ELEX 7660 – Digital System Design

Project: Rubik's Cube Solver

Calvin Lee

Jake Jarvis

Luke Pankratz

Table of Contents

Project Introduction

Part of the requirements of the ELEX 7660 Digital System Design course was a project component in which students apply their own knowledge of programming and engineering along with the material learned in class to create a project that demonstrates to the instructor what they can do or that helps further their own future careers. The project was open-ended, with a few constraints: we were free to create any project we wanted to as long as a majority of the programming and functionality came from the provided FPGA, and were only provided $25 per person to purchase parts. The course focused around learning the System Verilog hardware description language and the use of it to program an FPGA for various applications.

Project Overview

For us, we thought that solving a Rubik's Cube seemed like a reasonable challenge that could be done with an FPGA. At first, we didn't think this project would be too hard of a challenge, other than learning how to solve a Rubik's Cube. After some research online, we were able to find other people who have attempted the same project, and they were nice enough to make their own reports and code available for public viewing. Many of our ideas came from what we were able to find online: using servo motors to drive claws which would hold and rotate the cube and its faces, and the number of claws we decided to implement in our design.

During our initial research, we found that a lot of the code needed for automatic solving of the cube and ways of throttling the correct sequence of commands to the multiple servos was going to be very complex, and we spent a lot of time trying to decipher how it was done by others. We also did not have a way for the FPGA to read the faces of the cube, meaning we would have had to input it manually for the cube to automatically solve it. We eventually decided to abandon automatic solving because it was too time-consuming to write the hundreds of lines of code and algorithms needed in the C program.

Our final design included the FPGA, a keypad, three claws, and six servo motors, due to time constraints and lack of materials. We had originally wanted to implement four claws and eight servo motors, and to use C programming and the Nios platform to automatically solve the Rubik's Cube and feed the required moves through to the FPGA. The face colours would have been input manually by adjusting an array in the C program prior to operation, and the algorithms for moves would have been taken from the source code we found online. Instead, we cut down on our design and used keypad inputs to turn the claws and cube in different ways.

Materials

The servos used were the Hitec HS-422 standard size R/C motors. These are powered by a DC voltage of 4.8 to 6 volts and are controlled by pulse width modulation of a signal fed into its third pin. The pulses run at 50 Hz (20 ms) and the pulse width is anywhere from 1 to 2 ms long. A pulse width of 1.5 ms coincides with the motor being in the neutral position, and increasing it up to 2 ms will turn the motor clockwise 45˚ (likewise in the other direction, a 1 ms pulse will cause a -45˚ degree turn). These servos are used to rotate and operate gripper claws which hold and turn the cube and its faces.

The claws used were the Standard Gripper Kit A, which are designed to be used with standard servos like the Hitec HS-422. Both of these products were purchased through Sparkfun.

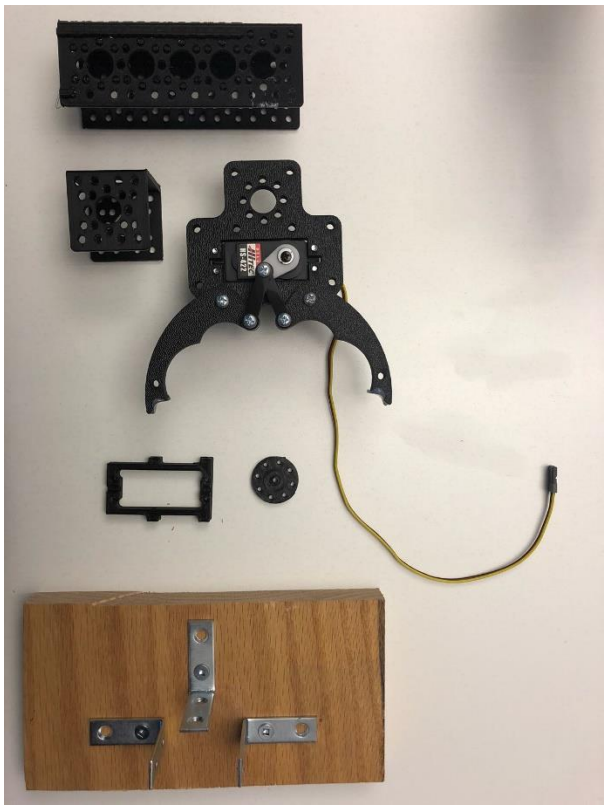

Figure 1 - Claw and servo, next to 3D printed brackets and supports

The Rubik's Cube is just a standard cube, purchasable from many toy stores or online. Quality of the cube should be important, as we want a cube that is well oiled so that its faces may turn smoothly without interfering with claw and servo operation.

The FPGA used was the one provided to us and used in all our labs for the course, the Cyclone IV EP4CE22F17C6, which was mounted on a Terasic DE0-Nano development board to allow interfacing to the FPGA via external pin headers.

Orientation

The three claws were positioned with two opposite from each other and the last one 90˚ from the others, forming a T-shape with the claws facing the cube in the centre. The claws and motors are held together by brackets made by 3D printing and mounted on wooden boards. Wires run from each of the motors to a small breadboard, which connects the wires to the input and output pins of the FPGA board. The FPGA board was also connected to the keypad used in the second lab, to provide a method of sending commands to the FPGA which would control the servos.

