ELC 5396 02 Class Report 3

Elizabeth Kooiman

October 26, 2023

Summary

The purpose of the project was to integrate the temperature sensor and the seven segment LEDs on the Nexys4 DDR board and display the temperature on the sseg. The code for this project can be found in GitHub at `https://github.com/elizabethkooiman/Kooiman_SoC`.

First, the hardware components required were integrated into the system. This included the sseg and the temperature sensor. The mmio_sys_vanilla.sv module was changed to include the slots for the debounce, the sseg, and i2c. Then, the additional input and output signals from and to these modules were included in the input and outputs of the overall module. Then, the instantiation of the mmio system in the mcs_top_vanilla.sv file was edited to include the added inputs and outputs. Finally, the constraint file was checked to make sure the required pins were set to let the integrated components function. The bitstream was generated and the xsa file was exported.

Then, the xsa file was used to create a platform project. The platform project was built, and then an application project was created on top of this platform project. The required files for the modules were added to the project. Then, the function provided in the book to test the temperature sensor was run to confirm the functioning of the component. The test function provided in the book for the sseg was then run to ensure it could run properly. Once this was confirmed, a new function that integrated the temperature sensor reading and displayed it on the sseg was created. Another function that converts an integer to the required uint8_t for the sseg was created. These can be found in the main_vanilla_test.cpp file.

Results

The sseg successfully displayed the temperature in Celcius that was measured by the temperature sensor. A picture of the functioning sseg can be seen in Figure 1.

