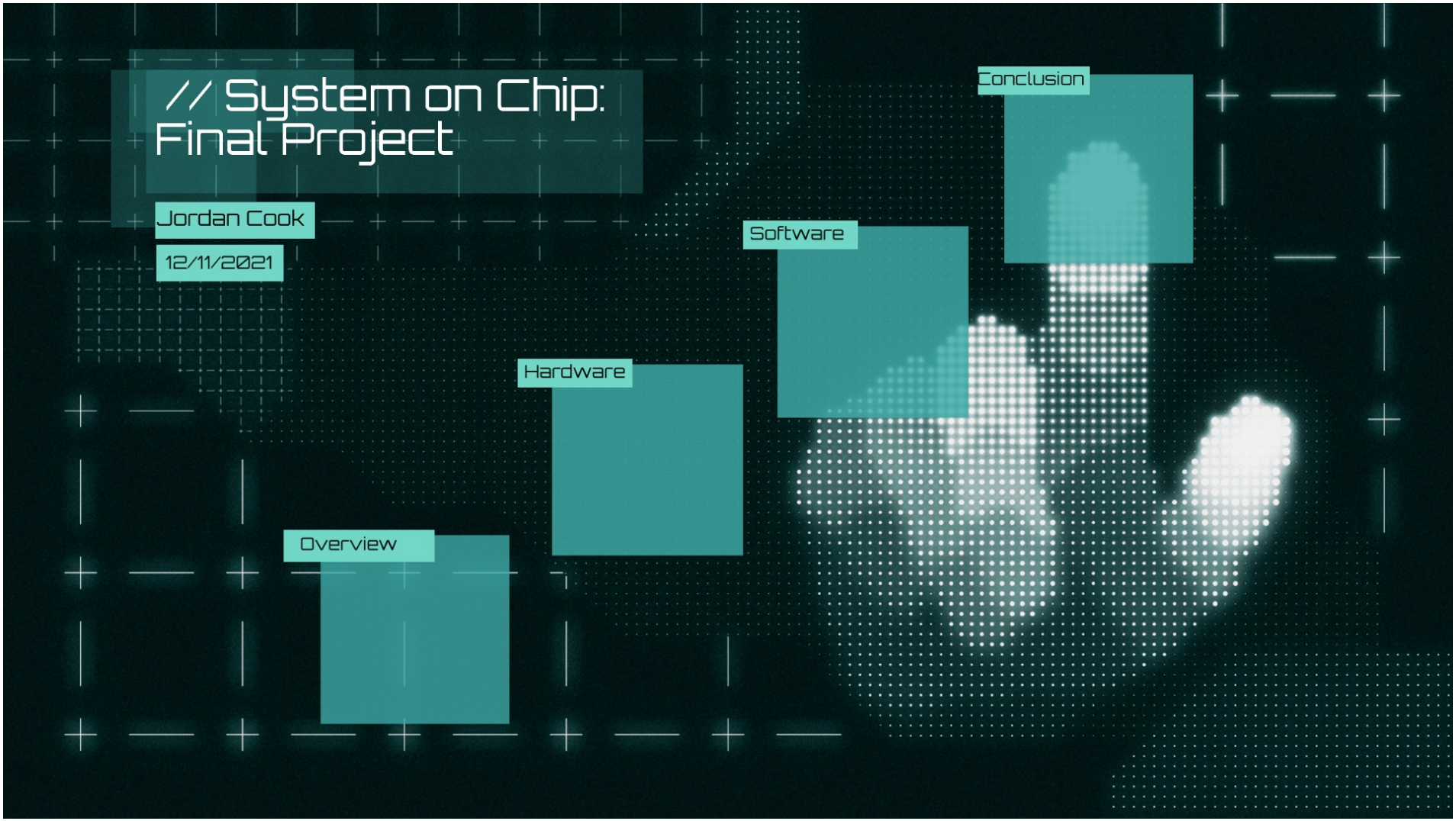

// System on Chip: Final Project
Jordan Cook
12/11/2021
Conclusion
Software
Hardware
Overview

Overview

- This project utilized a Nexys 4 DDR board to display the voltage and temperature of the chip (XIlinx Artix-7 XC7A100T-1CSG324C).
- The value would be shown on a seven-segment display as well as having indication lights
- Both hardware and software were included in this project
- Changes display based on user input

Why?

Applications

Components

Why?

- It is important to read the voltage on a chip to determine if things are functioning normally
  - If a chip is reading a voltage of 0, it indicates that it is no longer working.
  - If it reads a voltage close to the maximum (in this case 3.3V), it indicates it may be getting to high
- Temperature is important in a chip because once it gets too hot, it could damage the hardware
  - Things may get shut down as well if it overheats
- Visual representation of the data makes the system user friendly
  - If a user is not an engineer or does not know the proper values, it is important to have some sort of way to represent range

Overview

- This project utilized a Nexys 4 DDR board to display the voltage and temperature of the chip (XIlinx Artix-7 XC7A100T-1CSG324C).
- The value would be shown on a seven-segment display as well as having indication lights
- Both hardware and software were included in this project
- Changes display based on user input

Why?

Applications

Components

Applications

- Currently every CPU is equipped with hardware to measure the thermal temperature
- Anything from a cell-phone to the International Space Station reads the internal voltage and temperature to verify that the current power consumption is acceptable
- Changing lights are eye-catching and businesses everywhere use these techniques to draw in people
- Data visualization psychologically helps humans understand the values and aids in comparisons

Overview

- This project utilized a Nexys 4 DDR board to display the voltage and temperature of the chip (XIlinx Artix-7 XC7A100T-1CSG324C).
- The value would be shown on a seven-segment display as well as having indication lights
- Both hardware and software were included in this project
- Changes display based on user input

Why?

Applications

Components

Components

- Analog to Digital Converter
- Pulse Width Modulation
- Seven-Segment Display
- General Purpose Input and Output

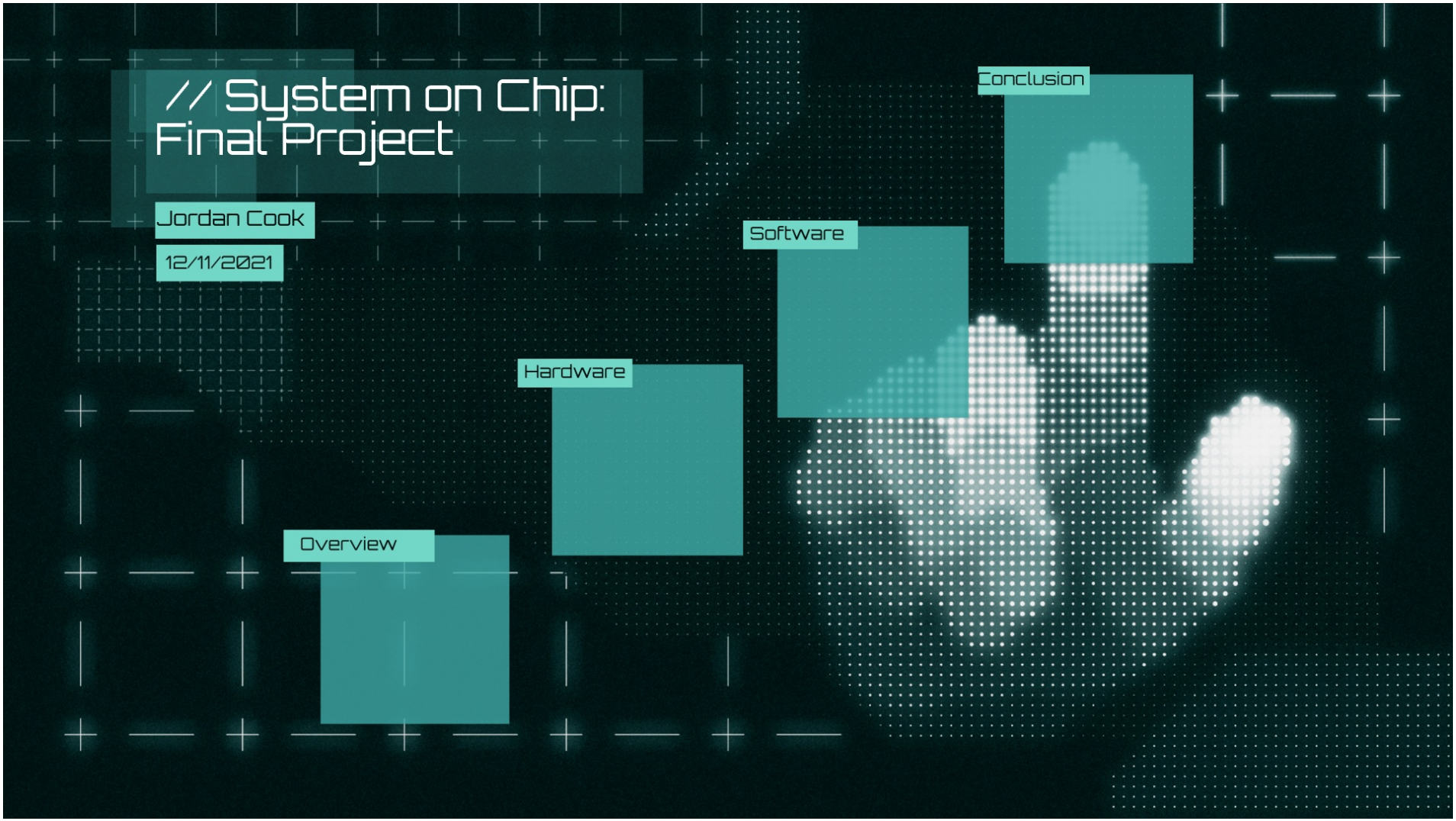
Overview

- This project utilized a Nexys 4 DDR board to display the voltage and temperature of the chip (XIlinx Artix-7 XC7A100T-1CSG324C).
- The value would be shown on a seven-segment display as well as having indication lights
- Both hardware and software were included in this project
- Changes display based on user input

Why?