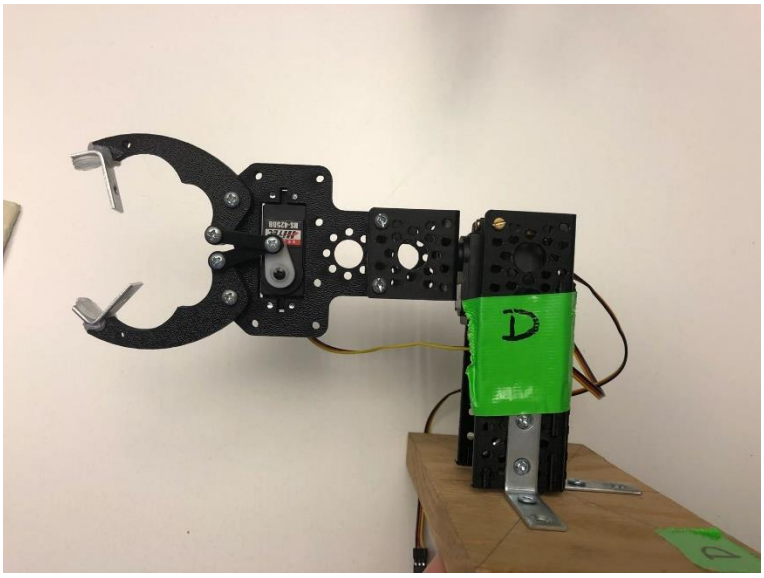

Figure 2 - Claw and servo, mounted on wooden board

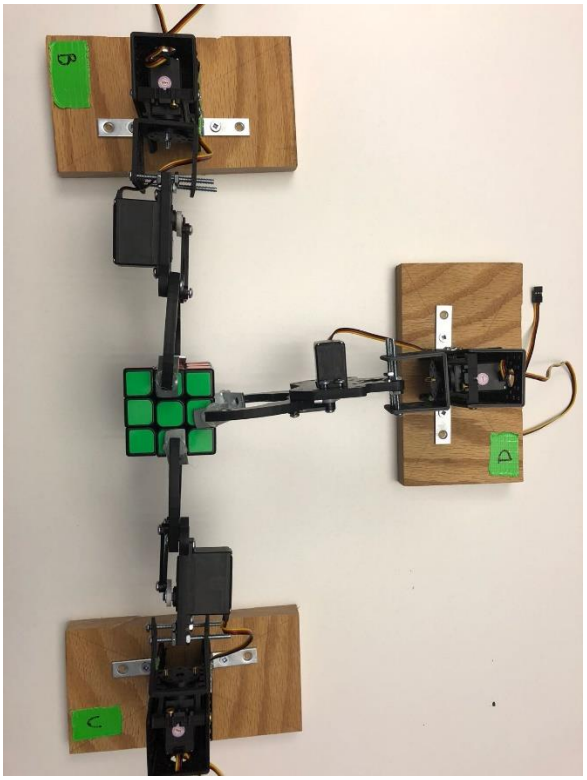

Figure 3 - End result, with claws holding cube

We had initially wanted four claws to hold the cube up, so that rotate the cube in any direction would be easier, but we had to settle for three claws because of schedule delays and taking too much time finding a solution for properly providing the modulated pulses needed to run the servos.

Modules

Our Verilog code was broken up into a few modules: A column sequencing module that helps to detect which key was pressed on the keypad, one which decoded the keypad input into a numerical value, and another module that took the numerical inputs and converted them into a sequence of commands of the servos to go through to turn the cube.

0.1 Column Sequencer and Decoder

The column sequencer and decoder were taken from our lab two work, which involved writing code to enable detection of keypresses on the provided keypad. When a certain row is set low, the code will toggle an output on each column to detect which key is providing a low signal. The low signal is then used to activate an input, which is converted into a number corresponding to the value shown on the keypad, and output to wherever we need it. In our case, the output number would be used to initiate commands for rotating the cube or its faces.

0.2 Servos and Delays

The servo module was done entirely from scratch. We had wanted to take the code from online resources to run our servos and solve the cube, but we were using different FPGAs and did not have enough to time figure out how the code worked or how to implement it properly in Nios or Quartus. Instead, we had to figure out how to divide the clock and what code to write to create a pulse width between 1 and 2 milliseconds within a 20 millisecond pulse period. Without the time to settle on a more refined or cleaner solution, we had to settle on a crude attempt at creating and implementing delays and pulse modulations.

All code will be included in the appendix.

Results

We were able to get a rough setup of mounted claws that could hold the cube up properly a majority of the time. Sometimes during execution of moves, the claws would not align properly due to the crudeness of the way we tried to implement grip on the them, and the cube would need to be quickly realigned by hand before the next move was executed. The delays we created in the code gave us a good bit of time to keep an eye of and fix the position of the cube before the next move. The delays were not created to be efficient and reduce the time between moves. Rather, they were designed as very rough and long delays before we even realized that the extra time was needed to keep the cube and claws in the proper places.

The keypad and decoding modules worked perfectly, as they were taken from successful labs at the beginning of the course and did not require much modification. The keypresses would successfully be read by the decoding module and the input would be sent to the servo module that would run a sequence of commands depending on the input.

Looking back at the original goals, we were unable to create an automatic solver. The complexity of a lot of the required code and implementation was too much for us at this level, and we did not have the time to put effort into delving into it.

