module with one 32 bit inputs: REG

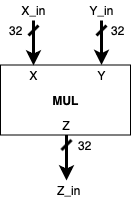


Figure F. A multiplication module with two 32 bit inputs: MUL

The two-piece CU-DP system block diagram is shown in Figure Q. This task was completed with the knowledge of the datapath design from the previous task. Essentially, all control signals needed to drive the datapath were signals that would be given to the datapath from the control unit. These include the signals to load the register and counter, select signals for the multiplexores, and the enable signal for the counter. Status signals from operations in the datapath are given to the control unit to influence the state machine transitions. Other signals needed by the assignment requirements are also input or output by the control unit. The following is a list of all signals input into the control unit: Go (Go), clock (clk), datapath error (DP\_error), and a greater-than signal (GT). The control unit outputs the following: a control unit error signal (CU\_error) and a Done signal to indicate computation is complete. The datapath receives the following signals as inputs: clock (clk), input n (n), load counter (Load\_cnt), enable (en), load register (Load\_reg), mux select 1 (sel1), and mux select 2 (sel2). The datapath outputs the result (product) of the computation. A full description showing where each signal comes from and the bus width of each signal is shown in Figure Q.

The ASM chart design is based on the datapath design and the assignment requirements. There are a total of five states, S0, S1, S2, S3, S4, and S5. S0 is the IDLE state: it waits for a Go signal before transitioning to the next state, S1. S1 is a loading state wherein the register is loaded with a 1 for the initial multiplication calculation. S1 also allows for the down counter to be loaded with initial value of n. S1 then transitions to S2 or S5 depending on if the input n is less than or equal to 12. This check is performed in the datapath with the CMP\_Error module which compares two input values. In S5, an error signal is sent from the datapath to the control unit. The control unit then outputs the error signal to indicate the error. S2 is a wait state. This state acts as a way to ensure the register and down counter are loaded correctly and a state to transition to when performing the multiplication calculation in S3. S2 transitions to either S3 or S4 depending on a greater-than (GT) signal from the datapath. This check is performed by another comparator in the datapath. If the output of the down counter is greater than or equal to 1, then S2 will transition to S3. In S3, the actual calculation is performed: the value stored in the register multiplied with the value output from the down counter. Additionally, the down counter is decremented in this state. S3 then transitions unconditionally so S2. S2 will transition to S4 when the output of the down counter is less than 1. In S4, the system will output the final result of the calculation (product) and a Done signal to indicate that the calculation was completed successfully. S4 then transitions unconditionally to S0. The state transition diagram is shown in Figure T. This state transition diagram is an example of a Moore machine.

The bubble diagram is based solely on the state transition diagram. Using knowledge from CMPE 124 and CMPE 125, we were able to draw the bubble diagram successfully as seen in Figure U.

The output table we constructed is visible in Figure V. It shows the control signals output by the control unit state machine necessary to drive the datapath. The output table was constructed by following along with the state transition diagram constructed in Figure T and the datapath design in Figure R. For example, during S1, the initial loading state, we looked at what control signals were necessary to load the register and down counter with the initial values. These values were then input into the output table for S1. This method was performed for all states in the state transition diagram to build the output table. Again, the signals necessary to drive the datapath or output by the factorial machine are visible in Figure V.

Verilog code was written following the basic principles of Verilog coding found from online tutorials and learned in CMPE 125. Starting with the smallest modules in the microarchitecture design of the datapath in Figure R, we wrote code for each basic building block. The code follows how the module should behave. For example, the register loads data when the load\_reg input is high and on the rising edge of an input clock. This process was followed for all building blocks.

Next, modules were coded to contain and instantiate the smaller building blocks. For example, for the datapath module, all smaller building blocks were instantiated and initialized. Then, all instances of the modules were connected to each other with virtual wires as described in the datapath design in Figure R. This process was followed for the control unit but with the state transition diagram in Figure T.

Finally, a larger module were designed to contain the control unit and datapath. The control unit and datapath were wired with their respective input and output signals as described in the CU-DP diagram in Figure Q. This larger module comprises the main system design of the Factorial machine.

For the testbenches, a testbench was written for the CU, DP, and Factorial modules. The CU and DP testbenches were designed to be as basic as possible to verify that both worked properly. For the CU testbench, we tested to ensure the state machine transitioned to each state properly and output the correct control signals against the output table in Figure V. Inputs to the state machine were stimulated with hard-coded values. For example, the inputs GT, DP\_error, clk, and go were all given a value in each state. These input signals drive the state transitions. Figure J shows the waveforms verifying that the correct control unit signals were output in each state. The code for this testbench is visible in Table T.

Likewise, a testbench was written to verify the proper function of the datapath. In this case, values for inputs to the datapath were hard-coded with n having an initial value of 4. We wrote the testbench to work as if a control unit was driving the control signals for the counter, register, and multiplexors to verify that the correct result of the calculation was produced. The waveform showing verification of this testbench and datapath design is visible in Figure I. The code for this testbench is visible in Table U.

Finally, an exhaustive, self-checking testbench was written to test both the control unit and datapath at the same time. This testbench loops from 14 to 0, giving input n a different value in each iteration of the loop. The loop also properly generates the correct number of clock cycles for each input n. Next, the loop also calculates n! itself based on the input n for that iteration. Finally, the loop compares the calculated result of the Factorial module with the actual result of n! for that iteration and displays a message to the console indicating whether the calculation was correct or wrong. The output of the console is visible in Figure W. Careful considerations were made to ensure the correct number of clock cycles occurs in the loop and to compare the two values before the state machine transitions back to S0 (where the result would reset to 0) in each iteration. Code for this testbench is visible in Table S. Waveforms for this testbench are visible in Figure H.

The Factorial system design was implemented on the Basys3 FPGA board successfully. This task was completed with the creation of a few new modules and a Factorial\_FPGA module that integrates everything together. The new modules were copied from the Appendix given in Assignment 1. These include the hex2led, clk\_gen, bin2hex32, HILO\_MUX, and LED\_MUX modules. These modules help take the product calculated by the Factorial design and display it on the Basys3’s seven-segment displays.

The Factorial\_FGPA module instantiates the Factorial module with the control unit and datapath modules with the necessary inputs and outputs and binds them to the hardware resources on the board. For example, the input n for the Factorial module is tied to the four right most switches on the board so that n can be input in binary. This also includes assigned LEDs for output signals like Done and Error and a debounced button to control the clock for the control unit’s state machine. Other board resources are described in Figure K. The Factorial\_FPGA also instantiates all modules necessary to display the product on the seven-segment display. These modules help convert the result to a hexadecimal and properly choose which LEDs to light up on the display to properly show the hexadecimal result. A HILO\_sel signal helps to show the most significant 16 bits or the least significant 16 bits of the result. The design for this module is visible in Figure S. The code for this module is visible in Table C.

**Waveforms**

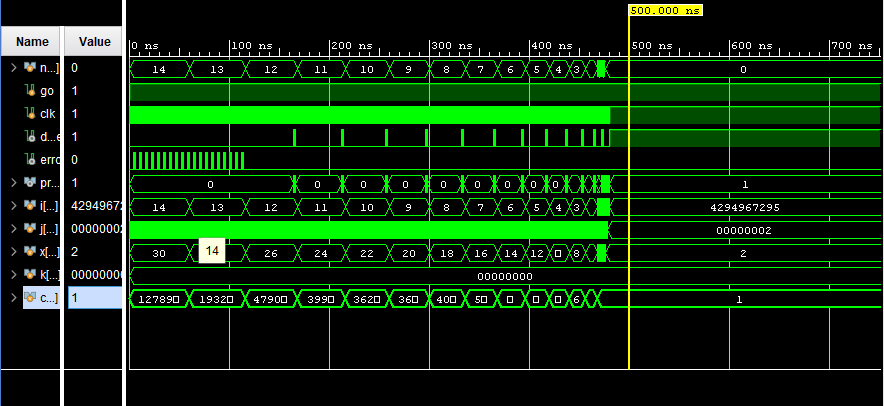


Figure G. An overview of the testbench waveforms for the Factorial module depicts input values 0-14. Exhaustive testing is depicted. For each value, the expected factorial is depicted in the bottom-most waveform. Meanwhile, the error signal flags as according to whether the input value is larger than 12. The Go signal remains on throughout the duration of the factorial computation. For values 0-12, the done signal is on for the completion of each factorial computation.