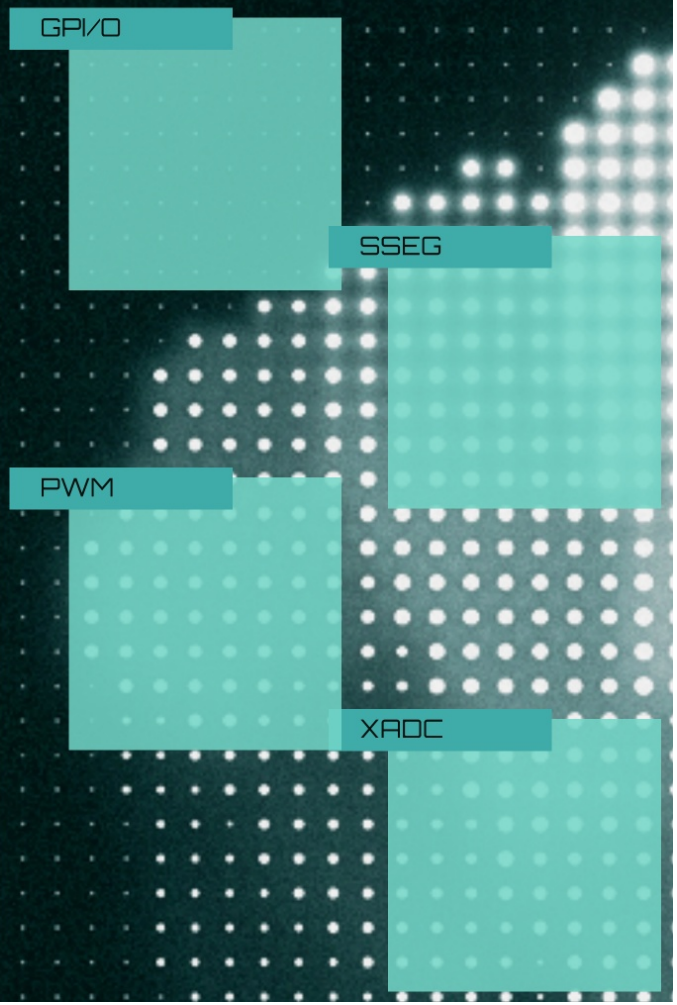
Applications

Components


// System on Chip: Final Project
Jordan Cook
12/11/2021
Overview
Hardware
Software
Conclusion

Hardware

- All modules incorporated were from Dr. Pong Chu
- A pattern was followed each time something was added
  - First, the files were imported
  - Next, an instantiation of that wrapper was made in the memory-wrapped input/output module
    - Any additional inputs and outputs were also included in the declarations
  - In the top level module, any changes that needed to be made were done
    - The added inputs or outputs were connected in the MMIO instantiation
    - The top inputs and outputs were verified to the given names in the constraints file
- Once all these steps were complete, a bit stream was generated to be exported into software



GPIO

- The hardware portion for general input / output is fairly simple; it is just writing or reading from a register
- If it is an input, the pattern of switches that are high is stored in a register
- If it is an output, the pattern of LED's that are supposed to be turned on is also read from a register

Slot Example

Input Example

Slot Example

```
(
   input logic clk,
   input  logic reset,
   // FPro bus
   input  logic mmio_cs,
   input  logic mmio_wr,
   input  logic mmio_rd,
   input  logic [20:0] mmio_addr, //
   input  logic [31:0] mmio_wr_data,
   output logic [31:0] mmio_rd_data,
   // switches and LEDs
   input logic [N_SW-1:0] sw,
   output logic [N_LED-1:0] led,
```

```
// slot 2: gpo
chu_gpo #(.W(N_LED)) gpo_slot2
(.clk(clk),
 .reset(reset),
 .cs(cs_array[`S2_LED]),
 .read(mem_rd_array[`S2_LED]),
 .write(mem_wr_array[`S2_LED]),
 .addr(reg_addr_array[`S2_LED]),
 .rd_data(rd_data_array[`S2_LED]),
 .wr_data(wr_data_array[`S2_LED]),
 .dout(led)
 );
```

GPIO

- The hardware portion for general input / output is fairly simple; it is just writing or reading from a register
- If it is an input, the pattern of switches that are high is stored in a register
- If it is an output, the pattern of LED's that are supposed to be turned on is also read from a register

Slot Example

Input Example

Input Example

```
// signal declaration
logic [W-1:0] rd_data_reg;

// body
always_ff @(posedge clk, posedge reset)
   if (reset)
      rd_data_reg <= 0;
   else
      rd_data_reg <= din;

assign rd_data[W-1:0] = rd_data_reg;
assign rd_data[31:W] = 0;
```

GPIO

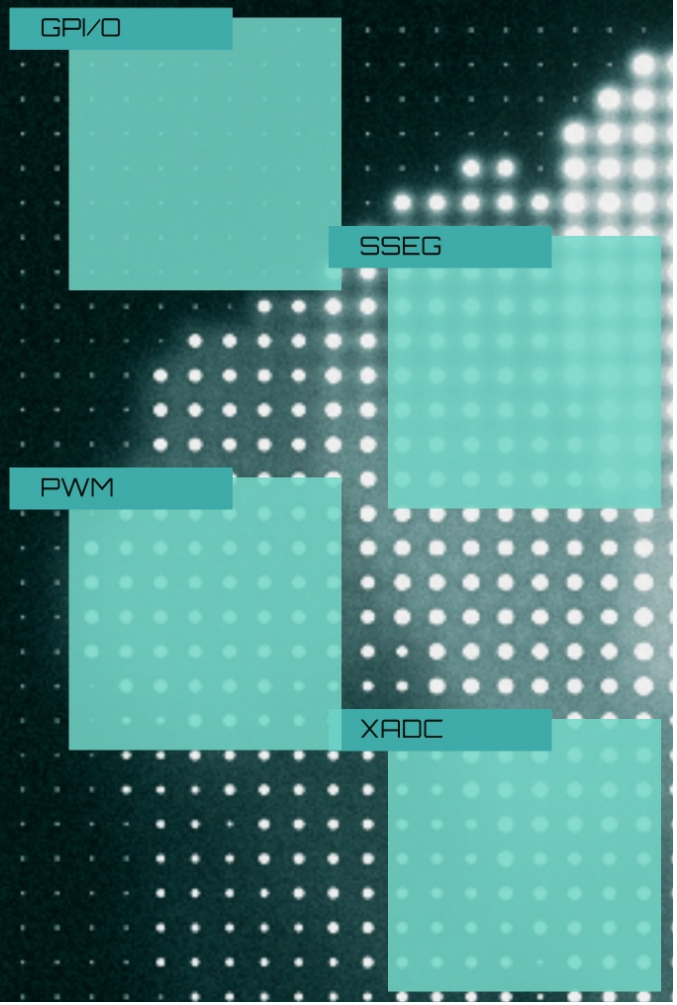
- The hardware portion for general input / output is fairly simple; it is just writing or reading from a register
- If it is an input, the pattern of switches that are high is stored in a register
- If it is an output, the pattern of LED's that are supposed to be turned on is also read from a register

Slot Example

Input Example

Hardware

- All modules incorporated were from Dr. Pong Chu
- A pattern was followed each time something was added
  - First, the files were imported
  - Next, an instantiation of that wrapper was made in the memory-wrapped input/output module
    - Any additional inputs and outputs were also included in the declarations
  - In the top level module, any changes that needed to be made were done
    - The added inputs or outputs were connected in the MMIO instantiation
    - The top inputs and outputs were verified to the given names in the constraints file
- Once all these steps were complete, a bit stream was generated to be exported into software



SSEG

- The seven-segment display was similar to the 16 LED's
  - This module read data from a register and assigned which assigned would be based based on that data
- The outputs are 'sseg', which lights up a specific segment, and 'an', which lights up a certain place (out of the eight options)