Issues

During the design process, we ran into many issues that we definitely did not expect to encounter. One of the biggest problems was attempting to write a module that would reliably power and turn the servos. In the beginning we were experimenting with different clocks and counters, and were likely overthinking the solution for quite a while. The end product in our code utilizes just one clock to count for the servo pulses and delay counters, which was also another problem we had. Initially, we were struggling to come up with a way that would delay multiple moves so that we could rotate and open/close the claws in different sequences. Our final version ended up utilizing the same divided clock as the one used for the servos, and it would increment a delay counter which would initiate a move after a specified count is reached, imitating a crude delay setup.

Another big issue outside of the program and code was trying to hold the cube properly with the three claws. During our demonstration, it could be seen that the claws would often misalign and the cube would slip and almost fall out. A possible solution to this would be to mount all three claws onto the same flat platform so that they are all on even ground. It may also be a problem of the claws, as we didn't have an even way of grasping the cube. We superglued makeshift brackets onto the tips of the claws provide a way crude way for them to grip the cube. It would've been better if something with more friction was used instead of metal brackets that slipped easily.

What we've learned

During the design process of this project, we've learned not to be too ambitious at the onset of the project. We attempted to create a very advanced and involved solution without analyzing how much time it would take to implement everything. As the project continued, we had to continue to cut down on the number of things we wanted to do because we did not expect how much time or effort it would to complete the task. We've also learned how to interface with the FPGA pins and internally connect or set different parameters within the FPGA, and different ways to implement counters and delay functions within the syntax of System Verilog. We've realized how important timers can be in a system, and important it can be for modules to be able to send and receive signals from each other. For future projects, we will definitely attempt to create more achievable goals and utilize all the lessons we've learned during this project to avoid hitting similar obstacles.

References

https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/s2015/akw62_rq35_sp2283/akw62_rq35_sp2283/index.html - Reference code, hardware, setup from another group of students' project.

http://www.ti.com/lit/ug/tidu737/tidu737.pdf - Texas Instruments DE-0 Nano user manual.

Appendix

Column Sequencer

```
module colseq (
      input logic [3:0] kpr,
      input logic clk, reset_n,
      output logic[3:0] kpc);

      logic [3:0] kpc_next;

      always_comb begin

            if(!reset_n)
                  kpc_next = 4'b0111;

            else if(kpr != 4'b1111)
                  kpc_next = kpc;

            else
                  unique case(kpc)
                        4'b0111: kpc_next = 4'b1011;
                        4'b1011: kpc_next = 4'b1101;
                        4'b1101: kpc_next = 4'b1110;
                        4'b1110: kpc_next = 4'b0111;
                        default: kpc_next = 4'b1011;
                  endcase
      end

      always_ff@(posedge clk) begin
            kpc <= kpc_next;
      end


endmodule
```

Keypad Decoder

```
module kpdecode   (input logic [3:0] kpc,
                        input logic [3:0] kpr,
                        input logic reset_n,
                        input logic clk,
                        output logic kphit,
                        output logic [3:0] num,
                        output logic [3:0] num_const,
                        output logic [29:0] delay);

      always_comb begin
            // Selects the correct LED output
            // for a given row and column input
            unique case (kpr) // Active low
                  4'b1110:    unique case (kpc) // Active low
                                          4'b1110: num = 13;
                                          4'b1101: num = 15;
                                          4'b1011: num = 0;
                                          4'b0111: num = 14;
                                          default: num = 4'hx;
                                    endcase
```

```
                4'b1101:    unique case (kpc) // Active low
                                    4'b1110: num = 12;
                                    4'b1101: num = 9;
                                    4'b1011: num = 8;
                                    4'b0111: num = 7;
                                    default: num = 4'hx;
                            endcase

                4'b1011:    unique case (kpc) // Active low
                                    4'b1110: num = 11;
                                    4'b1101: num = 6;
                                    4'b1011: num = 5;
                                    4'b0111: num = 4;
                                    default: num = 4'hx;
                            endcase

                4'b0111:    unique case (kpc) // Active low
                                    4'b1110: num = 10;
                                    4'b1101: num = 3;
                                    4'b1011: num = 2;
                                    4'b0111: num = 1;
                                    default: num = 4'hx;
                            endcase

                default:    num = 4'hx;
        endcase

        // Modifies kbhit based on if a key is
        // being pressed or not
        if (kpr == 4'b1111)
            kphit = 0;
        else
            kphit = 1;

    end

    always @(posedge clk) begin
        if (kphit) begin
            num_const <= num;
        end else begin
            num_const <= num_const;
        end
    end

endmodule
```