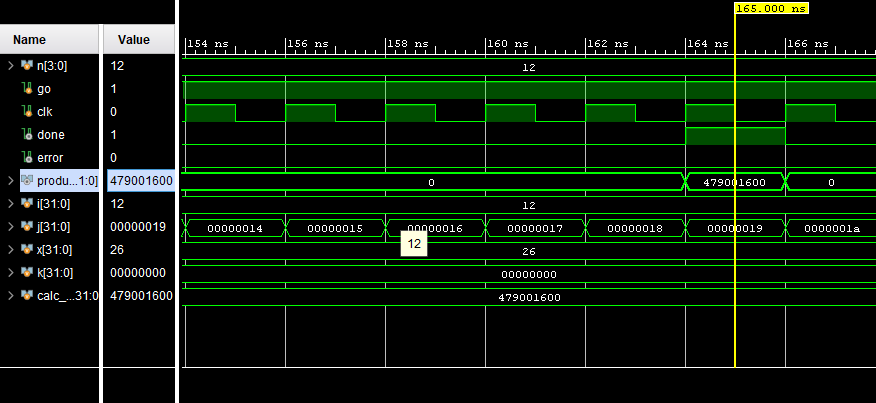
****

Figure H. A zoomed in view of factorial testbench waveforms for the input of 12 displays the expected result. The Go signal remains on throughout the duration of the computation. The clock cycle allows the transitions of states. Meanwhile, the error flag remains low, since the input value does not exceed 12. The factorial computation of 479001600 is captured in the 6th clock cycle. The value of I provided by the testbench represents the number of integers being multiplied. The value of J represents the multiplied results to reach the factorial result. The value of X represents the number of clock cycles required to compute the factorial. The value of k represents the number of iterations required for a factorial computation. The factorial testbench depicts the waveforms as expected for the input value of 12.

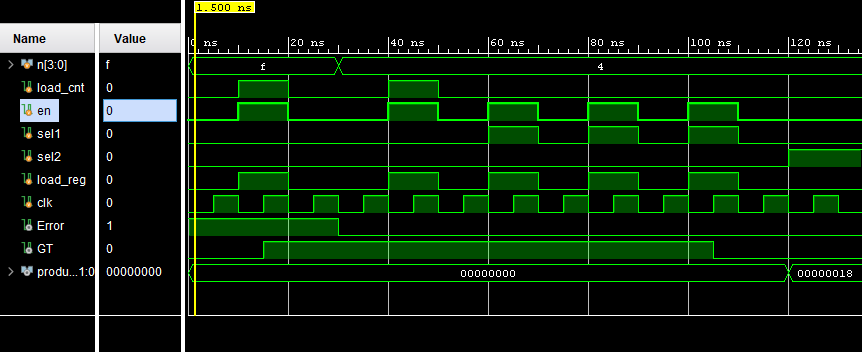


Figure I. The testbench waveforms for the datapath module are displayed. For a test input of 4, the expected output signals provided to the datapath from the control unit are displayed for each state as depicted in Figure Q. The expected values were validated with the output table of the CU in Figure V. Meanwhile, the datapath outputs an error signal which is high for an input value larger than 12 for input f. The datapath also outputs GT, which remains high for values from the counter which are larger than one. The expected factorial result of the datapath is displayed by simulating the inputs provided by the control unit.

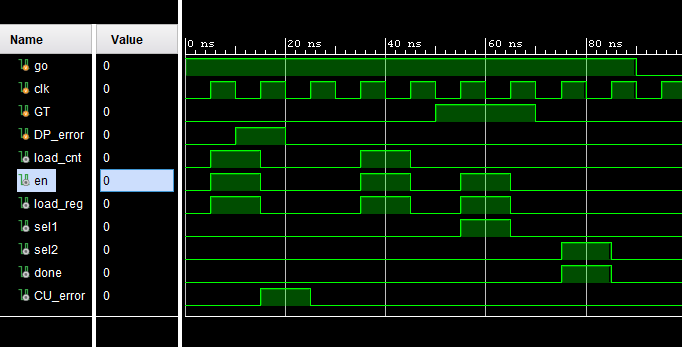


Figure J. The testbench waveforms for the control unit are displayed. The Go signal is depicted as high as the clock signal cycles through the states. By providing the GT and DP\_error signals from the datapath, the states reached were verified as according to the paths depicted in Figure U. The resultant output signals of the control unit were accurate according to the states shown in the output table of Figure V.

**Hardware Validation**

Exhaustive testing occurred successfully to validate and display the computed factorial value in hexadecimal for each value from 0-12. After the Go switch was assigned as a high value, random testing occurred by selecting an n value for the rightmost switches. The clock signal was assigned a button to cycle through the states. Meanwhile, hardware validation of the Go and Done signals occurred by assigning them LEDs as according to Table A.

Table A. Hardware Configuration

|  |  |
| --- | --- |
| **Variable** | **Pin** |
| Go | R2 |
| Done | U16 |
| N[0-3] | V17, V16, W16, W17 |
| HILO\_sel | T1 |
| Error | E19 |
| Rst | U18 |
| Clock | T18 |

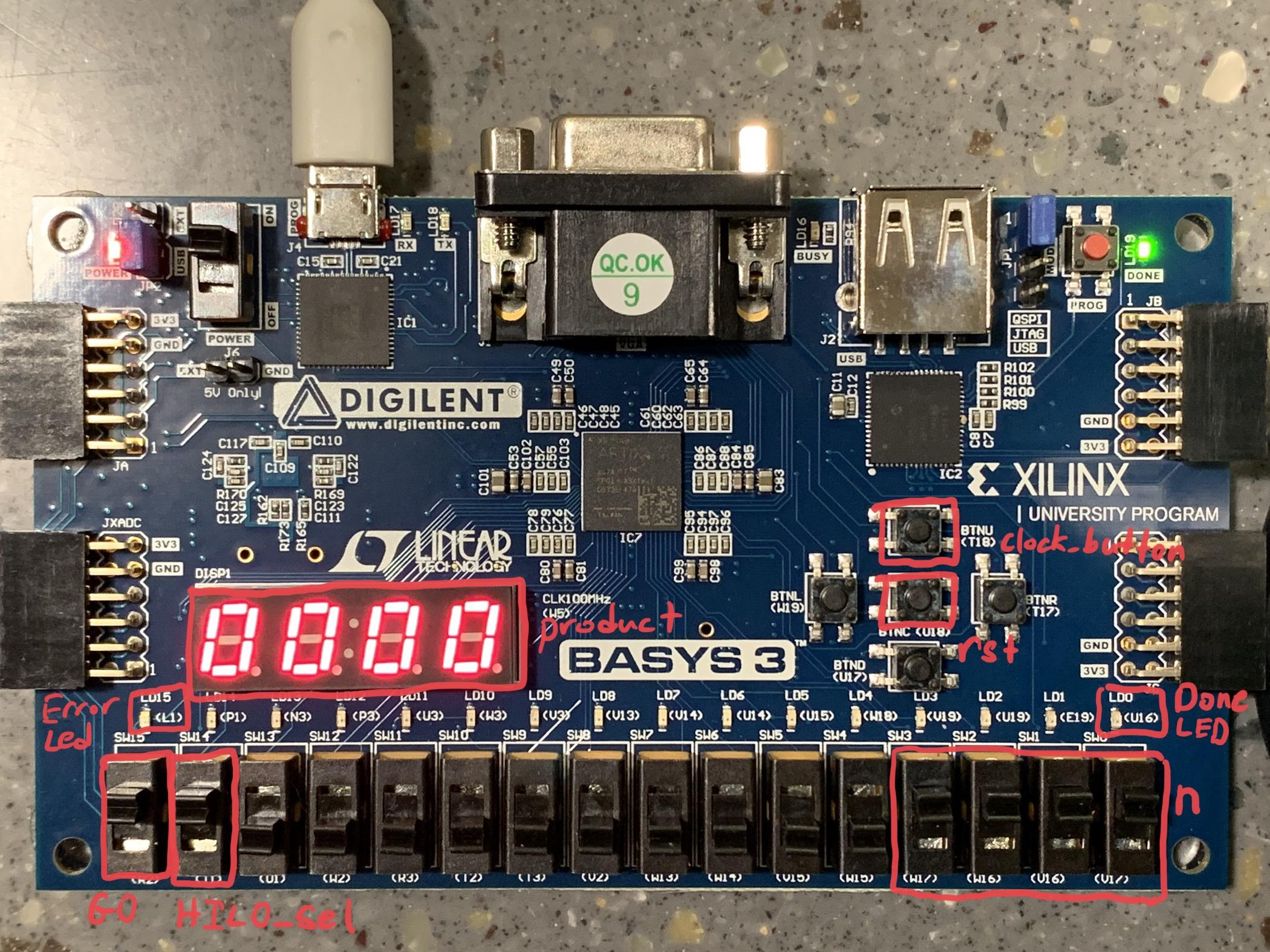


Figure K. Photo showing button, switch, and seven segment display configuration. n is inputted in binary via the rightmost switches with the rightmost switch acting as the least significant bit. The Go input signal is given by the leftmost switch. The Done and Error LEDs are marked accordingly to the right and left of the array of LEDs, respectively. The HILO\_sel switch controls which half of the 32 bit output is displayed on the display: in the LO position, the display shows the least significant 16 bits in hexadecimal format while in the HI position, the display shows the most significant 16 bits, also in hexadecimal form. The marked buttons control the clock of the Control Unit and the rst signal for the appendix modules (LEDtoMUX, clk\_gen, etc.)