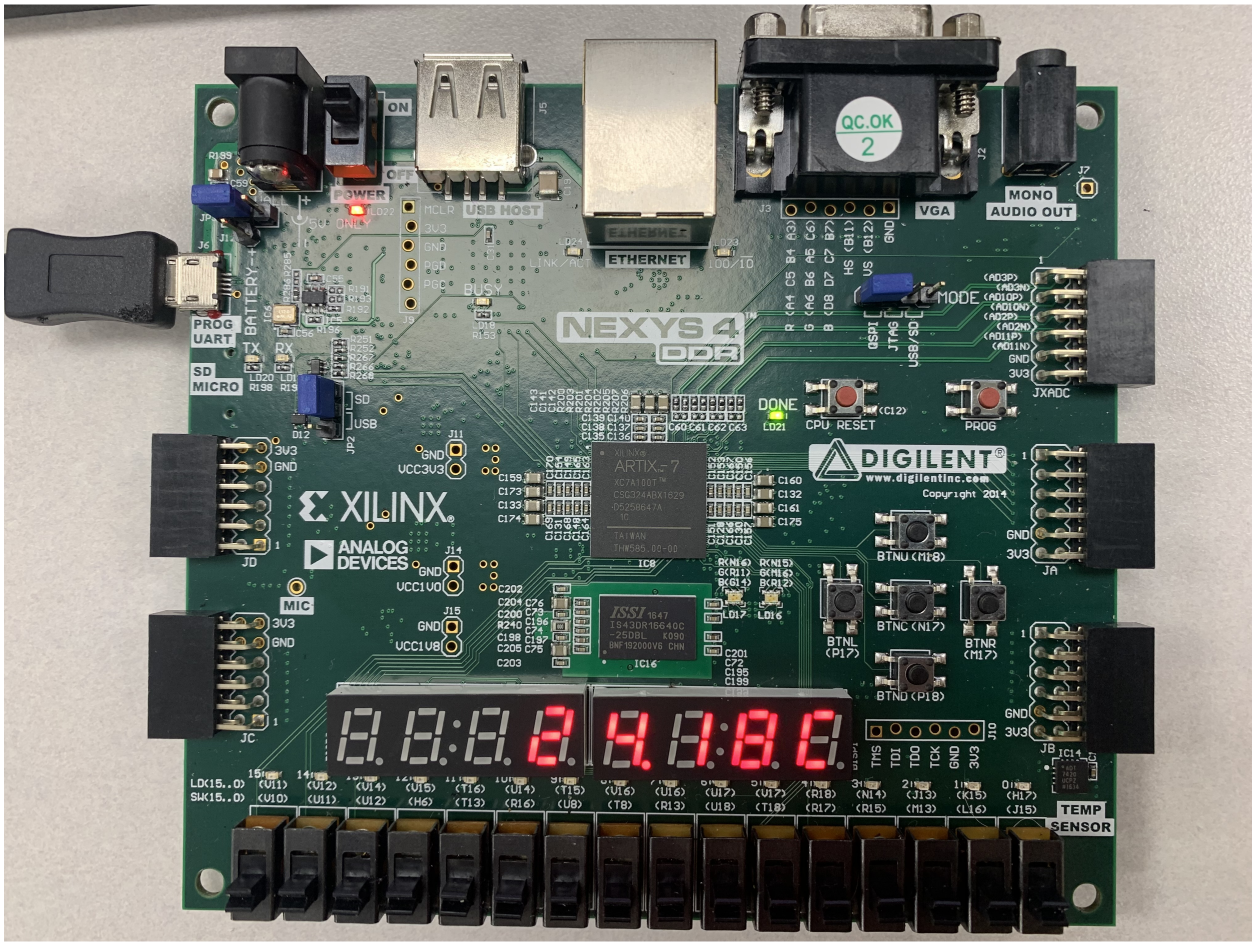

Figure 1: Temperature Sensor sseg Display

Code

Listing 1: edited mmio module

```
‘include "chu_io_map.svh"
module mmio_sys_vanilla
#(
   parameter N_SW = 8,
   parameter N_LED = 8
)
(
   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, // 11 LSB used; 2^6 slots; 2^5 reg each
   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,
   // uart
   input logic rx,
   output logic tx,
   // btn
   input logic [4:0] btn,
   // 8-digit 7-seg LEDs
   output logic [7:0] an,
   output logic [7:0] sseg,
   // i2c temperature sensor
   output logic tmp_i2c_scl,
   inout  tri tmp_i2c_sda
);

   // declaration
   logic [63:0] mem_rd_array;
   logic [63:0] mem_wr_array;
   logic [63:0] cs_array;
   logic [4:0] reg_addr_array [63:0];
   logic [31:0] rd_data_array [63:0];
   logic [31:0] wr_data_array [63:0];

   // body
   // instantiate mmio controller
   chu_mmio_controller ctrl_unit
   (.clk(clk),
    .reset(reset),
    .mmio_cs(mmio_cs),
    .mmio_wr(mmio_wr),
    .mmio_rd(mmio_rd),
    .mmio_addr(mmio_addr),
    .mmio_wr_data(mmio_wr_data),
    .mmio_rd_data(mmio_rd_data),
    // slot interface
    .slot_cs_array(cs_array),
    .slot_mem_rd_array(mem_rd_array),
    .slot_mem_wr_array(mem_wr_array),
    .slot_reg_addr_array(reg_addr_array),
    .slot_rd_data_array(rd_data_array),
    .slot_wr_data_array(wr_data_array)
    );

   // slot 0: system timer
   chu_timer timer_slot0
   (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S0_SYS_TIMER]),
    .read(mem_rd_array[`S0_SYS_TIMER]),
    .write(mem_wr_array[`S0_SYS_TIMER]),
    .addr(reg_addr_array[`S0_SYS_TIMER]),
    .rd_data(rd_data_array[`S0_SYS_TIMER]),
    .wr_data(wr_data_array[`S0_SYS_TIMER])
    );

   // slot 1: UART
   chu_uart uart_slot1
   (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S1_UART1]),
    .read(mem_rd_array[`S1_UART1]),
    .write(mem_wr_array[`S1_UART1]),
    .addr(reg_addr_array[`S1_UART1]),
    .rd_data(rd_data_array[`S1_UART1]),
    .wr_data(wr_data_array[`S1_UART1]),
    .tx(tx),
    .rx(rx)
    );
   //assign rd_data_array[1] = 32'h00000000;

   // 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)
    );

   // slot 3: gpi
   chu_gpi #(.W(N_SW)) gpi_slot3
   (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S3_SW]),
    .read(mem_rd_array[`S3_SW]),
    .write(mem_wr_array[`S3_SW]),
    .addr(reg_addr_array[`S3_SW]),
    .rd_data(rd_data_array[`S3_SW]),
    .wr_data(wr_data_array[`S3_SW]),
    .din(sw)
    );

    // slot 7: btn
     chu_debounce_core #(.W(5), .N(20)) debounce_slot7 //
     (.clk(clk),
      .reset(reset),
      .cs(cs_array[`S7_BTN]),
      .read(mem_rd_array[`S7_BTN]),
      .write(mem_wr_array[`S7_BTN]),
      .addr(reg_addr_array[`S7_BTN]),
      .rd_data(rd_data_array[`S7_BTN]),
      .wr_data(wr_data_array[`S7_BTN]),
      .din(btn)
      );
    // slot 8: led mux
   chu_led_mux_core led_slot8
   (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S8_SSEG]),
    .read(mem_rd_array[`S8_SSEG]),
    .write(mem_wr_array[`S8_SSEG]),
    .addr(reg_addr_array[`S8_SSEG]),
    .rd_data(rd_data_array[`S8_SSEG]),
    .wr_data(wr_data_array[`S8_SSEG]),
    .sseg(sseg),
    .an(an)
    );
    // slot 10: i2c
    chu_i2c_core i2c_slot10
    (.clk(clk),
     .reset(reset),
     .cs(cs_array[`S10_I2C]),
     .read(mem_rd_array[`S10_I2C]),
     .write(mem_wr_array[`S10_I2C]),
     .addr(reg_addr_array[`S10_I2C]),