Servo

```
module servo(input logic clk,
                        input logic RxD,
                        output RCServo_BP,
                        output RCServo_BR,
                        output RCServo_CP,
                        output RCServo_CR,
                        output RCServo_DP,
                        output RCServo_DR,
                        input logic [3:0] num_const,
                        output logic [7:0] leds,
                        input logic kphit
                        );


parameter N = 40;
parameter DELAY_1 = 80;
parameter DELAY_2 = 100;

// Claw B
parameter WBP_OPEN = 5;
parameter WBR_REST = 5;
parameter WBP_CLOSE = 3;
parameter WBR_TURN = 3;
// Claw C
parameter WCP_OPEN = 3;
parameter WCR_REST = 2;
parameter WCP_CLOSE = 8;
parameter WCR_TURN = 4;
// Claw D
parameter WDP_OPEN = 2;
parameter WDR_REST = 1;
parameter WDP_CLOSE = 5;
parameter WDR_TURN = 3;


logic [19:0]      count;
logic [19:0]      count_next;

logic [29:0]      DELAY = 0;
logic [29:0]      delay_next;
logic [29:0]      delay2 = 0;
logic [29:0]      delay_next2;
logic [29:0]      delay3 = 0;
logic [29:0]      delay_next3;
logic [29:0]      delay4 = 0;
logic [29:0]      delay_next4;
logic [29:0]      delay5 = 0;
logic [29:0]      delay_next5;
logic [29:0]      delay6 = 0;
logic [29:0]      delay_next6;
logic [29:0]      delay7 = 0;
logic [29:0]      delay_next7;
logic [29:0]      delay8 = 0;
logic [29:0]      delay_next8;
logic [29:0]      delay9 = 0;
logic [29:0]      delay_next9;
logic [29:0]      delay10 = 0;
logic [29:0]      delay_next10;
```

```
logic [29:0]       delay11 = 0;
logic [29:0]       delay_next11;

logic [1:0]        RCServo_BR_NEXT;
logic [1:0]        RCServo_BP_NEXT;
logic [1:0]        RCServo_CR_NEXT;
logic [1:0]        RCServo_CP_NEXT;
logic [1:0]        RCServo_DR_NEXT;
logic [1:0]        RCServo_DP_NEXT;

always_comb begin

      count_next = count;
      // DELAY
      delay_next = DELAY;
      delay_next2 = delay2;
      delay_next3 = delay3;
      delay_next4 = delay4;
      delay_next5 = delay5;
      delay_next6 = delay6;
      delay_next7 = delay7;
      delay_next8 = delay8;
      delay_next9 = delay9;
      delay_next10 = delay10;
      delay_next11 = delay11;

      // CLAW B
      RCServo_BR_NEXT=RCServo_BR;
      RCServo_BP_NEXT=RCServo_BP;
      // CLAW C
      RCServo_CR_NEXT=RCServo_CR;
      RCServo_CP_NEXT =RCServo_CP;
      // CLAW D
      RCServo_DR_NEXT=RCServo_DR;
      RCServo_DP_NEXT =RCServo_DP;


      if (count == N - 1) begin
            count_next = 0;
      end
      else
            count_next = count + 1;

      unique case (num_const)
////////////////////////////////////////////////////////////////////////////////
///////
      1:

      begin
      // TURN CLAW B TO REST POSITION
      if (count < WBR_REST)
            RCServo_BR_NEXT = 1; // PULSE HIGH
      else
            RCServo_BR_NEXT = 0; // PULSE LOW

      // TURN CLAW B OPEN
      if (count < WBP_OPEN)
            RCServo_BP_NEXT = 1; // PULSE HIGH
      else
            RCServo_BP_NEXT = 0; // PULSE LOW
```

```
        // TURN CLAW C TO REST POSITION
        if (count < WCR_REST)
                RCServo_CR_NEXT = 1; // PULSE HIGH
        else
                RCServo_CR_NEXT = 0; // PULSE LOW

        // TURN CLAW C OPEN
        if (count < WCP_OPEN)
                RCServo_CP_NEXT = 1; // PULSE HIGH
        else
                RCServo_CP_NEXT = 0; // PULSE LOW

        // TURN CLAW D OPEN
        if (count < WDP_OPEN)
                RCServo_DP_NEXT = 1; // PULSE HIGH
        else
                RCServo_DP_NEXT = 0; // PULSE LOW

        // TURN CLAW D REST
        if (count < WDR_REST)
                RCServo_DR_NEXT = 1; // PULSE HIGH
        else
                RCServo_DR_NEXT = 0; // PULSE LOW
        end
////////////////////////////////////////////////////////////////////////////////
///////
        2:
        begin

        if (count == N - 1) begin
                delay_next = DELAY + 1;

        end
        // TURN CLAW B TO REST POSITION
        if (count < WBR_REST)
                RCServo_BR_NEXT = 1; // PULSE HIGH
        else
                RCServo_BR_NEXT = 0; // PULSE LOW

        // CLAW B CLOSE
        if (DELAY > DELAY_2)
        begin
        if (count < WBP_CLOSE)
                RCServo_BP_NEXT = 1; // PULSE HIGH
        else
                RCServo_BP_NEXT = 0; // PULSE LOW
        end

        // TURN CLAW C TO REST POSITION
        if (count < WCR_REST)
                RCServo_CR_NEXT = 1; // PULSE HIGH
        else
                RCServo_CR_NEXT = 0; // PULSE LOW

        // TURN CLAW C CLOSE
        if (DELAY > 2*DELAY_2)
        begin
        if (count < WCP_CLOSE)
                RCServo_CP_NEXT = 1; // PULSE HIGH
```

```
	else
		RCServo_CP_NEXT = 0; // PULSE LOW
	end

	// TURN CLAW D CLOSE
	if (DELAY > 4*DELAY_2)
	begin
	if (count < WDP_CLOSE)
		RCServo_DP_NEXT = 1; // PULSE HIGH
	else
		RCServo_DP_NEXT = 0; // PULSE LOW
	end

	// TURN CLAW D REST
	if (count < WDR_REST)
		RCServo_DR_NEXT = 1; // PULSE HIGH
	else
		RCServo_DR_NEXT = 0; // PULSE LOW
	end
////////////////////////////////////////////////////////////////////////////////
///////
	3: // CLAW B TURN
	begin
		if (count == N - 1) begin
		delay_next2 = delay2 + 1;

	end
		// TURN CLAW B TO REST POSITION
		if (count < WBR_REST)
			RCServo_BR_NEXT = 1; // PULSE HIGH