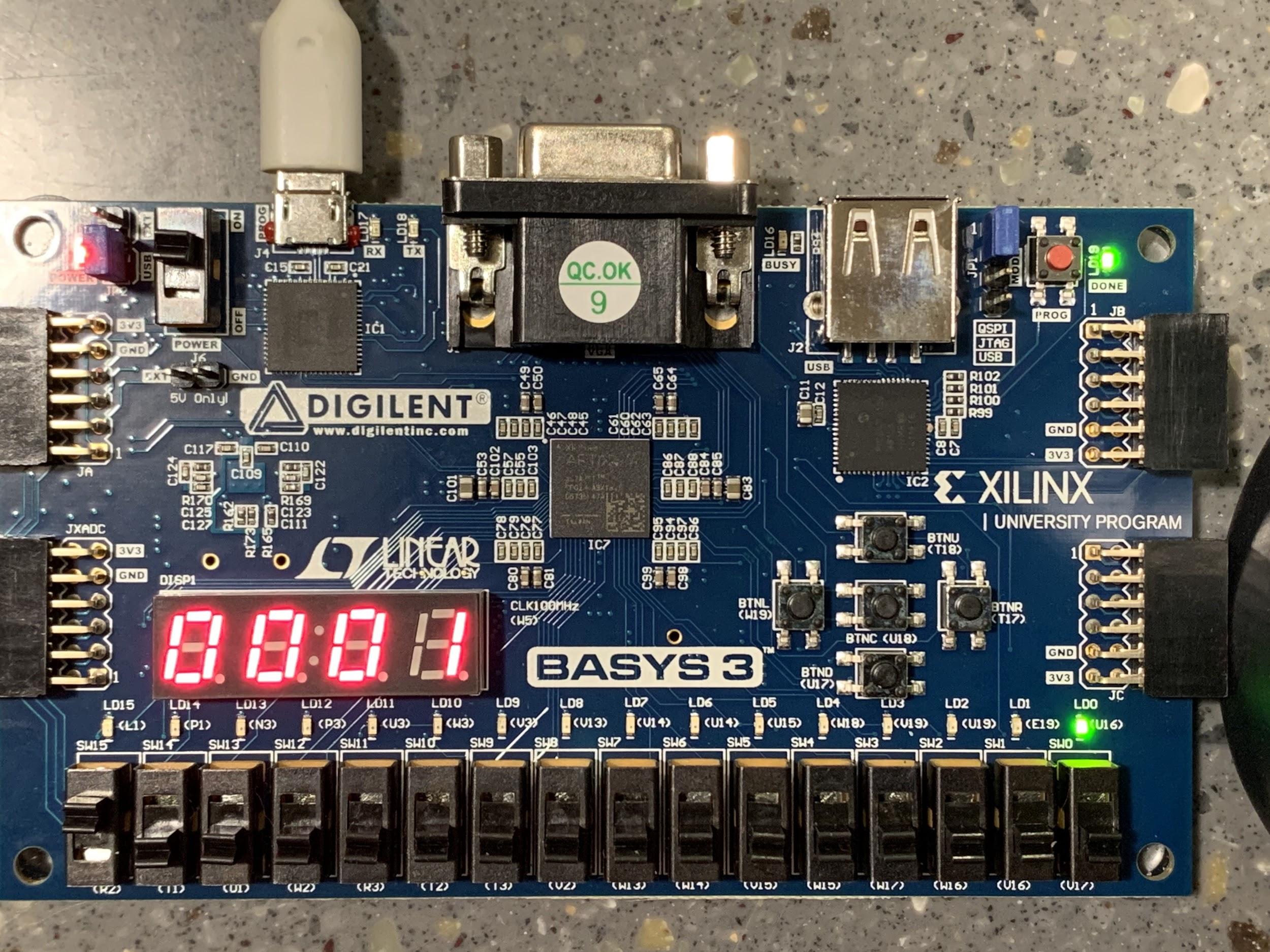


Figure L. This photo shows the calculated factorial result with n = 0. The output is displayed on the seven segment display and the Done LED is lit to show the calculation is complete.

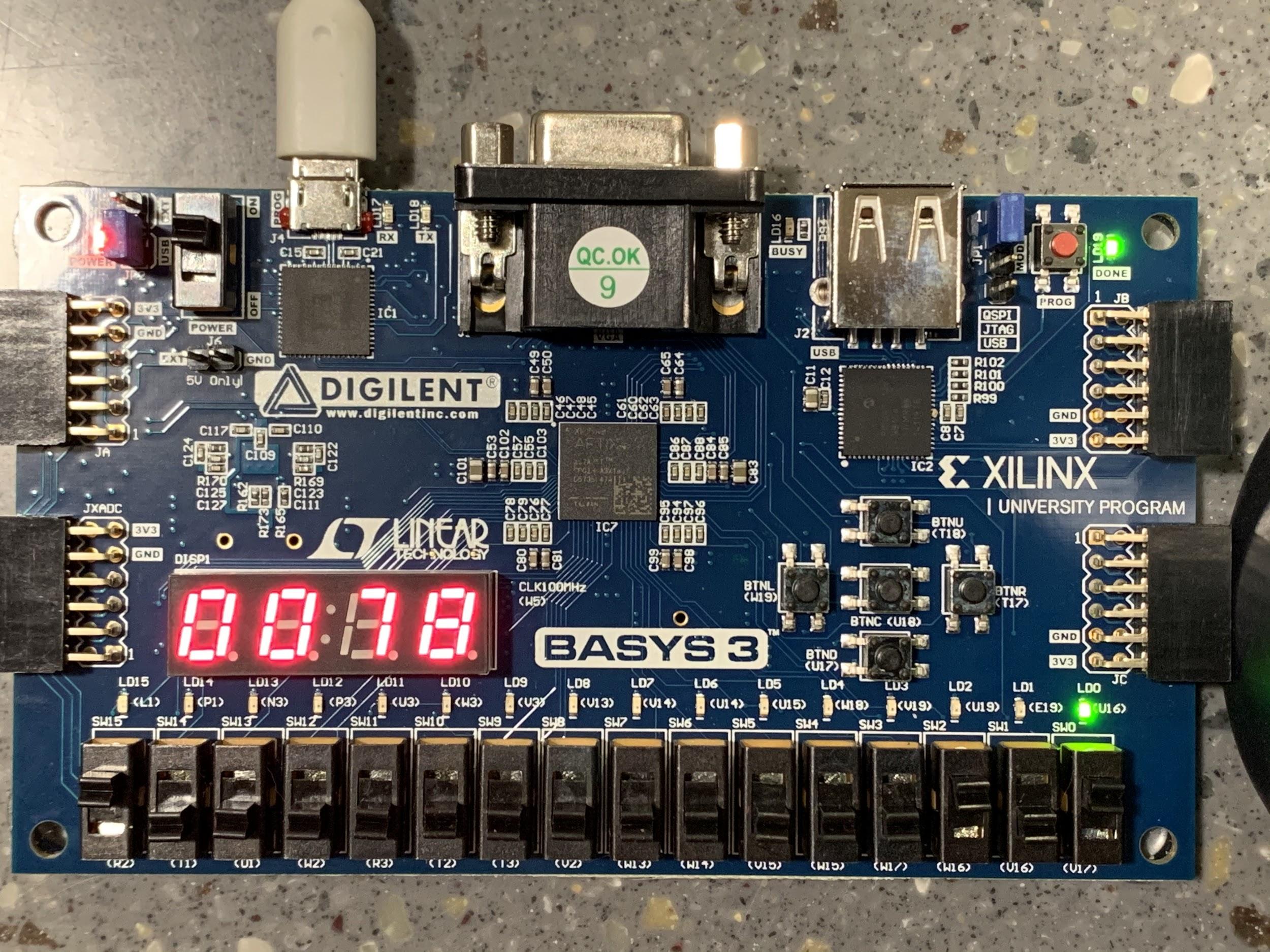


Figure M. This photo shows the calculated factorial result with n = 5. The output is displayed on the seven segment display in hexadecimal and the Done LED is lit to show the calculation is complete.