SSEG-Core

```
// declaration
logic [31:0] d0_reg, d1_reg;
logic wr_en, wr_d0, wr_d1;

// instantiate led multplexing circuit
led_mux8  led_mux8_unit (
 .clk(clk), .reset(reset),
 .in7(d1_reg[31:24]), .in6(d1_reg[23:16]),
 .in5(d1_reg[15:8]),  .in4(d1_reg[7:0]),
 .in3(d0_reg[31:24]), .in2(d0_reg[23:16]),
 .in1(d0_reg[15:8]),  .in0(d0_reg[7:0]),
 .sseg(sseg), .an(an)
);

// registers
always_ff @(posedge clk, posedge reset)
   if (reset) begin
      d0_reg <= 0;
      d1_reg <= 0;
   end
   else begin
      if (wr_d0)
         d0_reg <= wr_data;
      if (wr_d1)
         d1_reg <= wr_data;
  end
```

SSEG

- The seven-segment display was similar to the 16 LED's
  - This module read data from a register and assigned which assigned would be based based on that data
- The outputs are 'sseg', which lights up a specific segment, and 'an', which lights up a certain place (out of the eight options)

LED Mux

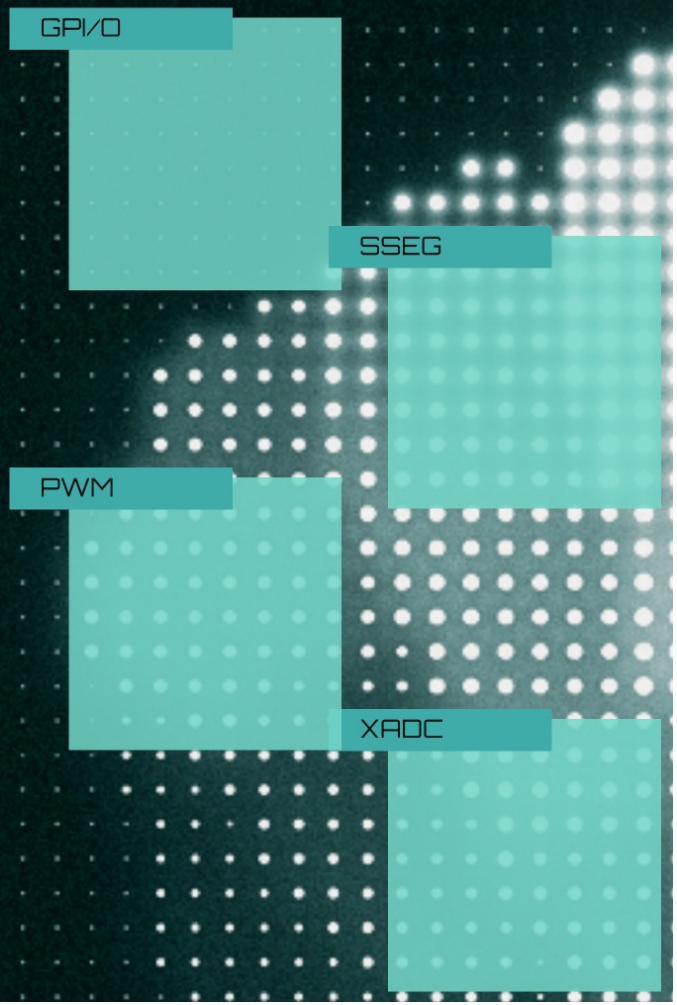
```
always_comb
   unique case (q_reg[N-1:N-3])
      3'b000: begin
         an = 8'b1111_1110;
         sseg = in0;
      end
      3'b001: begin
         an = 8'b1111_1101;
         sseg = in1;
      end
      3'b010: begin
         an = 8'b1111_1011;
         sseg = in2;
      end
      3'b011: begin
         an = 8'b1111_0111;
         sseg = in3;
      end
      3'b100: begin
         an = 8'b1110_1111;
         sseg = in4;
      end
      3'b101: begin
         an = 8'b1101_1111;
         sseg = in5;
```

SSEG

- The seven-segment display was similar to the 16 LED's
  - This module read data from a register and assigned which assigned would be based based on that data
- The outputs are 'sseg', which lights up a specific segment, and 'an', which lights up a certain place (out of the eight options)

Hardware

- All modules incorporated were from Dr. Pong Chu
- A pattern was followed each time something was added
  - First, the files were imported
  - Next, an instantiation of that wrapper was made in the memory-wrapped input/output module
    - Any additional inputs and outputs were also included in the declarations
  - In the top level module, any changes that needed to be made were done
    - The added inputs or outputs were connected in the MMIO instantiation
    - The top inputs and outputs were verified to the given names in the constraints file
- Once all these steps were complete, a bit stream was generated to be exported into software



PWM

- The concept of PWM is having a signal high for a percentage of time to simulate an analog response within digital system
-
- The 8 bit PWM used in this hardware allows the output to be a fraction of the 256 resolution
-
- This was specifically used with the RGB lights, but the PWM output could work with any of the LED's

Multi-Bit PWM

Multi-Bit PWM

```
always_ff @(posedge clk, posedge reset)
   if (reset) begin
      q_reg <= 0;
      d_reg <= 0;
      pwm_reg <= 0;
   end
   else begin
      q_reg <= q_next;
      d_reg <= d_next;
      pwm_reg <= pwm_next;
  end
// "prescale" counter
assign q_next = (q_reg==dvsr_reg) ? 0 : q_reg + 1;
assign tick = q_reg==0;
// duty cycle counter
assign d_next = (tick) ? d_reg + 1 : d_reg;
assign d_ext = {1'b0, d_reg};
// comparison circuit
generate
  genvar i;
  for (i=0; i<W; i=i+1) begin
    assign pwm_next[i] = d_ext < duty_2d_reg[i];
  end
endgenerate
assign pwm_out = pwm_reg;
// read data not used
assign rd_data = 32'b0 ;
```

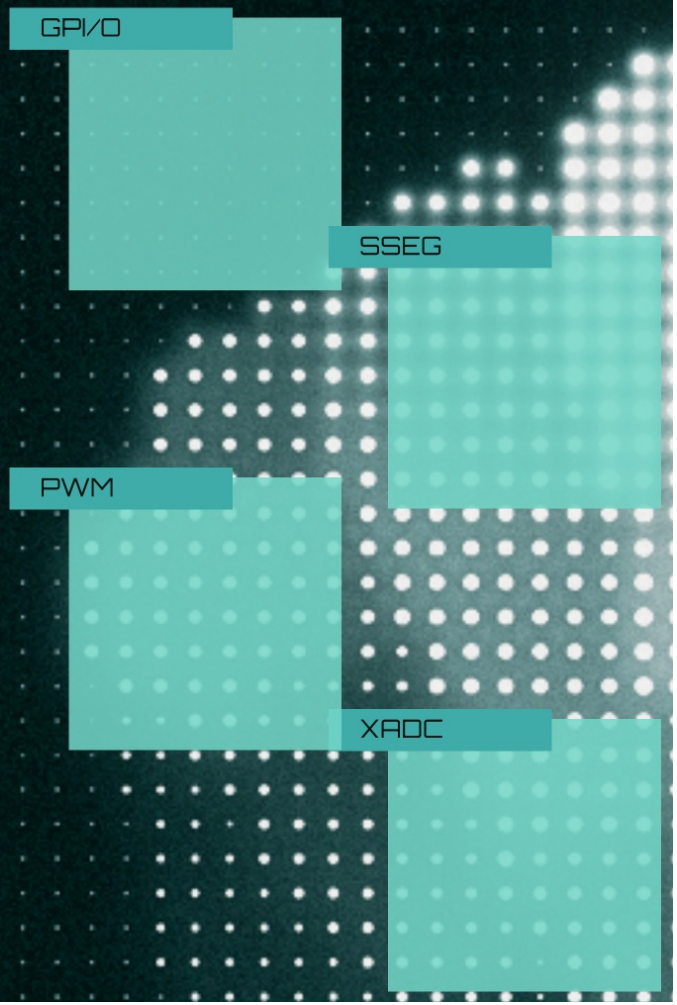
PWM

- The concept of PWM is having a signal high for a percentage of time to simulate an analog response within digital system
-
- The 8 bit PWM used in this hardware allows the output to be a fraction of the 256 resolution
-
- This was specifically used with the RGB lights, but the PWM output could work with any of the LED's

Multi-Bit PWM

Hardware

- All modules incorporated were from Dr. Pong Chu
- A pattern was followed each time something was added
  - First, the files were imported
  - Next, an instantiation of that wrapper was made in the memory-wrapped input/output module
    - Any additional inputs and outputs were also included in the declarations
  - In the top level module, any changes that needed to be made were done
    - The added inputs or outputs were connected in the MMIO instantiation
    - The top inputs and outputs were verified to the given names in the constraints file
- Once all these steps were complete, a bit stream was generated to be exported into software