		else
			RCServo_BR_NEXT = 0; // PULSE LOW

		// CLAW B CLOSE
		if (count < WBP_CLOSE)
			RCServo_BP_NEXT = 1; // PULSE HIGH
		else
			RCServo_BP_NEXT = 0; // PULSE LOW

		// TURN CLAW C TO REST POSITION
		if (count < WCR_REST)
			RCServo_CR_NEXT = 1; // PULSE HIGH
		else
			RCServo_CR_NEXT = 0; // PULSE LOW

		// TURN CLAW C CLOSE
		if (count < WCP_CLOSE)
			RCServo_CP_NEXT = 1; // PULSE HIGH
		else
			RCServo_CP_NEXT = 0; // PULSE LOW

		// TURN CLAW D CLOSE
		if (count < WDP_CLOSE)
			RCServo_DP_NEXT = 1; // PULSE HIGH
		else
			RCServo_DP_NEXT = 0; // PULSE LOW

		// TURN CLAW D REST
		if (count < WDR_REST)
			RCServo_DR_NEXT = 1; // PULSE HIGH
```

```
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW
////////////////////////////////////////////////////////////////////////////////
///////

            if (delay2 > DELAY_1)
            begin
            if (count < WBR_TURN)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end

            if (delay2 > 2*DELAY_1)
            begin
            if (count < WBP_OPEN)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            if (delay2 > 4*DELAY_1)
            begin
            if (count < WBR_REST)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end

            if (delay2 > 6*DELAY_1)
            begin
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end


      end

      4: // CLAW B TURN'
      begin
            if (count == N - 1)
                  delay_next3 = delay3 + 1;

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
```

```
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW
////////////////////////////////////////////////////////////////////////////////
///////
            if (delay3 > DELAY_1)
            begin
            if (count < WBP_OPEN)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            if (delay3 > 2*DELAY_1)
            begin
            if (count < WBR_TURN)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end

            if (delay3 > 4*DELAY_1)
            begin
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            if (delay3 > 6*DELAY_1)
            begin
            if (count < WBR_REST)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end
      end
      ////////////////////////////////////////////////////////////////////////////////
//////////////
      5: // CLAW C TURN
      begin

            // START DELAY TIMER
```

```
            if (count == N - 1)
            delay_next4 = delay4 + 1;

////////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//          CLAW C --> ROTATE - OPEN - ROTATE
////////////////////////////////////////////////////////////////////////////////
///////

            if (delay4 > DELAY_1)
            begin
            if (count < WCR_TURN)
                  RCServo_CR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CR_NEXT  = 0; //  PULSE LOW
            end

            if (delay4 > 2*DELAY_1)
```

```
            begin
            if (count < WCP_OPEN)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

            if (delay4 > 4*DELAY_1)
            begin
            if (count < WCR_REST)
                  RCServo_CR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CR_NEXT  = 0; //  PULSE LOW
            end

            if (delay4 > 6*DELAY_1)
            begin
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

      end // END CASE 5 (CLAW C TURN)

      ////////////////////////////////////////////////////////////////////////////
//////////////
      6: // CLAW C TURN'
      begin

////////////////////////////////////////////////////////////////////////////
///////
//          START DELAY TIMER
////////////////////////////////////////////////////////////////////////////
///////
            if (count == N - 1)
                  delay_next5 = delay5 + 1;

////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
```

```
                    RCServo_CR_NEXT = 0; // PULSE LOW

              // TURN CLAW C CLOSE
              if (count < WCP_CLOSE)
                    RCServo_CP_NEXT = 1; // PULSE HIGH
              else
                    RCServo_CP_NEXT = 0; // PULSE LOW

              // TURN CLAW D CLOSE
              if (count < WDP_CLOSE)
                    RCServo_DP_NEXT = 1; // PULSE HIGH
              else
                    RCServo_DP_NEXT = 0; // PULSE LOW

              // TURN CLAW D REST
              if (count < WDR_REST)
                    RCServo_DR_NEXT = 1; // PULSE HIGH
              else
                    RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//            CLAW C --> OPEN - ROTATE - PINCH - ROTATE
////////////////////////////////////////////////////////////////////////////////
///////

              // OPEN CLAW C
              if (delay5 > DELAY_1)
              begin
              if (count < WCP_OPEN)
                    RCServo_CP_NEXT  = 1; //  PULSE HIGH
              else
                    RCServo_CP_NEXT  = 0; //  PULSE LOW
              end

              // ROTATE CLAW C
              if (delay5 > 2*DELAY_1)
              begin
              if (count < WCR_TURN)
                    RCServo_CR_NEXT  = 1; //  PULSE HIGH
              else
                    RCServo_CR_NEXT  = 0; //  PULSE LOW
              end

              // CLOSE CLAW C
              if (delay5 > 4*DELAY_1)
              begin