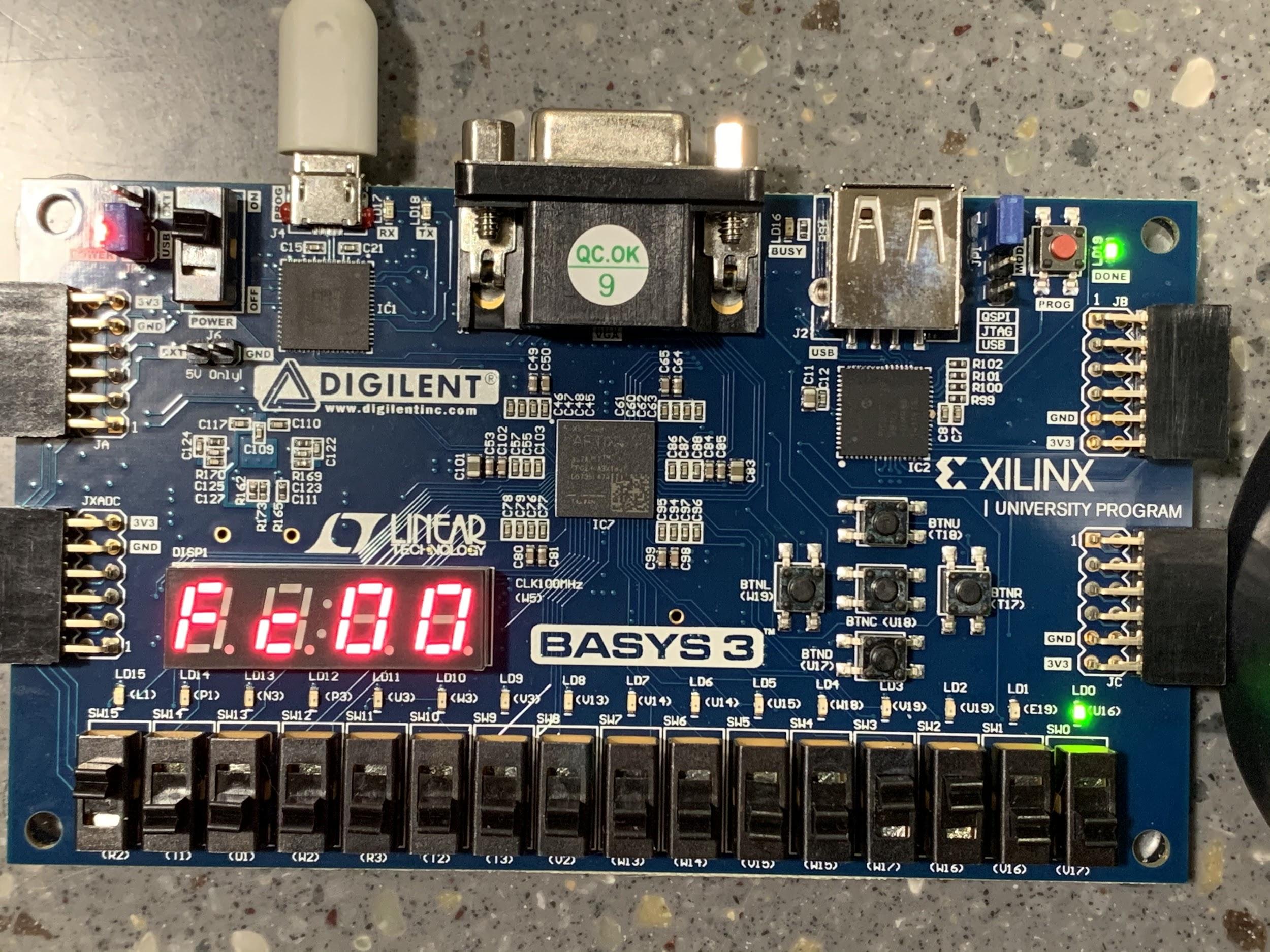


Figure N. This first photo shows the calculated factorial result with n = 12. The output is displayed on the seven segment display and the Done LED is lit to show the calculation is complete. The display is only showing the least significant 16 bits of the result as 12! results in a 32 bit result. Please see the following photo for the most significant 16 bits.

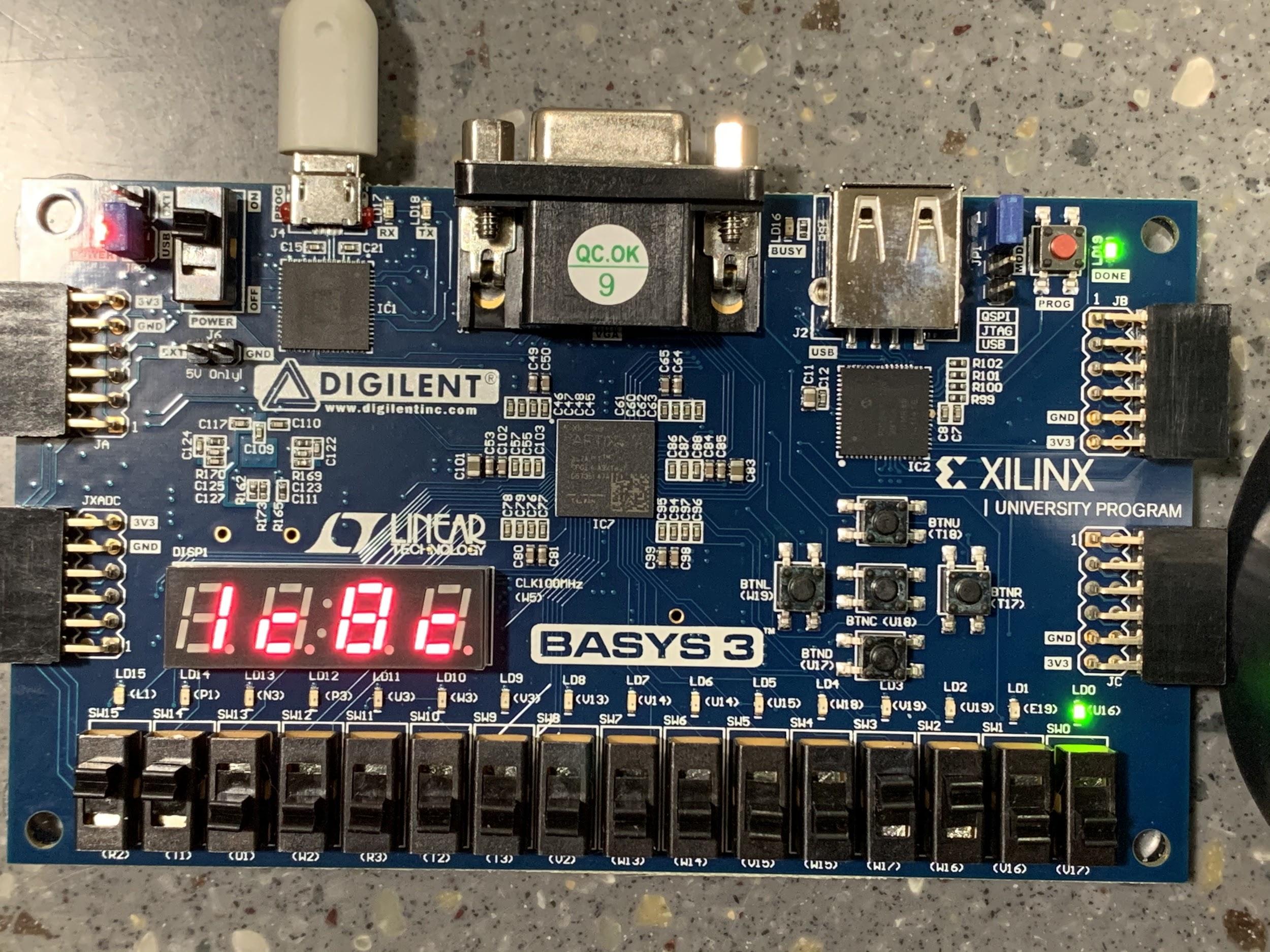


Figure O. This first photo shows the calculated factorial result with n = 12. This is a continuation of the last photo and shows the 16 most significant bits with the HILO\_sel switch set to high.