XADC

- The XADC hardware takes an analog input and converts it to a digital value that can be used in computing
- The system used is 12 bits
- The hardware reads a base and a max value and places the current reading on a scale between those two points
- The Nexys 4 DDR Board has on-chip sensors that can measure the voltage and chip temperature

Chip Temp and Vcc

PWM Slot and Top

Chip Temp and Vcc

```
if (rdy && channel == 5'b00000)
   tmp_out_reg <= adc_data;
if (rdy && channel == 5'b00001)
   vcc_out_reg <= adc_data;
```

```
3'b100:
   r_data <= {16'h0000, tmp_out_reg};
default:
   r_data <= {16'h0000, vcc_out_reg};
```

*** Xilinx made the lower xadc module not included in code above

XADC

- The XADC hardware takes an analog input and converts it to a digital value that can be used in computing
- The system used is 12 bits
- The hardware reads a base and a max value and places the current reading on a scale between those two points
- The Nexys 4 DDR Board has on-chip sensors that can measure the voltage and chip temperature

Chip Temp and Vcc

PWM Slot and Top

PWM Slot and Top

```
 chu_io_pwm_core #(.W(W), .R(R)) gpi_slot6
(.clk(clk),
 .reset(reset),
 .cs(cs_array[`S6_PWM]),
 .read(mem_rd_array[`S6_PWM]),
 .write(mem_wr_array[`S6_PWM]),
 .addr(reg_addr_array[`S6_PWM]),
 .rd_data(rd_data_array[`S6_PWM]),
 .wr_data(wr_data_array[`S6_PWM]),
 .pwm_out(pwm_out)
 );
```

```
 mmio_sys_vanilla #(.N_SW(16),.N_LED(16)) mmio_unit (
 .clk(clk),
 .reset(reset_sys),
 .mmio_cs(fp_mmio_cs),
 .mmio_wr(fp_wr),
 .mmio_rd(fp_rd),
 .mmio_addr(fp_addr),
 .mmio_wr_data(fp_wr_data),
 .mmio_rd_data(fp_rd_data),
 .sw(sw),
 .led(led),
 .rx(rx),
 .tx(tx),
 .adc_p(adc_p),
 .adc_n(adc_n),
 .sseg(sseg),
 .an(an),
 .pwm_out({rgb_led1, rgb_led2}),
 .scl(tmp_i2c_scl),
 .sda(tmp_i2c_sda)
);
```

XADC

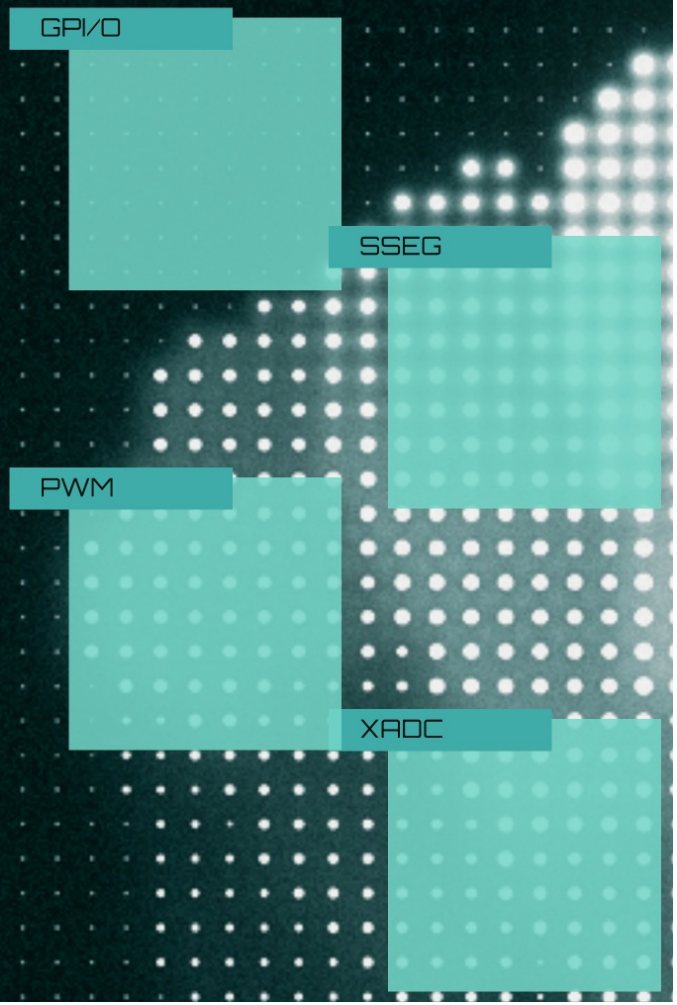
- The XADC hardware takes an analog input and converts it to a digital value that can be used in computing
- The system used is 12 bits
- The hardware reads a base and a max value and places the current reading on a scale between those two points
- The Nexys 4 DDR Board has on-chip sensors that can measure the voltage and chip temperature