              if (count < WCP_CLOSE)
                    RCServo_CP_NEXT  = 1; //  PULSE HIGH
              else
                    RCServo_CP_NEXT  = 0; //  PULSE LOW
              end

              // ROTATE CLAW C
              if (delay5 > 6*DELAY_1)
              begin
              if (count < WCR_REST)
                    RCServo_CR_NEXT  = 1; //  PULSE HIGH
              else
                    RCServo_CR_NEXT  = 0; //  PULSE LOW
```

```
            end
      end // END CASE 6 (CLAW C TURN')

      ////////////////////////////////////////////////////////////////////////////
//////////////
      7: // CLAW D TURN
      begin

            // START DELAY TIMER
            if (count == N - 1)
            delay_next6 = delay6 + 1;

////////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//          CLAW D --> ROTATE - OPEN - ROTATE
////////////////////////////////////////////////////////////////////////////////
///////
```

```
            if (delay6 > DELAY_1)
            begin
            if (count < WDR_TURN)
                  RCServo_DR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DR_NEXT  = 0; //  PULSE LOW
            end

            if (delay6 > 2*DELAY_1)
            begin
            if (count < WDP_OPEN)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            if (delay6 > 4*DELAY_1)
            begin
            if (count < WDR_REST)
                  RCServo_DR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DR_NEXT  = 0; //  PULSE LOW
            end

            if (delay6 > 6*DELAY_1)
            begin
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

      end // END CASE 7 (CLAW D TURN)

      ////////////////////////////////////////////////////////////////////////////
//////////////
      8: // CLAW D TURN'
      begin

////////////////////////////////////////////////////////////////////////////
///////
//          START DELAY TIMER
////////////////////////////////////////////////////////////////////////////
///////

            if (count == N - 1)
                  delay_next7 = delay7 + 1;

////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW
```

```
            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//          CLAW D --> OPEN - ROTATE - PINCH - ROTATE
////////////////////////////////////////////////////////////////////////////////
///////

            // OPEN CLAW D
            if (delay7 > DELAY_1)
            begin
            if (count < WDP_OPEN)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW D
            if (delay7 > 2*DELAY_1)
            begin
            if (count < WDR_TURN)
                  RCServo_DR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW C
            if (delay7 > 4*DELAY_1)
            begin
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
```

```
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW C
            if (delay7 > 6*DELAY_1)
            begin
            if (count < WDR_REST)
                  RCServo_DR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DR_NEXT  = 0; //  PULSE LOW
            end
      end // END CASE 8 (CLAW D TURN')

      ////////////////////////////////////////////////////////////////////////////
//////////////

      9: // ROTATE CUBE
      begin

////////////////////////////////////////////////////////////////////////////
///////
//          START DELAY TIMER
////////////////////////////////////////////////////////////////////////////
///////

            if (count == N - 1)
                  delay_next8 = delay8 + 1;

////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
```

```
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//          CLAW C+D --> ROTATE
////////////////////////////////////////////////////////////////////////////////
///////

            // OPEN CLAW D
            if (delay8 > DELAY_1)
            begin
            if (count < WDP_OPEN)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW C + B
            if (delay8 > 2*DELAY_1)
            begin
            if (count < WBR_TURN)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW

            if (count < WCR_TURN)
                  RCServo_CR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW D
            if (delay8 > 3*DELAY_1)
            begin
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            // OPEN CLAW B
            if (delay8 > 4*DELAY_1)
            begin
            if (count < WBP_OPEN)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW B
            if (delay8 > 6*DELAY_1)
            begin
```

```
            if (count < WBR_REST)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW B
            if (delay8 > 8*DELAY_1)
            begin
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            // OPEN CLAW C
            if (delay8 > 10*DELAY_1)
            begin
            if (count < WCP_OPEN)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW C
            if (delay8 > 12*DELAY_1)
            begin
            if (count < WCR_REST)
                  RCServo_CR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW C
            if (delay8 > 14*DELAY_1)
            begin
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

      end // END CASE 8 (CLAW D TURN')


////////////////////////////////////////////////////////////////////////////////
///////

      10: // ROTATE' CUBE
      begin