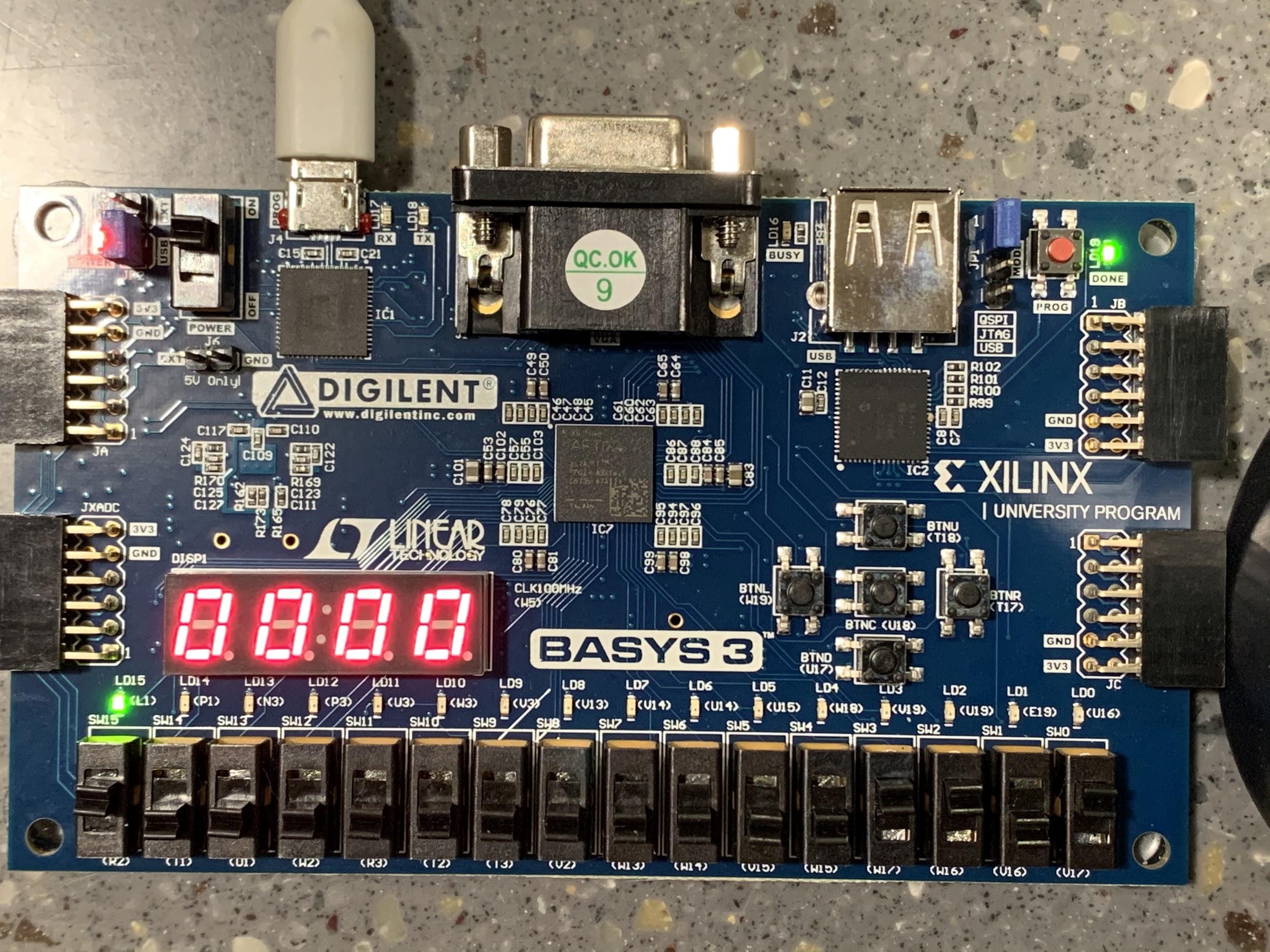


Figure P. This photo showcases the scenario when input n is greater than 12. In this case, n = 13. The Error LED is lit to indicate that a value greater than 12 was entered and the display shows 0 to indicate that no calculation was performed on the input.

**Accomplished Tasks**

1. *Designed the systems datapath*
2. *Designed the CU-DP diagram*
3. *Designed the ASM chart for the control unit*
4. *Drew the bubble diagram based on the state transition diagram*
5. *Created the CU output table*
6. *Wrote the Factorial design code and testbenches in Verilog to verify the design*
7. *Tested design and verification on the FPGA*

**Conclusion**

The purpose of the lab was fulfilled successfully. Design and verification of the FPGA board had led to a system which was able to accurately display factorial computations in hexadecimal form. The factorial computations displayed as expected for input values ranging from zero to twelve. A foundation of FPGA design was reviewed upon, leading to learning further about how to verify the control unit and datapath designs.

Verification of the control unit and datapath modules in Verilog led to learning further about how each signal can correspond to the computed output. Rather than testing all paths of the ASM chart at once, for future labs we will test one path at a time for overall efficiency and synchronicity. Upon verifying the testbench, we learned how signals can be provided to the control unit and datapath modules to test various states. The hardware validation phase led to a new understanding of how signals can be incorporated with buttons and leds to ensure that each state can be depicted in addition to the computed hexadecimal factorial result. Initially, values ranging from one to fourteen were verified. By including a computation for zero, we learned that a desired result can be produced before cycling through all states. Providing a general formula for the amount of clock cycles for each computed result was necessary. However, the formula initially was unable to predict the accurate amount of clock cycles for an input value of zero. In addition to implementing a formula for clock cycles, further waveform analysis will occur during future labs to ensure the ideal amount of clock cycles for each input value.

**Appendix A**

Table B. Function of Datapath Modules

|  |  |
| --- | --- |
| Module | Functionality |
| CMP\_Error | The CMP\_Error module provides the Error output signal when the integer provided by the user is greater than twelve. It takes an input A, B and compares A > B. If true, Error = 1. |
| CMP\_GT | The CMP\_GT module provides the GT output signal while the output of the counter is greater than one. It takes an input A, B and compares A > B. If true, GT = 1. |
| MUX1 | MUX1 selects the value of each new output of the MUL module for providing to the REG module. |
| MUX2 | MUX2 receives the factorial from REG, to provide an output of product |
| CNT | CNT is a down counter which decrements the initial input value for obtaining a factorial result |
| REG | REG stores the product result provided by MUX1 to allow retrieval from MUL and MUX2 |
| MUL | MUL multiplies the result of the down counter and the output of REG |