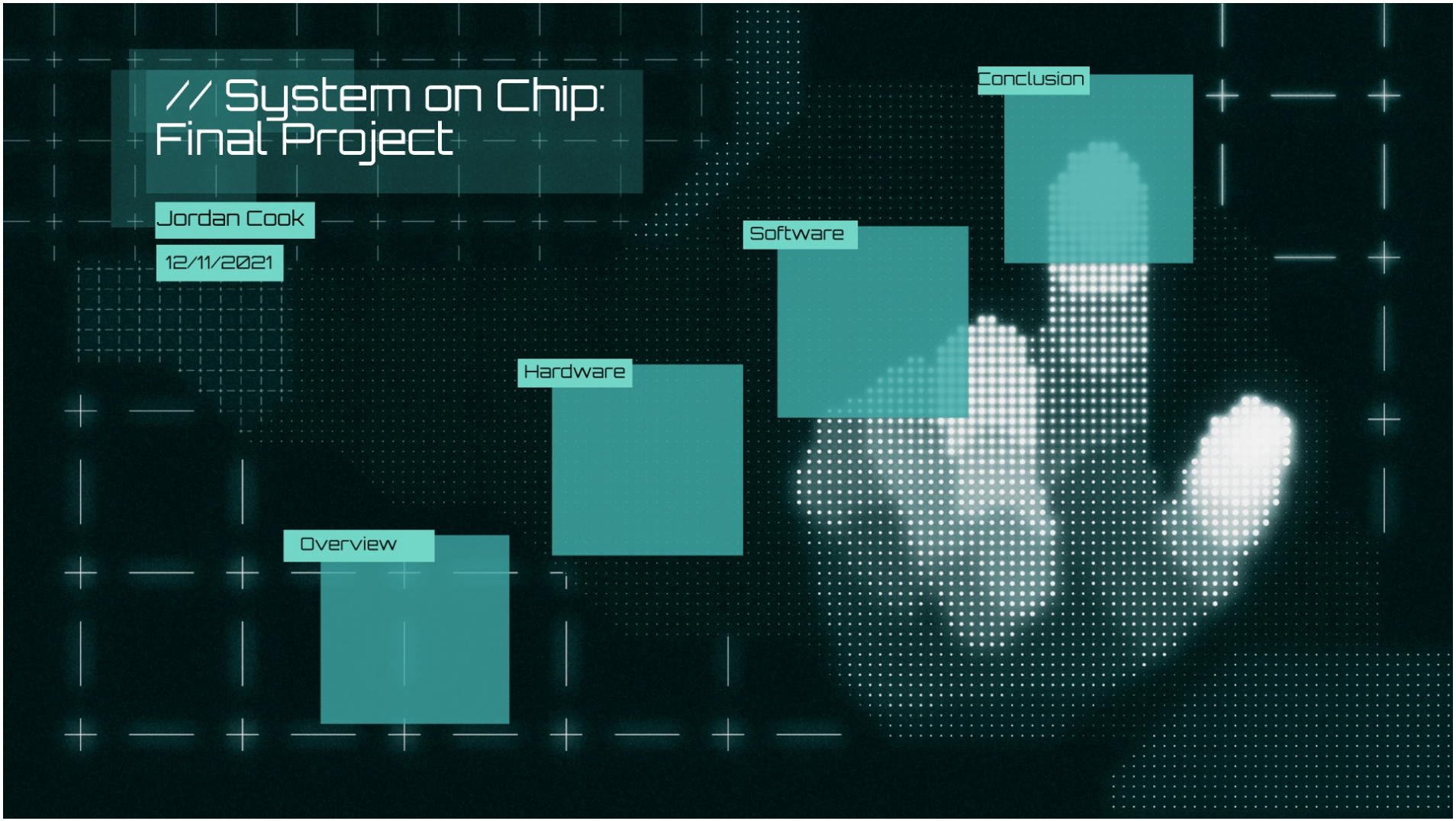
Chip Temp and Vcc

PWM Slot and Top

Hardware

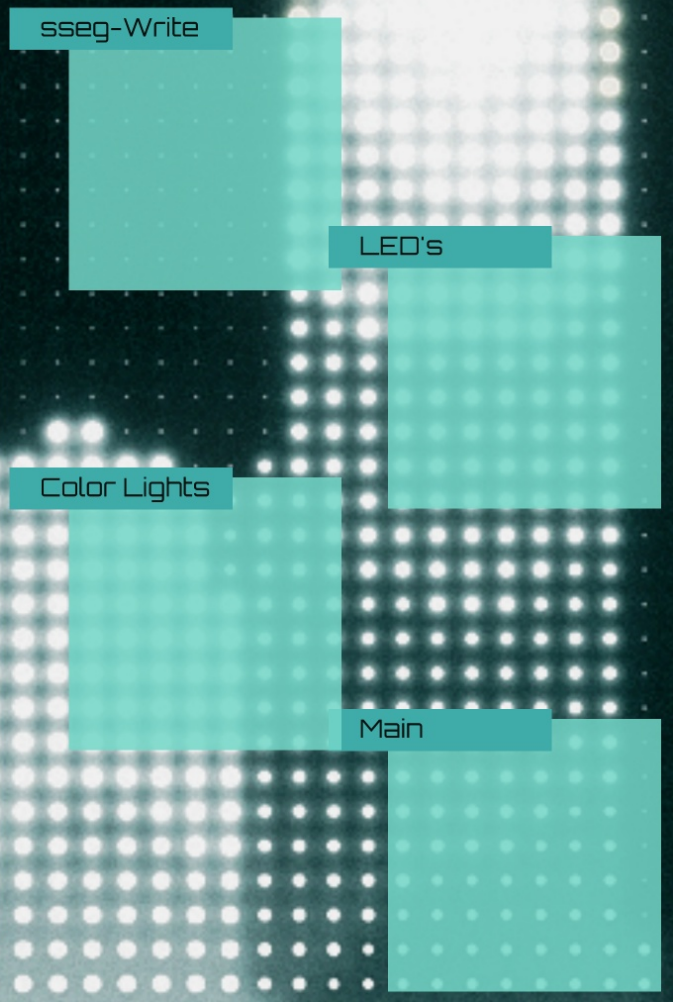
- All modules incorporated were from Dr. Pong Chu
- A pattern was followed each time something was added
  - First, the files were imported
  - Next, an instantiation of that wrapper was made in the memory-wrapped input/output module
    - Any additional inputs and outputs were also included in the declarations
  - In the top level module, any changes that needed to be made were done
    - The added inputs or outputs were connected in the MMIO instantiation
    - The top inputs and outputs were verified to the given names in the constraints file
- Once all these steps were complete, a bit stream was generated to be exported into software




// System on Chip: Final Project
Jordan Cook
12/11/2021
Conclusion
Software
Hardware
Overview

Software

- Using the hardware previously discussed, several functions in software were written to finalize the system
-
- A user could choose between seeing the voltage, the temperature in Celsius, or the temperature in Fahrenheit on the display screen based on which switch they flip
-
- The LED screen would display the value. Then the 16-LEDs would like up in proportion to the value being display. Finally, a color on the RGB lights would change to indicate which of the three values is showing up
-
- Many functions from Dr. Pong Chu were repurposed for this project



sseg-Write

- The first step in this section was to read the voltage on the chip This was done using the 'read_fpga_vcc' function
- From there, that double value was converted into 4 different uint8_t's that isolated each of the digits in the value
- Those values were written to the seven-segment display and a decimal point was put in the proper place
- An almost identical process was done for the temperature, but an additional sseg value was added to indicate Fahrenheit or Celsius

Xadc Functions

Writing Function

Xadc Functions

```
uint16_t XadcCore::read_raw(int n) {
   uint16_t rd_data;

   rd_data = (uint16_t) io_read(base_addr, n) & 0x0000ffff;
   return (rd_data);
}

double XadcCore::read_adc_in(int n) {
   uint16_t raw;
   raw = read_raw(n) >> 4;
   return ((double) raw / 4096.0);
}

// input source 5 is connected to vcc reading
double XadcCore::read_fpga_vcc() {
   return (read_adc_in(VCC_REG) * 3.0);
}

// input source 4 is connected to temperature reading
double XadcCore::read_fpga_temp() {
   return (read_adc_in(TMP_REG) * 503.975 - 273.15);
}
```

sseg-Write

- The first step in this section was to read the voltage on the chip This was done using the 'read_fpga_vcc' function
- From there, that double value was converted into 4 different uint8_t's that isolated each of the digits in the value
- Those values were written to the seven-segment display and a decimal point was put in the proper place
- An almost identical process was done for the temperature, but an additional sseg value was added to indicate Fahrenheit or Celsius

Xadc Functions

Writing Function

Writing Function

```
void writeVCC(XadcCore *vcc, SsegCore *seg) {
    double voltage;
    voltage = vcc -> read_fpga_vcc();
    voltage *= 1000;

    // this turns off any segments currently on
    for (int i = 0; i < 8; i++) {
      seg->write_1ptn(0xff, i);
    }

    // this turns off any decimal points currently on
    seg -> set_dp(0x00);

    uint8_t d1 = voltage / 1000;
    uint8_t d2 =(voltage - d1 * 1000) / 100;
    uint8_t d3 = (voltage - d1 * 1000 - d2 * 100) / 10;
    uint8_t d4 = (voltage - d1 * 1000 - d2 * 100 - d3 * 10);

    uart.disp(d3);
    uart.disp("\n\r");

    seg -> write_1ptn(seg -> h2s(d1), 3);
    seg -> write_1ptn(seg -> h2s(d2), 2);
    seg -> write_1ptn(seg -> h2s(d3), 1);
    seg -> write_1ptn(seg -> h2s(d4), 0);
    seg -> set_dp(8);
    sleep_ms(50);

}
```

sseg-Write

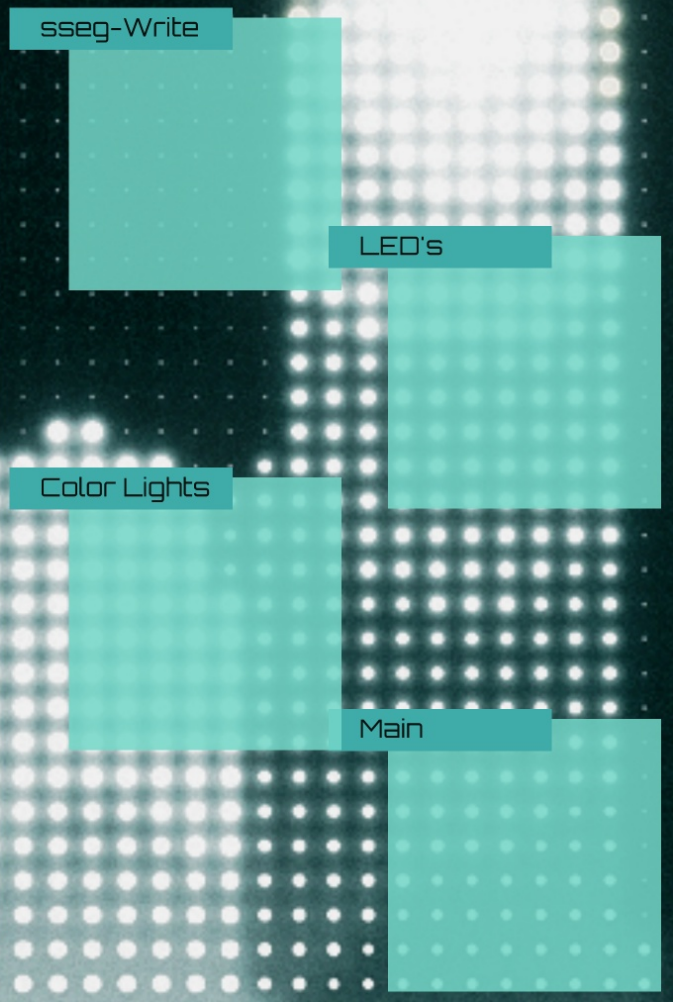
- The first step in this section was to read the voltage on the chip This was done using the 'read_fpga_vcc' function
- From there, that double value was converted into 4 different uint8_t's that isolated each of the digits in the value
- Those values were written to the seven-segment display and a decimal point was put in the proper place
- An almost identical process was done for the temperature, but an additional sseg value was added to indicate Fahrenheit or Celsius

Xadc Functions

Writing Function

Software

- Using the hardware previously discussed, several functions in software were written to finalize the system
-
- A user could choose between seeing the voltage, the temperature in Celsius, or the temperature in Fahrenheit on the display screen based on which switch they flip
-
- The LED screen would display the value. Then the 16-LEDs would like up in proportion to the value being display. Finally, a color on the RGB lights would change to indicate which of the three values is showing up
-
- Many functions from Dr. Pong Chu were repurposed for this project