////////////////////////////////////////////////////////////////////////////////
///////
//          START DELAY TIMER
////////////////////////////////////////////////////////////////////////////////
///////

            if (count == N - 1)
                  delay_next9 = delay9 + 1;
```

```
////////////////////////////////////////////////////////////////////////////////
///////
//          INITIALIZE CLAW A, B, C, D
////////////////////////////////////////////////////////////////////////////////
///////

            // TURN CLAW B TO REST POSITION
            if (count < WBR_REST)
                  RCServo_BR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BR_NEXT = 0; // PULSE LOW

            // CLAW B CLOSE
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_BP_NEXT = 0; // PULSE LOW

            // TURN CLAW C TO REST POSITION
            if (count < WCR_REST)
                  RCServo_CR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CR_NEXT = 0; // PULSE LOW

            // TURN CLAW C CLOSE
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_CP_NEXT = 0; // PULSE LOW

            // TURN CLAW D CLOSE
            if (count < WDP_CLOSE)
                  RCServo_DP_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DP_NEXT = 0; // PULSE LOW

            // TURN CLAW D REST
            if (count < WDR_REST)
                  RCServo_DR_NEXT = 1; // PULSE HIGH
            else
                  RCServo_DR_NEXT = 0; // PULSE LOW

////////////////////////////////////////////////////////////////////////////////
///////
//          CLAW C+D --> ROTATE
////////////////////////////////////////////////////////////////////////////////
///////

            // OPEN CLAW B
            if (delay9 > DELAY_1)
            begin
            if (count < WBP_OPEN)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW B
            if (delay9 > 2*DELAY_1)
            begin
```

```
            if (count < WBR_TURN)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW B
            if (delay9 > 3*DELAY_1)
            begin
            if (count < WBP_CLOSE)
                  RCServo_BP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BP_NEXT  = 0; //  PULSE LOW
            end

            // OPEN CLAW C
            if (delay9 > 4*DELAY_1)
            begin
            if (count < WCP_OPEN)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW C
            if (delay9 > 5*DELAY_1)
            begin
            if (count < WCR_TURN)
                  RCServo_CR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CR_NEXT  = 0; //  PULSE LOW
            end

            // CLOSE CLAW C
            if (delay9 > 6*DELAY_1)
            begin
            if (count < WCP_CLOSE)
                  RCServo_CP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_CP_NEXT  = 0; //  PULSE LOW
            end

            // OPEN CLAW D
            if (delay9 > 7*DELAY_1)
            begin
            if (count < WDP_OPEN)
                  RCServo_DP_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_DP_NEXT  = 0; //  PULSE LOW
            end

            // ROTATE CLAW B + c
            if (delay9 > 8*DELAY_1)
            begin
            if (count < WBR_REST)
                  RCServo_BR_NEXT  = 1; //  PULSE HIGH
            else
                  RCServo_BR_NEXT  = 0; //  PULSE LOW

            if (count < WCR_REST)
```

```
                RCServo_CR_NEXT  = 1; //  PULSE HIGH
        else
                RCServo_CR_NEXT  = 0; //  PULSE LOW
        end

        // CLOSE CLAW D
        if (delay9 > 9*DELAY_1)
        begin
        if (count < WDP_CLOSE)
                RCServo_DP_NEXT  = 1; //  PULSE HIGH
        else
                RCServo_DP_NEXT  = 0; //  PULSE LOW
        end

end // END CASE 10 (ROATE')

default:
begin

// TURN CLAW B TO REST POSITION
if (count < WBR_TURN)
      RCServo_BR_NEXT = 1; // PULSE HIGH
else
      RCServo_BR_NEXT = 0; // PULSE LOW

// TURN CLAW B OPEN
if (count < WBP_OPEN)
      RCServo_BP_NEXT = 1; // PULSE HIGH
else
      RCServo_BP_NEXT = 0; // PULSE LOW

// TURN CLAW C TO REST POSITION
if (count < WCR_TURN)
      RCServo_CR_NEXT = 1; // PULSE HIGH
else
      RCServo_CR_NEXT = 0; // PULSE LOW

// TURN CLAW C OPEN
if (count < WCP_OPEN)
      RCServo_CP_NEXT = 1; // PULSE HIGH
else
      RCServo_CP_NEXT = 0; // PULSE LOW

// TURN CLAW D OPEN
if (count < WDP_OPEN)
      RCServo_DP_NEXT = 1; // PULSE HIGH
else
      RCServo_DP_NEXT = 0; // PULSE LOW

// TURN CLAW D REST
if (count < WDR_TURN)
      RCServo_DR_NEXT = 1; // PULSE HIGH
else
      RCServo_DR_NEXT = 0; // PULSE LOW
end



endcase
```

```
end


always@(posedge clk)
begin

	count <= count_next;
	DELAY <= delay_next;
	delay2 <= delay_next2;
	delay3 <= delay_next3;
	delay4 <= delay_next4;
	delay5 <= delay_next5;
	delay6 <= delay_next6;
	delay7 <= delay_next7;
	delay8 <= delay_next8;
	delay9 <= delay_next9;
	delay10 <= delay_next10;
	delay11 <= delay_next11;
	RCServo_BR <= RCServo_BR_NEXT;
	RCServo_BP <= RCServo_BP_NEXT;
	RCServo_CR <= RCServo_CR_NEXT;
	RCServo_CP <= RCServo_CP_NEXT;
	RCServo_DR <= RCServo_DR_NEXT;
	RCServo_DP <= RCServo_DP_NEXT;

	// RESET DELAY IF NEW BUTTON PRESSED
	if (kphit) begin
	DELAY <= 0;
	delay2 <= 0;
	delay3 <= 0;
	delay4 <= 0;
	delay5 <= 0;
	delay6 <= 0;
	delay7 <= 0;
	delay8 <= 0;
	delay9 <= 0;
	delay10 <= 0;
	delay11 <= 0;
	end

end


endmodule
```