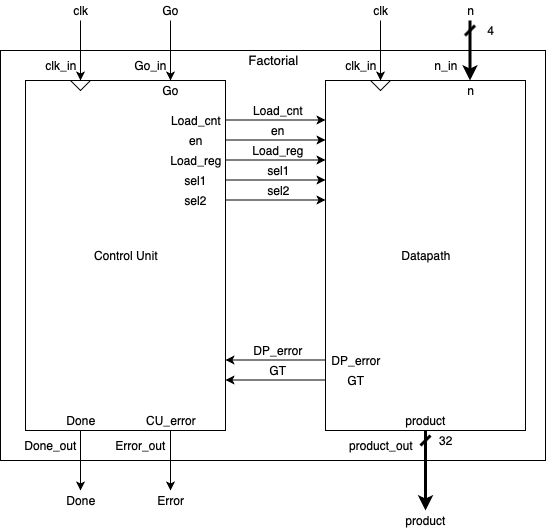


Figure Q. Control Unit and Datapath Modules

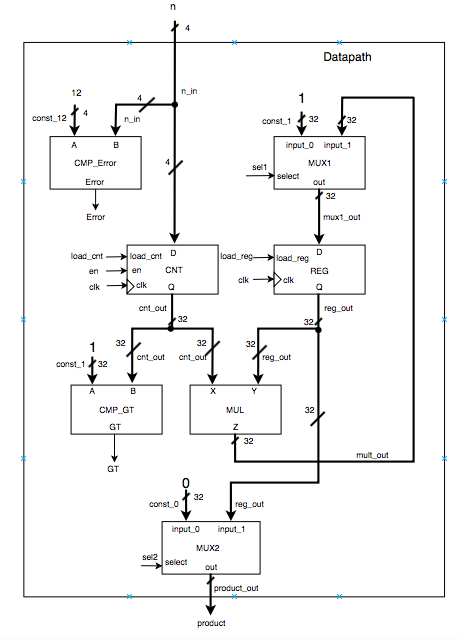


Figure R. Datapath Module

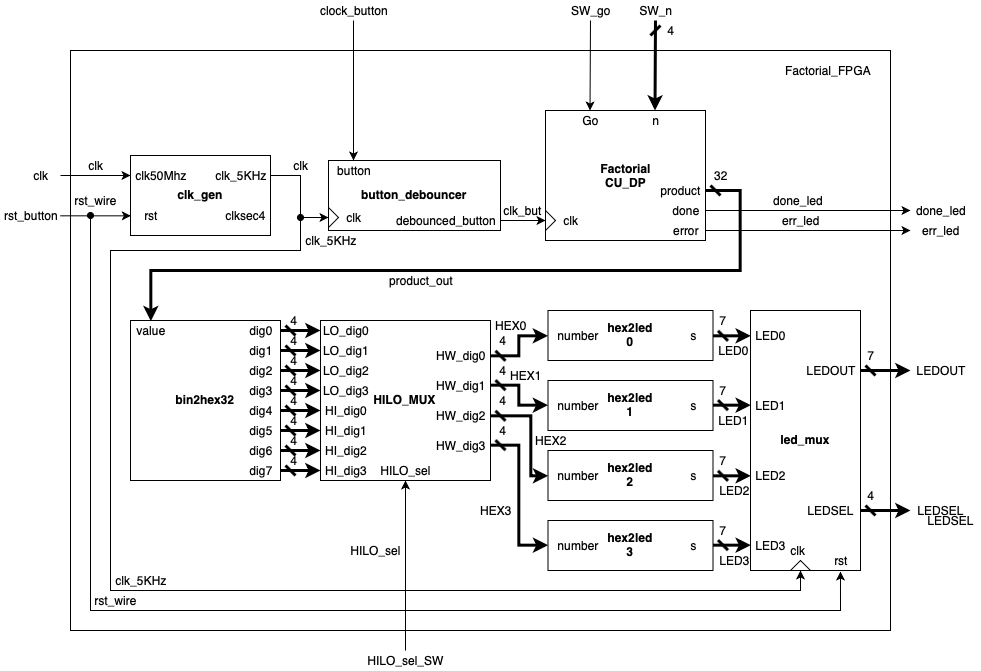


Figure S. Factorial\_FPGA

**Appendix B**

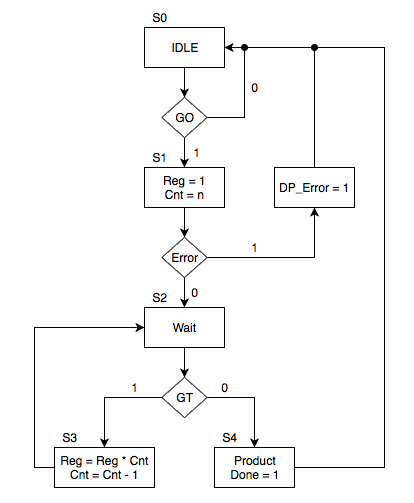


Figure T. ASM Chart

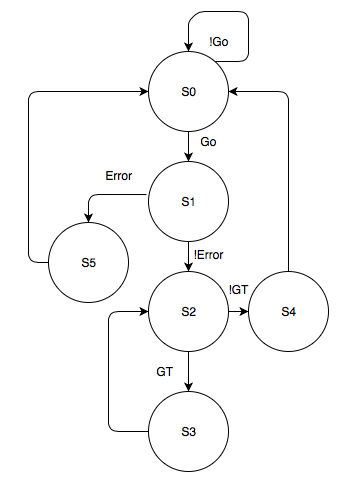
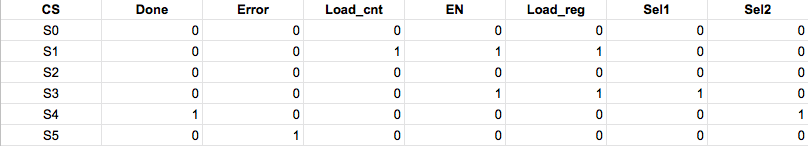


Figure U. Bubble Diagram

Figure V. CU Output Table

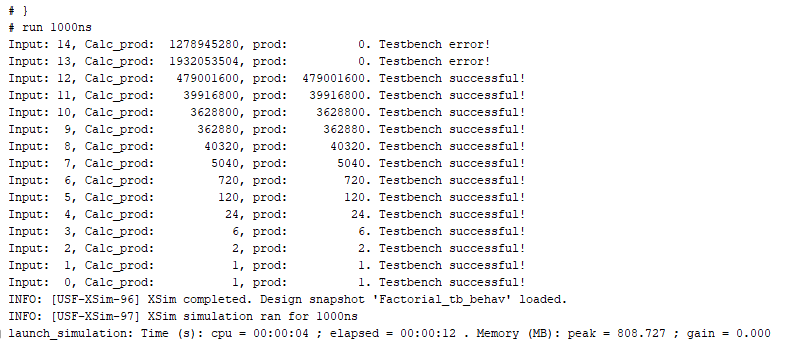
****

Figure W. Successful Testbench Output

**Appendix C**

Table C. Factorial Code

|  |
| --- |
| **Factorial\_FPGA.v** |
| `timescale 1ns / 1ps  module Factorial\_FPGA(  // From Assignment 1 Appendix  input clk,  input rst,  input HILO\_sel,  output [3:0] LEDSEL,  output [6:0] LEDOUT,    // User written  input SW\_go,  input [3:0] SW\_n,  input clock\_button,  output done\_led,  output err\_led  );  // From Assignment 1 Appendix  reg [31:0] test\_val;  // initial test\_val = 32'hfedc\_3210;  initial test\_val = 32'hba98\_7654;  wire clk\_5kHz;  wire [3:0] dig0, dig1, dig2, dig3, dig4, dig5, dig6, dig7;  wire [3:0] HEX3, HEX2, HEX1, HEX0;  wire [6:0] LED3, LED2, LED1, LED0;  // User written  wire [31:0] product\_out;  wire debounced\_button;  button\_debouncer BUT(  .button(clock\_button),  .debounced\_button(debounced\_button),  .clk(clk\_5kHz)  );  Factorial FACTO (  .go(SW\_go),  .clk(debounced\_button),  .n(SW\_n),  .done(done\_led),  .error(err\_led),  .product(product\_out)  );    clk\_gen CLK0 (  .clk50MHz(clk),  .rst(rst),  //clksec4,  .clk\_5KHz(clk\_5kHz)  );  bin2hex32 CONV1(  .value(product\_out),  .dig0(dig0),  .dig1(dig1),  .dig2(dig2),  .dig3(dig3),  .dig4(dig4),  .dig5(dig5),  .dig6(dig6),  .dig7(dig7)  );  HILO\_MUX HLM(  .HI\_dig3(dig7),  .HI\_dig2(dig6),  .HI\_dig1(dig5),  .HI\_dig0(dig4),  .LO\_dig3(dig3),  .LO\_dig2(dig2),  .LO\_dig1(dig1),  .LO\_dig0(dig0),  .HILO\_sel(HILO\_sel),  .HW\_dig3(HEX3),  .HW\_dig2(HEX2),  .HW\_dig1(HEX1),  .HW\_dig0(HEX0)  );  hex2led U\_LD\_3(  .number(HEX3),  .s0(LED3[0]),  .s1(LED3[1]),  .s2(LED3[2]),  .s3(LED3[3]),  .s4(LED3[4]),  .s5(LED3[5]),  .s6(LED3[6])  );  hex2led U\_LD\_2(  .number(HEX2),  .s0(LED2[0]),  .s1(LED2[1]),  .s2(LED2[2]),  .s3(LED2[3]),  .s4(LED2[4]),  .s5(LED2[5]),  .s6(LED2[6])  );  hex2led U\_LD\_1(  .number(HEX1),  .s0(LED1[0]),  .s1(LED1[1]),  .s2(LED1[2]),  .s3(LED1[3]),  .s4(LED1[4]),  .s5(LED1[5]),  .s6(LED1[6])  );  hex2led U\_LD\_0(  .number(HEX0),  .s0(LED0[0]),  .s1(LED0[1]),  .s2(LED0[2]),  .s3(LED0[3]),  .s4(LED0[4]),  .s5(LED0[5]),  .s6(LED0[6])  );  LED\_MUX U3(  .clk(clk\_5kHz),  .rst(rst),  .LED0(LED0),  .LED1(LED1),  .LED2(LED2),  .LED3(LED3),  .LEDOUT(LEDOUT),  .LEDSEL(LEDSEL)  );  endmodule |

Table D. CU Code

|  |
| --- |
| **CU.v** |
| `timescale 1ns / 1ps  /\* Source Code for the Building Blocks Used \*/  module CU(  input go,  input clk,  input GT,  input DP\_error,  output reg load\_cnt,  output reg en,  output reg load\_reg,  output reg sel1,  output reg sel2,  output reg done,  output reg CU\_error  );  integer state0 = 3'b000;  integer state1 = 3'b001;  integer state2 = 3'b010;  integer state3 = 3'b011;  integer state4 = 3'b100;  integer state5 = 3'b101;  reg [2:0] cs = 3'b000;  reg [2:0] ns;  //reg [6:0] cw;  //always@(cw){load\_cnt, en, load\_reg, sel1, sel2, done, CU\_error} = cw;  // cw stands for control word. This control word refers to each output bit sent by the CU to the DP  always@(posedge clk)  cs <= ns;  always@(\*) begin  case (cs)  state0: begin  done = 0;  CU\_error = 0;  load\_cnt = 0;  en = 0;  load\_reg = 0;  sel1 = 0;  sel2 = 0;  if(!go)  ns = state0;  else  ns = state1;  end  state1: begin  done = 0;  CU\_error = 0;  load\_cnt = 1;  en = 1;  load\_reg = 1;  sel1 = 0;  sel2 = 0;  if(DP\_error)  ns = state5;  else  ns = state2;  end  state2: begin  done = 0;  CU\_error = 0;  load\_cnt = 0;  en = 0;  load\_reg = 0; // updated  sel1 = 0; //updated  sel2 = 0;  if(GT)  ns = state3;  else  ns = state4;  end  state3: begin  done = 0;  CU\_error = 0;  load\_cnt = 0;  en = 1;  load\_reg = 1;  sel1 = 1;  sel2 = 0;  ns = state2;  end  state4: begin  done = 1;  CU\_error = 0;  load\_cnt = 0;  en = 0;  load\_reg = 0;  sel1 = 0;  sel2 = 1;  ns = state0;  end  state5: begin  done = 0;  CU\_error = 1;  load\_cnt = 0;  en = 0;  load\_reg = 0;  sel1 = 0;  sel2 = 0;  ns = state0;  end  default: begin  done = 0;  CU\_error = 1;  load\_cnt = 0;  en = 0;  load\_reg = 0;  sel1 = 0;  sel2 = 1;  ns = state0;  end    endcase  end  endmodule |