LED's

- In order to understand the level of voltage that was being displayed, a series of LEDs would light up in sequence
- For voltage, the chip reading was taken as a ratio from 0 to 3.3V and for temperature, the ratio was taken from 0 to 100 (for both Fahrenheit and Celsius)
- The new function utilized the 'read_fpga_vcc()', 'read_fpga_temp() and 'led->write' functions

LED function

LED Function

```
void voltageLED(XadcCore *vcc, GpoCore *led_p) {

    double voltage;
    voltage = vcc -> read_fpga_vcc();

    double num = voltage / 3.3 * 16;

    for (int i = 0; i < num - 1; i++) {
        led_p->write(1, i);
        sleep_ms(60);
    }
}
```

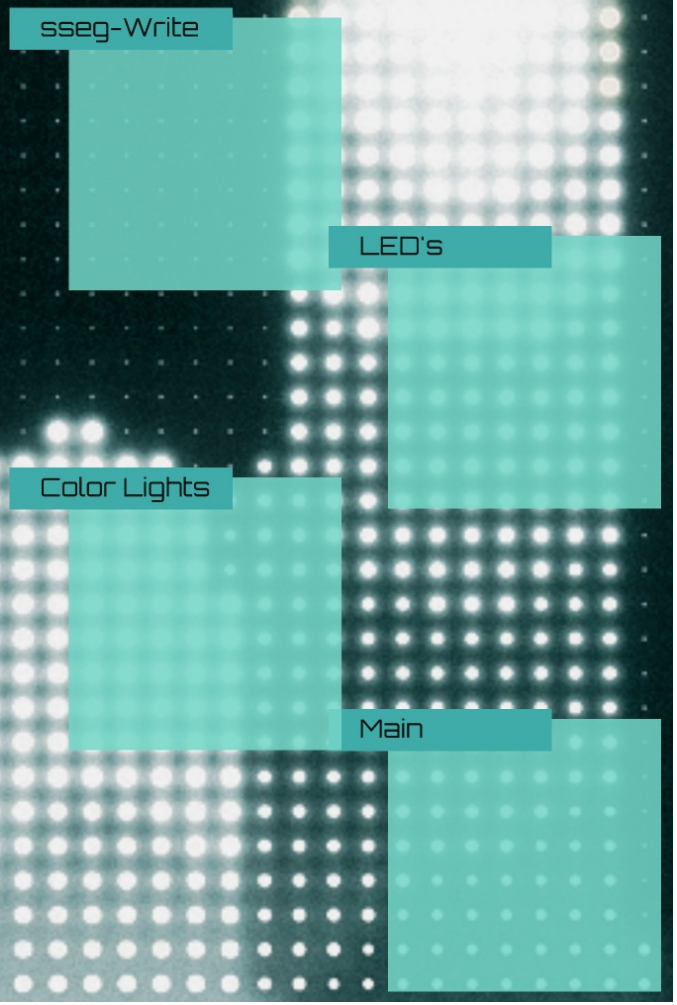
LED's

- In order to understand the level of voltage that was being displayed, a series of LEDs would light up in sequence
- For voltage, the chip reading was taken as a ratio from 0 to 3.3V and for temperature, the ratio was taken from 0 to 100 (for both Fahrenheit and Celsius)
- The new function utilized the 'read_fpga_vcc()', 'read_fpga_temp() and 'led->write' functions

LED function

Software

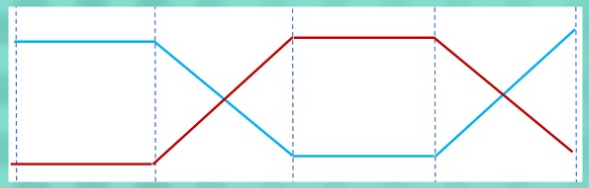
- Using the hardware previously discussed, several functions in software were written to finalize the system
-
- A user could choose between seeing the voltage, the temperature in Celsius, or the temperature in Fahrenheit on the display screen based on which switch they flip
-
- The LED screen would display the value. Then the 16-LEDs would like up in proportion to the value being display. Finally, a color on the RGB lights would change to indicate which of the three values is showing up
-
- Many functions from Dr. Pong Chu were repurposed for this project



Color Lights

- A color-changing light was displayed to indicate to the user what type of value was being shown on the sseg display
  - Red to green was voltage
  - Green to blue was Celsius temperature
  - Blue to red was Fahrenheit temperature

- Each color-changer had four segments
  - Color 1 full, color 2 off
  - Color 1 diminish, color 2 increase
  - Color 1 off, color 2 full
  - color 1 increase, color 2 increase



Setting duty

Color-Changing

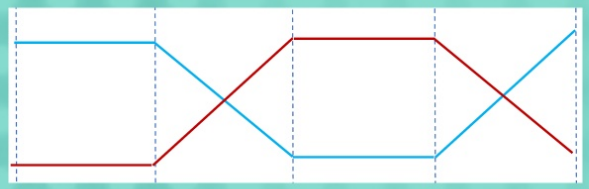
Setting duty

```
void PwmCore::set_duty(int duty, int channel) {
    uint32_t d;

    if (duty > MAX) {
        d = MAX;
    } else {
        d = duty;
    }
    io_write(base_addr, DUTY_REG_BASE + channel, d);
}
```

Color Lights

- A color-changing light was displayed to indicate to the user what type of value was being shown on the sseg display
  - Red to green was voltage
  - Green to blue was Celsius temperature
  - Blue to red was Fahrenheit temperature

- Each color-changer had four segments
  - Color 1 full, color 2 off
  - Color 1 diminish, color 2 increase
  - Color 1 off, color 2 full
  - color 1 increase, color 2 increase



Setting duty

Color-Changing

Color-Changing

```
double P = 1.2589; // this number to the 20th power = 100
```

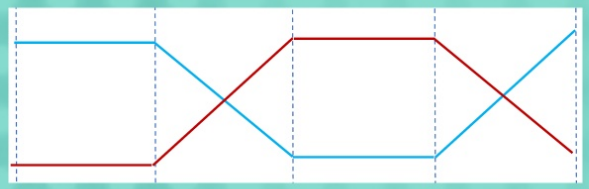
```
if (i <= 20) {
    rgb -> set_duty(percent * 1.0, red);
    rgb -> set_duty(percent * 1.0, red + 3);

    rgb -> set_duty(0.0, blue);
    rgb -> set_duty(0.0, blue + 3);

    sleep_ms(5);
}
else if (i <= 40) {
    bright1 *= P;
    duty1 = bright1 / 100.0;

    bright2 /= P;
    duty2 = bright2;

    rgb -> set_duty(percent * duty1, blue);
    rgb -> set_duty(percent * duty1, blue + 3);
    rgb -> set_duty(percent * duty2, red);
    rgb -> set_duty(percent * duty2, red + 3);

    sleep_ms(100);
}
else if (i <= 60) {
    rgb -> set_duty(0.0, red);
    rgb -> set_duty(0.0, red + 3);

    rgb -> set_duty(percent * 1.0, blue);
    rgb -> set_duty(percent * 1.0, blue + 3);

    sleep_ms(5);
}
```

```
else if (i <=80) {
    bright1 /= P;
    duty1 = bright1 / 100.0;

    bright2 *= P;
    duty2 = bright2;

    rgb -> set_duty(percent * duty2, red);
    rgb -> set_duty(percent * duty2, red + 3);
    rgb -> set_duty(percent * duty1, blue);
    rgb -> set_duty(percent * duty1, blue + 3);

    sleep_ms(100);
}
```

Color Lights

- A color-changing light was displayed to indicate to the user what type of value was being shown on the sseg display
  - Red to green was voltage
  - Green to blue was Celsius temperature
  - Blue to red was Fahrenheit temperature

- Each color-changer had four segments
  - Color 1 full, color 2 off
  - Color 1 diminish, color 2 increase
  - Color 1 off, color 2 full
  - color 1 increase, color 2 increase