Test

```
module test ( input logic clk_50,
                  (* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
                        input logic RxD,
                        output RCServo_CP,
                        output RCServo_CR,
                        output RCServo_BP,
                        output RCServo_BR,
                        output RCServo_DP,
                        output RCServo_DR,
                        (* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
                        input logic [3:0] kpr,
                        output logic[3:0] kpc,
                        output logic [3:0] num,
                        output logic kphit,
                        output logic clk,
                        output logic [3:0] num_const,
                        output logic [7:0] leds,
                        output logic [7:0] LED,
                        output logic [29:0] delay
                        );

            //logic clk ;                // 2kHz clock for keypad scanning
            //logic RxD;
      logic reset_n = 1;
      pll pll0 ( .inclk0(clk_50), .c0(clk) ) ;
      servo servo1(.clk, .RxD, .RCServo_CP, .RCServo_CR, .RCServo_BP, .RCServo_BR,
.RCServo_DP, .RCServo_DR, .num_const, .leds, .kphit);
      colseq colseq1(kpr, clk, reset_n, kpc);
      kpdecode kpdecode1(kpc, kpr, reset_n, clk, kphit, num, num_const, delay);

//    assign LED = {num_const == 1,num_const == 2,num_const == 3,num_const ==
4,kphit,3'b000};

endmodule

// megafunction wizard: %ALTPLL%
// ...
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ...

module pll ( inclk0, c0);

            input     inclk0;
        output    c0;

            wire [0:0] sub_wire2 = 1'h0;
        wire [4:0] sub_wire3;
        wire  sub_wire0 = inclk0;
        wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
        wire [0:0] sub_wire4 = sub_wire3[0:0];
        wire  c0 = sub_wire4;

        altpll altpll_component ( .inclk (sub_wire1), .clk
          (sub_wire3), .activeclock (), .areset (1'b0), .clkbad
          (), .clkena ({6{1'b1}}), .clkloss (), .clkswitch
          (1'b0), .configupdate (1'b0), .enable0 (), .enable1 (),
          .extclk (), .extclkena ({4{1'b1}}), .fbin (1'b1),
          .fbmimicbidir (), .fbout (), .fref (), .icdrclk (),
```

```
        .locked (), .pfdena (1'b1), .phasecounterselect
        ({4{1'b1}}), .phasedone (), .phasestep (1'b1),
        .phaseupdown (1'b1), .pllena (1'b1), .scanaclr (1'b0),
        .scanclk (1'b0), .scanclkena (1'b1), .scandata (1'b0),
        .scandataout (), .scandone (), .scanread (1'b0),
        .scanwrite (1'b0), .sclkout0 (), .sclkout1 (),
        .vcooverrange (), .vcounderrange ());

    defparam
            altpll_component.bandwidth_type = "AUTO",
            altpll_component.clk0_divide_by = 25000,
            altpll_component.clk0_duty_cycle = 50,
            altpll_component.clk0_multiply_by = 1,
            altpll_component.clk0_phase_shift = "0",
            altpll_component.compensate_clock = "CLK0",
            altpll_component.inclk0_input_frequency = 20000,
            altpll_component.intended_device_family = "Cyclone IV E",
            altpll_component.lpm_hint = "CBX_MODULE_PREFIX=lab1clk",
            altpll_component.lpm_type = "altpll",
            altpll_component.operation_mode = "NORMAL",
            altpll_component.pll_type = "AUTO",
            altpll_component.port_activeclock = "PORT_UNUSED",
            altpll_component.port_areset = "PORT_UNUSED",
            altpll_component.port_clkbad0 = "PORT_UNUSED",
            altpll_component.port_clkbad1 = "PORT_UNUSED",
            altpll_component.port_clkloss = "PORT_UNUSED",
            altpll_component.port_clkswitch = "PORT_UNUSED",
            altpll_component.port_configupdate = "PORT_UNUSED",
            altpll_component.port_fbin = "PORT_UNUSED",
            altpll_component.port_inclk0 = "PORT_USED",
            altpll_component.port_inclk1 = "PORT_UNUSED",
            altpll_component.port_locked = "PORT_UNUSED",
            altpll_component.port_pfdena = "PORT_UNUSED",
            altpll_component.port_phasecounterselect = "PORT_UNUSED",
            altpll_component.port_phasedone = "PORT_UNUSED",
            altpll_component.port_phasestep = "PORT_UNUSED",
            altpll_component.port_phaseupdown = "PORT_UNUSED",
            altpll_component.port_pllena = "PORT_UNUSED",
            altpll_component.port_scanaclr = "PORT_UNUSED",
            altpll_component.port_scanclk = "PORT_UNUSED",
            altpll_component.port_scanclkena = "PORT_UNUSED",
            altpll_component.port_scandata = "PORT_UNUSED",
            altpll_component.port_scandataout = "PORT_UNUSED",
            altpll_component.port_scandone = "PORT_UNUSED",
            altpll_component.port_scanread = "PORT_UNUSED",
            altpll_component.port_scanwrite = "PORT_UNUSED",
            altpll_component.port_clk0 = "PORT_USED",
            altpll_component.port_clk1 = "PORT_UNUSED",
            altpll_component.port_clk2 = "PORT_UNUSED",
            altpll_component.port_clk3 = "PORT_UNUSED",
            altpll_component.port_clk4 = "PORT_UNUSED",
            altpll_component.port_clk5 = "PORT_UNUSED",
            altpll_component.port_clkena0 = "PORT_UNUSED",
            altpll_component.port_clkena1 = "PORT_UNUSED",
            altpll_component.port_clkena2 = "PORT_UNUSED",
            altpll_component.port_clkena3 = "PORT_UNUSED",
            altpll_component.port_clkena4 = "PORT_UNUSED",
            altpll_component.port_clkena5 = "PORT_UNUSED",
            altpll_component.port_extclk0 = "PORT_UNUSED",
            altpll_component.port_extclk1 = "PORT_UNUSED",
```

```
                altpll_component.port_extclk2 = "PORT_UNUSED",
                altpll_component.port_extclk3 = "PORT_UNUSED",
                altpll_component.width_clock = 5;


endmodule
```