Table E. DP Code

|  |
| --- |
| **DP.v** |
| `timescale 1ns / 1ps  module DP(  input [3:0] n,  input load\_cnt,  input en,  input load\_reg,  input sel1,  input sel2,  input clk,  output Error,  output GT,  output [31:0] product///reg or wire  );      wire [31:0] mux1\_out;  wire [31:0] reg\_out;  wire [31:0] cnt\_out;  wire [31:0] mult\_out;  // instantiate the building blocks, within parentheses is the wire or include same name  CMP\_ERROR U0(.A(4'b1100), .B(n), .ERROR(Error));  CMP\_GT U1(.A(32'h00000001), .B(cnt\_out), .GT(GT));  CNT U2(.D(n), .load\_cnt(load\_cnt), .en(en), .clk(clk), .Q(cnt\_out));  MUL U3(.X(cnt\_out), .Y(reg\_out),.Z(mult\_out));  MUX M1(.input\_0(1),.input\_1(mult\_out), .select(sel1), .out(mux1\_out));  MUX M2(.input\_0(0),.input\_1(reg\_out), .select(sel2), .out(product));  REG R1(.D(mux1\_out),.Q(reg\_out), .clk(clk), .load\_reg(load\_reg));  endmodule //DP |

Table F. Factorial Code

|  |
| --- |
| **Factorial.v** |
| module Factorial(  input wire go, clk,  input wire [3:0] n,  output wire done, error,  output wire [31:0] product  );  wire load\_cnt, en, load\_reg, sel1, sel2, DP\_error, GT;  DP DP0(  .n(n),  .load\_cnt(load\_cnt),  .en(en),  .load\_reg(load\_reg),  .sel1(sel1),  .sel2(sel2),  .clk(clk),  .Error(DP\_error),  .GT(GT),  .product(product)  );  CU CU0(  .go(go),  .clk(clk),  .GT(GT),  .DP\_error(DP\_error),  .load\_cnt(load\_cnt),  .en(en),  .load\_reg(load\_reg),  .sel1(sel1),  .sel2(sel2),  .done(done),  .CU\_error(error)  );  endmodule |

Table G. CMP\_Error Code

|  |
| --- |
| **CMP\_Error.v** |
| `timescale 1ns / 1ps  module CMP\_ERROR(  input [3:0] A, B,  output reg ERROR  );  always @ (A, B) begin  if (B > A)  ERROR = 1;  else  ERROR = 0;  end  endmodule |

Table H. CMP\_GT Code

|  |
| --- |
| **CMP\_GT.v** |
| `timescale 1ns / 1ps  module CMP\_GT(  input [31:0] A, B,  output reg GT  );  always @ (A, B) begin  if (B > A)  GT = 1;  else  GT = 0;  end  endmodule |

Table I. CNT Code

|  |
| --- |
| **CNT.v** |
| `timescale 1ns / 1ps  module CNT(  input [3:0] D,  input load\_cnt, en, clk,  output reg [31:0] Q  );    always @ (posedge clk) begin  if (en) begin  if (load\_cnt)  Q <= D;  else  Q <= Q - 1;  end else  Q <= Q;  end    endmodule |

Table J. MUX Code

|  |
| --- |
| **MUX.v** |
| `timescale 1ns / 1ps  module MUX(  input [31:0] input\_0, input\_1,  input select,  output reg [31:0] out  );    always @ (input\_0, input\_1, select)  begin  if (select)  out = input\_1;  else  out = input\_0;  end  endmodule |

Table K. REG Code

|  |
| --- |
| **REG.v** |
| `timescale 1ns / 1ps  module REG(  input [31:0] D,  input clk, load\_reg,  output reg [31:0] Q  );    always @ (posedge clk)  if(load\_reg)  Q <= D;  else  Q <= Q;  endmodule |

Table L. MUL Code

|  |
| --- |
| **MUL.v** |
| `timescale 1ns / 1ps  module MUL(  input [31:0] X, Y,  output reg [31:0] Z  );    always @ (X, Y) begin  Z = X \* Y;  end  endmodule |

Table M. LED\_MUX Code

|  |
| --- |
| **LED\_MUX.v** |
| module LED\_MUX (clk, rst, LED0, LED1, LED2, LED3, LEDOUT, LEDSEL);  input clk, rst;  input [6:0] LED0, LED1, LED2, LED3;  output[3:0] LEDSEL;  output[6:0] LEDOUT;  reg [3:0] LEDSEL;  reg [6:0] LEDOUT;  reg [1:0] index;    always @(posedge clk)  begin  if(rst)  index = 0;  else  index = index + 1;  end    always @(index or LED0 or LED1 or LED2 or LED3)  begin  case(index)  0: begin  LEDSEL = 4'b1110;  LEDOUT = LED0;  end  1: begin  LEDSEL = 4'b1101;  LEDOUT = LED1;  end  2: begin  LEDSEL = 4'b1011;  LEDOUT = LED2;  end  3: begin  LEDSEL = 4'b0111;  LEDOUT = LED3;  end  default: begin  LEDSEL = 0; LEDOUT = 0;  end  endcase  end  endmodule |

Table N. HILO\_MUX Code

|  |
| --- |
| **HILO\_MUX.v** |
| module HILO\_MUX(  input wire [3:0] HI\_dig3,  input wire [3:0] HI\_dig2,  input wire [3:0] HI\_dig1,  input wire [3:0] HI\_dig0,    input wire [3:0] LO\_dig3,  input wire [3:0] LO\_dig2,  input wire [3:0] LO\_dig1,  input wire [3:0] LO\_dig0,    input wire HILO\_sel,    output wire [3:0] HW\_dig3,  output wire [3:0] HW\_dig2,  output wire [3:0] HW\_dig1,  output wire [3:0] HW\_dig0  );  assign HW\_dig3 = HILO\_sel ? HI\_dig3 : LO\_dig3;  assign HW\_dig2 = HILO\_sel ? HI\_dig2 : LO\_dig2;  assign HW\_dig1 = HILO\_sel ? HI\_dig1 : LO\_dig1;  assign HW\_dig0 = HILO\_sel ? HI\_dig0 : LO\_dig0;  endmodule |

Table O. hex2led Code

|  |
| --- |
| **hex2led.v** |
| module hex2led(number, s0, s1, s2, s3, s4, s5, s6);  output s0, s1, s2, s3, s4, s5, s6;  input [3:0] number;  reg s0, s1, s2, s3, s4, s5, s6;  always @ (number)  begin // BCD to 7-segment decoding  case (number) // s0 - s6 are active low  4'h0: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=1; end  4'h1: begin s0=1; s1=0; s2=0; s3=1; s4=1; s5=1; s6=1; end  4'h2: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=1; s6=0; end  4'h3: begin s0=0; s1=0; s2=0; s3=0; s4=1; s5=1; s6=0; end  4'h4: begin s0=1; s1=0; s2=0; s3=1; s4=1; s5=0; s6=0; end  4'h5: begin s0=0; s1=1; s2=0; s3=0; s4=1; s5=0; s6=0; end  4'h6: begin s0=0; s1=1; s2=0; s3=0; s4=0; s5=0; s6=0; end  4'h7: begin s0=0; s1=0; s2=0; s3=1; s4=1; s5=1; s6=1; end  4'h8: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=0; s6=0; end  4'h9: begin s0=0; s1=0; s2=0; s3=1; s4=1; s5=0; s6=0; end  4'ha: begin s0=0; s1=0; s2=0; s3=0; s4=0; s5=1; s6=0; end  4'hb: begin s0=1; s1=1; s2=0; s3=0; s4=0; s5=0; s6=0; end  4'hc: begin s0=1; s1=1; s2=1; s3=0; s4=0; s5=1; s6=0; end  4'hd: begin s0=1; s1=0; s2=0; s3=0; s4=0; s5=1; s6=0; end  4'he: begin s0=0; s1=0; s2=1; s3=0; s4=0; s5=0; s6=0; end  4'hf: begin s0=0; s1=1; s2=1; s3=1; s4=0; s5=0; s6=0; end  default: begin s0=1; s1=1; s2=1; s3=1; s4=1; s5=1; s6=1; end  endcase  end  endmodule // end led |

Table P. clk\_gen Code

|  |
| --- |
| **clk\_gen.v** |
| `timescale 1ns / 1ps  // Code taken from Assignment 1 Appendix  module clk\_gen(clk50MHz, rst, clksec4, clk\_5KHz);  input clk50MHz, rst;  output clksec4, clk\_5KHz;  reg clksec4, clk\_5KHz;  integer count, count1;  always@(posedge clk50MHz)  begin  if(rst)  begin  count = 0;  count1 = 0;  clksec4 = 0;  clk\_5KHz =0;  end  else  begin  if(count == 100000000)  begin  clksec4 = ~clksec4;  count = 0;  end  if(count1 == 20000)  begin  clk\_5KHz = ~clk\_5KHz;  count1 = 0;  end  count = count + 1;  count1 = count1 + 1;  end  end  endmodule |