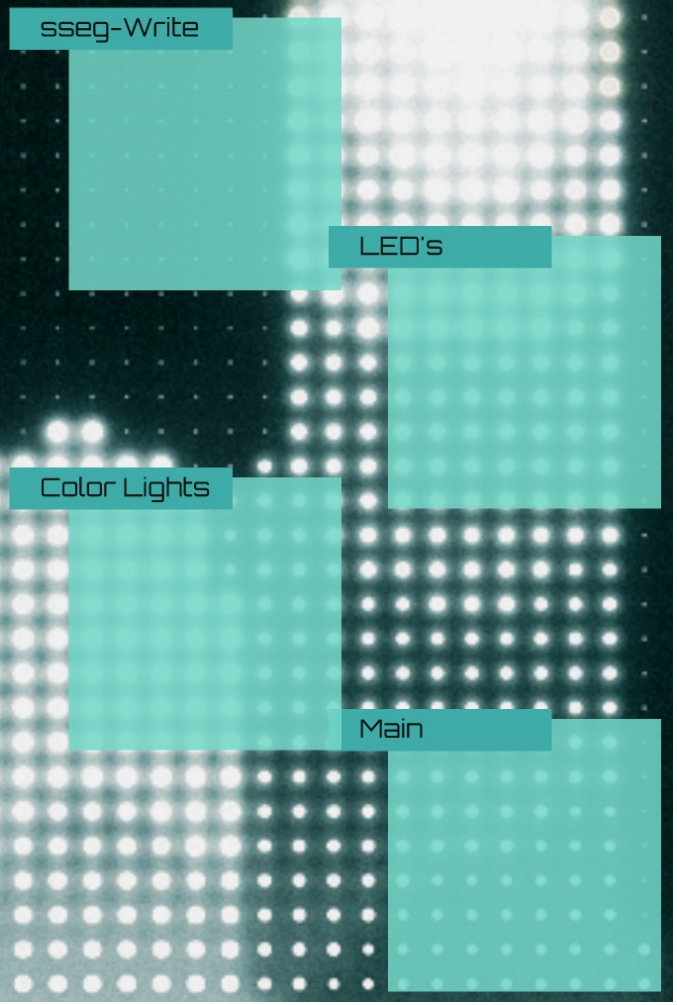


Setting duty

Color-Changing

Software

- Using the hardware previously discussed, several functions in software were written to finalize the system
-
- A user could choose between seeing the voltage, the temperature in Celsius, or the temperature in Fahrenheit on the display screen based on which switch they flip
-
- The LED screen would display the value. Then the 16-LEDs would like up in proportion to the value being display. Finally, a color on the RGB lights would change to indicate which of the three values is showing up
-
- Many functions from Dr. Pong Chu were repurposed for this project



Main

- The last step was incorporating everything in a main loop
- Every cycle, the switches were checked to seewhich value the user wanted to have displayed
- A bool was created to indicate Fahrenheit or Celsius and could aid in allowing the same functions to be used regardless of unit
- Each state would write the value to the sseg, light up the LEDs, and color change the RGB lights
- In retrospect, a state machine would have been a better design, but eh goal was still accomplished

Main Loop Code

Main Loop Code

```
int main() {
   while (1) {

      bool switch0 = sw.read(0);
      bool switch1 = sw.read(1);
      bool switch2 = sw.read(2);

      bool isFahr = 0;

      if (switch0 && !switch1 && !switch2) {
          writeVCC(&vcc, &seg);
          voltageLED(&vcc, &led);
          colorRG(&rgb, &sw);
      }
      else if (!switch0 && switch1 && !switch2) {
          isFahr = 0;
          writeTemp(&temp, &seg, isFahr);
          tempLED(&temp, &led, isFahr);
          colorGB(&rgb, &sw);
      }
      else if (!switch0 && !switch1 && switch2) {
          isFahr = 1;
          writeTemp(&temp, &seg, isFahr);
          tempLED(&temp, &led, isFahr);
          colorBR(&rgb, &sw);
        }
      else {
          errorLED(&led);
          writeZero(&seg);
      }
   }
}
```

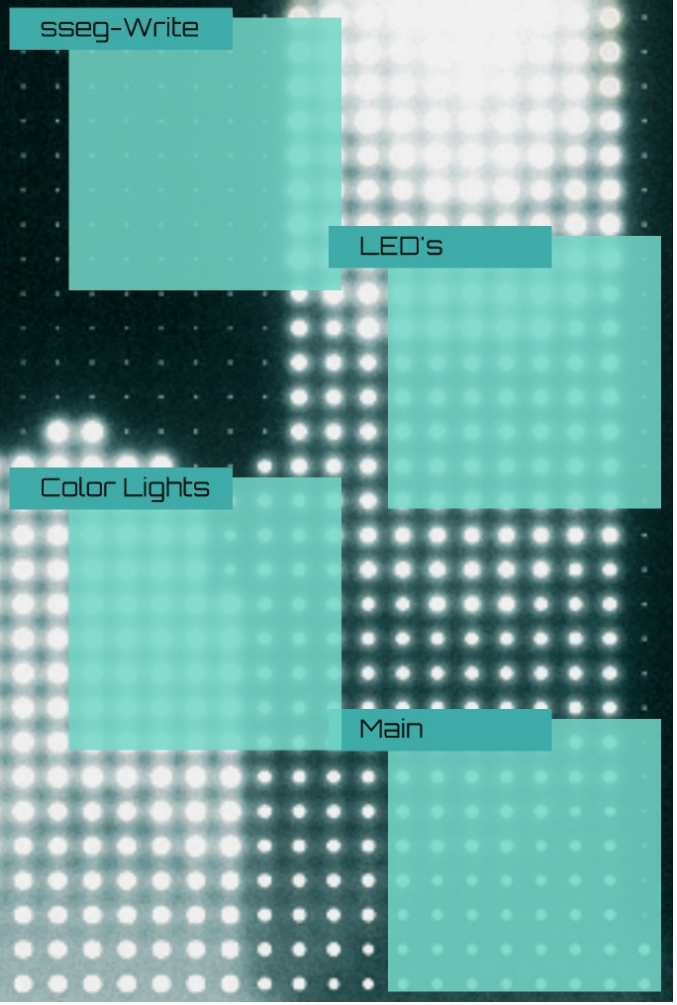
Main

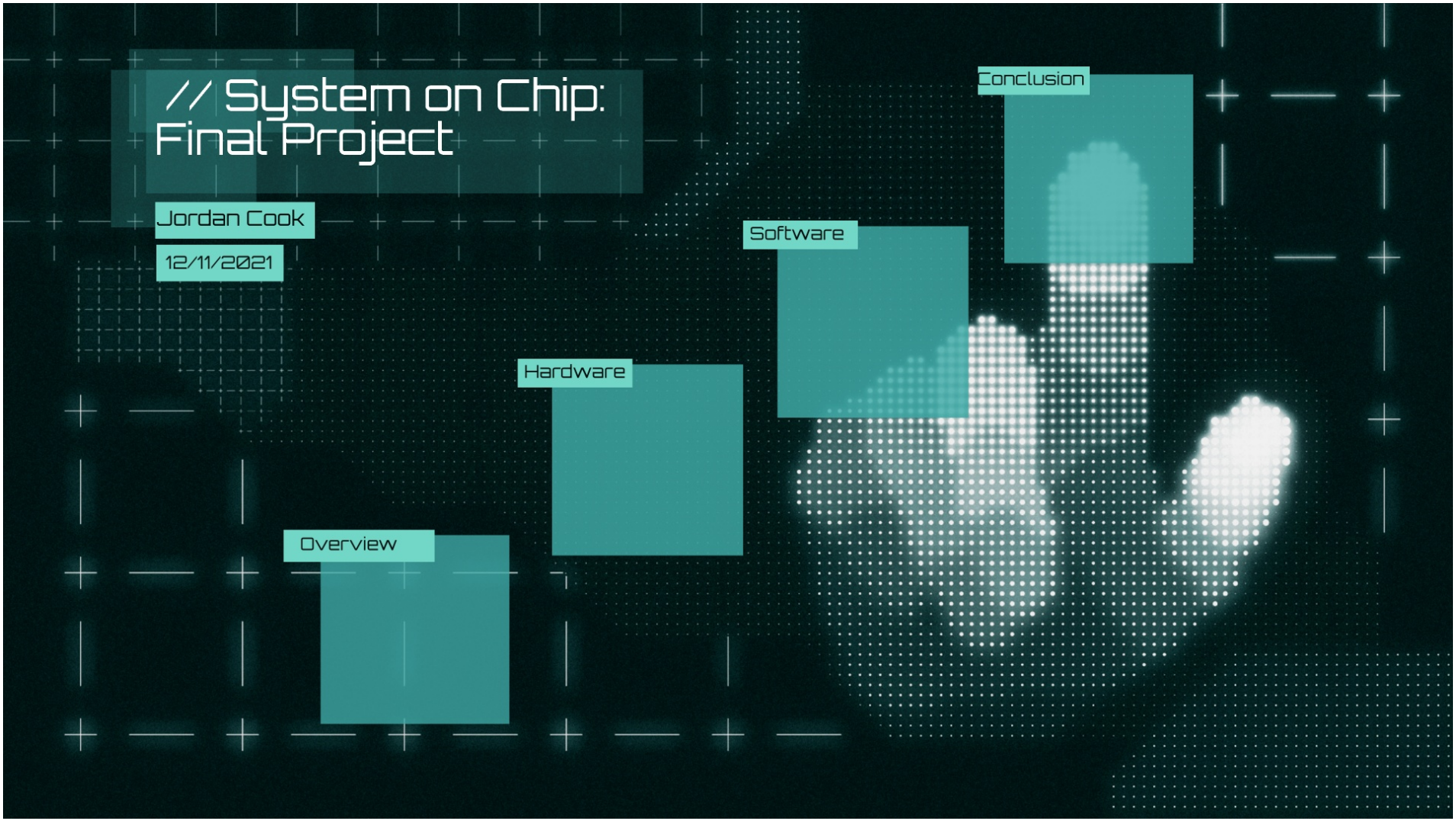
- The last step was incorporating everything in a main loop
- Every cycle, the switches were checked to seewhich value the user wanted to have displayed
- A bool was created to indicate Fahrenheit or Celsius and could aid in allowing the same functions to be used regardless of unit
- Each state would write the value to the sseg, light up the LEDs, and color change the RGB lights
- In retrospect, a state machine would have been a better design, but eh goal was still accomplished

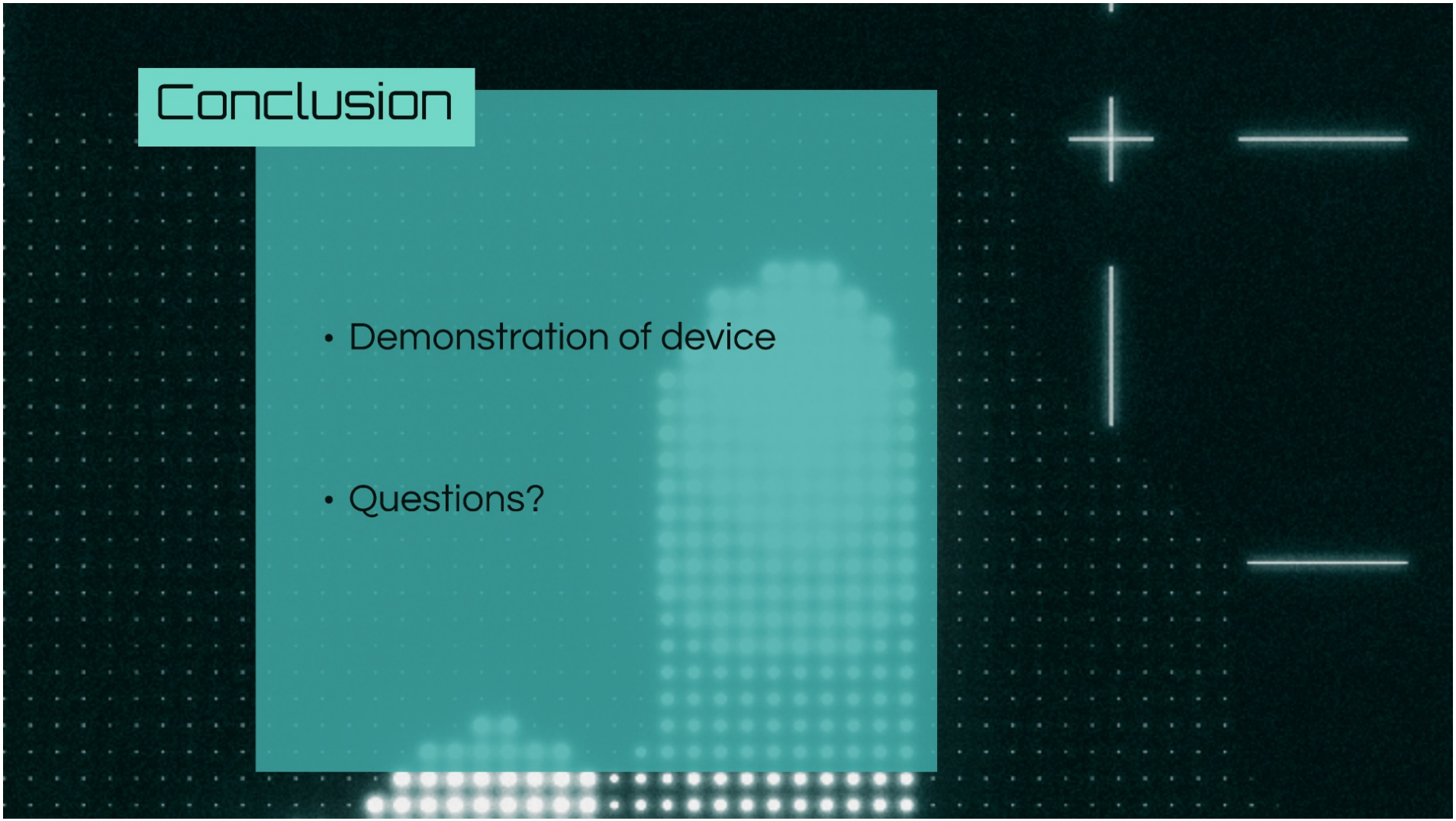
Main Loop Code

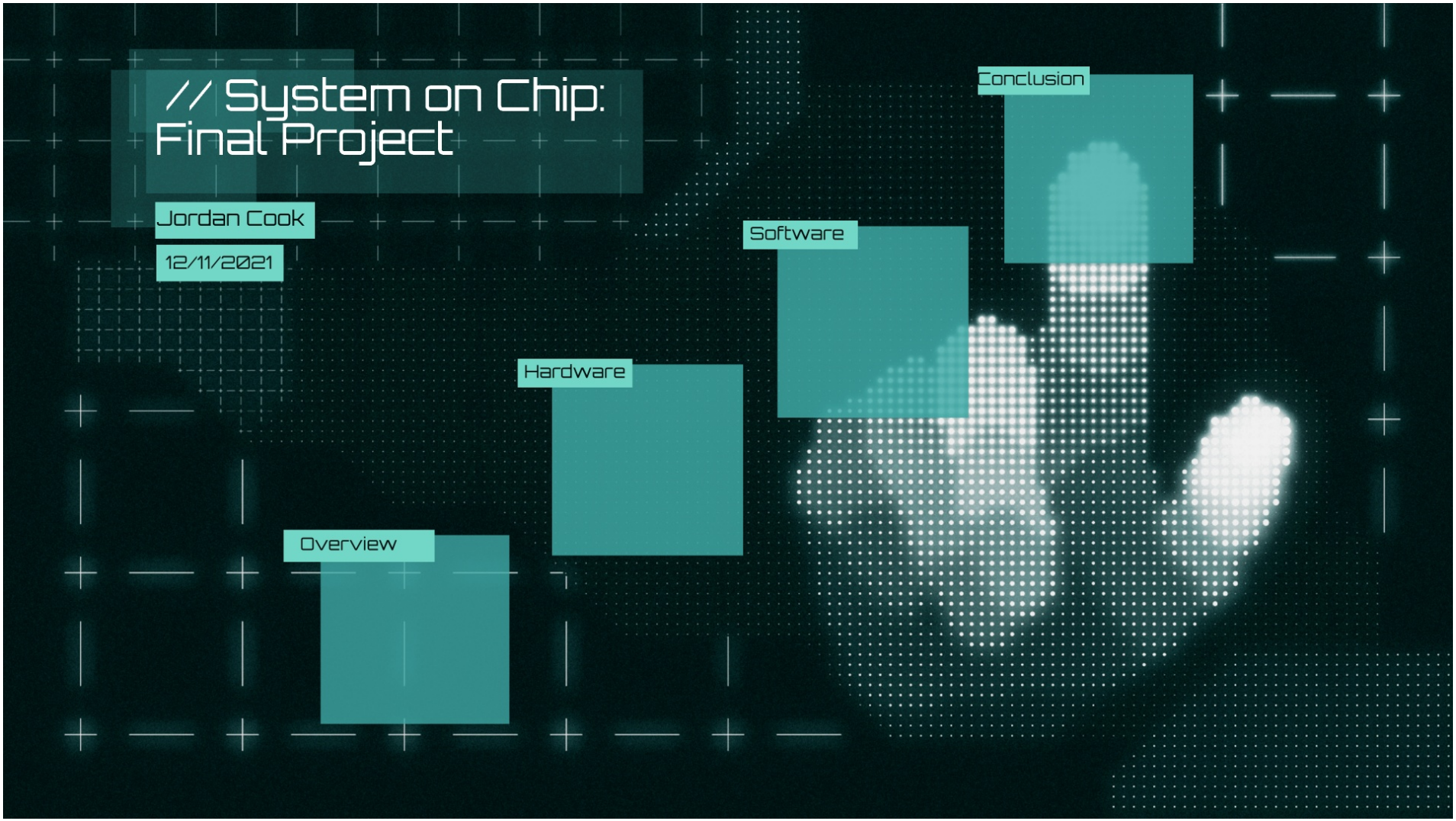
Software

- Using the hardware previously discussed, several functions in software were written to finalize the system
-
- A user could choose between seeing the voltage, the temperature in Celsius, or the temperature in Fahrenheit on the display screen based on which switch they flip
-
- The LED screen would display the value. Then the 16-LEDs would like up in proportion to the value being display. Finally, a color on the RGB lights would change to indicate which of the three values is showing up
-
- Many functions from Dr. Pong Chu were repurposed for this project




// System on Chip: Final Project
Jordan Cook
12/11/2021
Overview
Hardware
Software
Conclusion


Conclusion
• Demonstration of device
• Questions?


// System on Chip: Final Project
Jordan Cook
12/11/2021
Overview
Hardware
Software
Conclusion