Table Q. bin2hex32 Code

|  |
| --- |
| **bin2hex32.v** |
| module bin2hex32(  input wire [31:0] value,  output wire [3:0] dig0,  output wire [3:0] dig1,  output wire [3:0] dig2,  output wire [3:0] dig3,  output wire [3:0] dig4,  output wire [3:0] dig5,  output wire [3:0] dig6,  output wire [3:0] dig7  );  assign dig0 = value & 4'hFF;  assign dig1 = value >> 4 & 4'hFF;  assign dig2 = value >> 8 & 4'hFF;  assign dig3 = value >> 12 & 4'hFF;  assign dig4 = value >> 16 & 4'hFF;  assign dig5 = value >> 20 & 4'hFF;  assign dig6 = value >> 24 & 4'hFF;  assign dig7 = value >> 28 & 4'hFF;  endmodule |

Table R. basys3\_factorial\_cstr Code

|  |
| --- |
| **basys3\_factorial\_cstr.xdc** |
| ## This file is a general .xdc for the Basys3 rev B board  ## To use it in a project:  ## - uncomment the lines corresponding to used pins  ## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project  ## Code taken from Assignment 1 Appendix  ## Clock signal  set\_property PACKAGE\_PIN W5 [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]    # Switches  ## Switches  set\_property PACKAGE\_PIN V17 [get\_ports {SW\_n[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW\_n[0]}]  set\_property PACKAGE\_PIN V16 [get\_ports {SW\_n[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW\_n[1]}]  set\_property PACKAGE\_PIN W16 [get\_ports {SW\_n[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW\_n[2]}]  set\_property PACKAGE\_PIN W17 [get\_ports {SW\_n[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW\_n[3]}]  set\_property PACKAGE\_PIN T1 [get\_ports {HILO\_sel}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {HILO\_sel}]  set\_property PACKAGE\_PIN R2 [get\_ports {SW\_go}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {SW\_go}]    ## LEDs  set\_property PACKAGE\_PIN U16 [get\_ports {done\_led}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {done\_led}]  set\_property PACKAGE\_PIN L1 [get\_ports {err\_led}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {err\_led}]      ##7 segment display  set\_property PACKAGE\_PIN W7 [get\_ports {LEDOUT[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[0]}]  set\_property PACKAGE\_PIN W6 [get\_ports {LEDOUT[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[1]}]  set\_property PACKAGE\_PIN U8 [get\_ports {LEDOUT[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[2]}]  set\_property PACKAGE\_PIN V8 [get\_ports {LEDOUT[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[3]}]  set\_property PACKAGE\_PIN U5 [get\_ports {LEDOUT[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[4]}]  set\_property PACKAGE\_PIN V5 [get\_ports {LEDOUT[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[5]}]  set\_property PACKAGE\_PIN U7 [get\_ports {LEDOUT[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDOUT[6]}]  set\_property PACKAGE\_PIN U2 [get\_ports {LEDSEL[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[0]}]  set\_property PACKAGE\_PIN U4 [get\_ports {LEDSEL[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[1]}]  set\_property PACKAGE\_PIN V4 [get\_ports {LEDSEL[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[2]}]  set\_property PACKAGE\_PIN W4 [get\_ports {LEDSEL[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {LEDSEL[3]}]  ##Buttons  set\_property PACKAGE\_PIN U18 [get\_ports rst]  set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  set\_property PACKAGE\_PIN T18 [get\_ports clock\_button]  set\_property IOSTANDARD LVCMOS33 [get\_ports clock\_button] |

Table S. Factorial TB Code

|  |
| --- |
| **Factorial\_tb.v** |
| `timescale 1ns / 1ps  module Factorial\_tb;  reg [3:0] n;  reg go, clk;  wire done, error;  wire[31:0] product;  task tick;  begin  #1 clk = 0;  #1 clk = 1;  end  endtask  Factorial DUT0(.n(n), .go(go), .clk(clk), .done(done),.error(error), .product(product));  integer i = 0;  integer j = 0;  integer x = 0;  integer k = 0;  integer calc\_product = 0;  initial begin  for (i = 14; i >= 0; i = i - 1) begin  n = i;  go = 1;  x = (2\*n)+2; // Calculates how many clock cycles to find n!  calc\_product = 1;    // Calculates actual factorial value  for (k = i; k > 0; k = k - 1 ) begin  calc\_product = k\*calc\_product;  end    // Calculates factorial value in design  for (j = 1; j < x; j = j + 1) begin  tick;  if(x == 2) // Ensures system properly calculates n! when n = 0  begin  tick;  tick;  end;  end    if ( product == calc\_product)  $display("Input: %d, Calc\_prod: %d, prod: %d. Testbench successful!", n, calc\_product, product);  else  $display("Input: %d, Calc\_prod: %d, prod: %d. Testbench error!", n, calc\_product, product);    tick; // Advance clock  end  end  endmodule |

Table T. CU Testbench Code

|  |
| --- |
| **CU\_tb.v** |
| `timescale 1ns / 1ps  module CU\_tb;  reg go, clk, GT, DP\_error;  wire load\_cnt, en, load\_reg, sel1, sel2, done, CU\_error;    task tick;  begin  #5  clk = !clk;  #5  clk = !clk;  end  endtask    initial begin  $display("Start DP Test");  //state 0  GT = 0;  DP\_error = 0;  clk = 0;  go = 1;    tick; //goes to state 1    DP\_error = 1;    tick; //goes to state 5  DP\_error = 0;  tick; //goes to state 0  tick;  tick; // goes to state 1      GT = 1;    tick; //goes to state 2        tick; //goes to state 3    GT = 0;      tick; //goes to state 2    tick;//goes to state 4    go = 0;    tick; //goes to state 0          if(done == 1)begin  $display("Finished CU Test");  end  end    CU DUT(  .go(go),  .clk(clk),  .GT(GT),  .DP\_error(DP\_error),  .load\_cnt(load\_cnt),  .en(en),  .load\_reg(load\_reg),  .sel1(sel1),  .sel2(sel2),  .done(done),  .CU\_error(CU\_error)  );      endmodule |

Table U. DP Testbench Code

|  |
| --- |
| **DP\_tb.v** |
| module DP\_tb;  reg [3:0] n;  reg load\_cnt, en, sel1, sel2, load\_reg, clk;  wire Error, GT;  wire [31:0] product;    DP DUT(  .n(n),  .load\_cnt(load\_cnt),  .en(en),  .sel1(sel1),  .sel2(sel2),  .load\_reg(load\_reg),  .clk(clk),  .Error(Error),  .GT(GT),  .product(product)  );    task tick;  begin  #5  clk = !clk;  #5  clk = !clk;  end  endtask    initial begin  $display("Start DP Test");  // S0  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 0;  load\_reg = 0;  clk = 0;  n = 15;    tick; // S0 -> S1    // S1  load\_cnt = 1;  en = 1;  sel1 = 0;  sel2 = 0;  load\_reg = 1;    tick; // S1 -> S5    // S5  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 0;  load\_reg = 0;    tick; // S5 -> S0    // S0  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 0;  load\_reg = 0;  clk = 0;  n = 4;    tick; // S0 -> S1    // S1  load\_cnt = 1;  en = 1;  sel1 = 0;  sel2 = 0;  load\_reg = 1;    tick; // S1 -> S2    // S2  load\_cnt = 0;  en = 0; // 1?  sel1 = 0; // 1?  sel2 = 0;  load\_reg = 0;    tick; // S2 -> S3    // S3  load\_cnt = 0;  en = 1;  sel1 = 1;  sel2 = 0;  load\_reg = 1;    tick; // S3 -> S2  // S2  load\_cnt = 0;  en = 0; // 1?  sel1 = 0; // 1?  sel2 = 0;  load\_reg = 0;    tick; // S2 -> S3    // S3  load\_cnt = 0;  en = 1;  sel1 = 1;  sel2 = 0;  load\_reg = 1;    tick; // S3 -> S2    // S2  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 0;  load\_reg = 0;    tick; // S2 -> S3    // S3  load\_cnt = 0;  en = 1;  sel1 = 1;  sel2 = 0;  load\_reg = 1;    tick; // S3 -> S2    // S2  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 0;  load\_reg = 0;    tick; // S2 -> S4    // S4  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 1;  load\_reg = 0;    tick; // S4 -> S0    // S0  load\_cnt = 0;  en = 0;  sel1 = 0;  sel2 = 1;  load\_reg = 0;    $display("Finished DP Test");  end    endmodule |

**Headshots and description of self**

****

**Sidarth Shahri:** I am a senior computer engineering student! My hobbies include rock climbing, hiking, biking, and video games.



**Karine Worley:** I am also in senior year studying computer engineering. My hobbies are reading, photography, and listening to music.