     .rd_data(rd_data_array[`S10_I2C]),
     .wr_data(wr_data_array[`S10_I2C]),
     .scl(tmp_i2c_scl),
     .sda(tmp_i2c_sda)
     );
   // assign 0's to all unused slot rd_data signals
   generate
      genvar i;
      for (i=11; i<64; i=i+1) begin:  unused_slot_gen
         assign rd_data_array[i] = 32'hffffffff;
      end
   endgenerate
endmodule
```

```
   // slot interface
   //output wire [63:0] slot_cs_array,
   //output wire [63:0] slot_mem_rd_array,
   //output wire [63:0] slot_mem_wr_array,
   //output wire [4:0]  slot_reg_addr_array [0:63],
   //input wire  [31:0] slot_rd_data_array [63:0],
   //output wire [31:0] slot_wr_data_array [63:0]
   // verilog not allow 2d-array port
   //output wire [64*4-1:0]  slot_reg_addr_1d,
   //input wire  [64*32-1:0] slot_rd_data_1d,
   //output wire [64*32-1:0] slot_wr_data_1d

//entity mmio_sys_vanilla is
//   generic(
//     N_LED: integer;
//     N_SW: integer
// );
//   port(
//      -- FPro bus
//      clk          : in  std_logic;
//      reset        : in  std_logic;
//      mmio_cs      : in  std_logic;
//      mmio_wr      : in  std_logic;
//      mmio_rd      : in  std_logic;
//      mmio_addr    : in  std_logic_vector(20 downto 0); -- only 11 LSBs
   used
//      mmio_wr_data : in  std_logic_vector(31 downto 0);
//      mmio_rd_data : out std_logic_vector(31 downto 0);
//      -- switches and LEDs
//      sw           : in  std_logic_vector(N_SW-1 downto 0);
//      led          : out std_logic_vector(N_LED-1 downto 0);
//      -- uart
//      rx           : in  std_logic;
//      tx           : out std_logic
//   );
//end mmio_sys_vanilla;
//
//architecture arch of mmio_sys_vanilla is
//   signal cs_array       : std_logic_vector(63 downto 0);
//   signal reg_addr_array : slot_2d_reg_type;
//   signal mem_rd_array   : std_logic_vector(63 downto 0);
//   signal mem_wr_array   : std_logic_vector(63 downto 0);
//   signal rd_data_array  : slot_2d_data_type;
//   signal wr_data_array  : slot_2d_data_type;
//
//begin
//   --*****************************************************************
//   --  MMIO controller instantiation
//   --*****************************************************************
//   ctrl_unit : entity work.chu_mmio_controller
//      port map(
//         -- FPro bus interface
//          mmio_cs             => mmio_cs,
//          mmio_wr             => mmio_wr,
//          mmio_rd             => mmio_rd,
//          mmio_addr           => mmio_addr,
//          mmio_wr_data        => mmio_wr_data,
//          mmio_rd_data        => mmio_rd_data,
//          -- 64 slot interface
//          slot_cs_array       => cs_array,
//          slot_reg_addr_array => reg_addr_array,
//          slot_mem_rd_array   => mem_rd_array,
//          slot_mem_wr_array   => mem_wr_array,
//          slot_rd_data_array  => rd_data_array,
//          slot_wr_data_array  => wr_data_array
//       );
//
//    --*****************************************************************
//    -- IO slots instantiations
//    --*****************************************************************
//    -- slot 0: system timer
//    timer_slot0 : entity work.chu_timer
//       port map(
//          clk          => clk,
//          reset        => reset,
//          cs           => cs_array[S0_SYS_TIMER],
//          read         => mem_rd_array[S0_SYS_TIMER],
//          write        => mem_wr_array[S0_SYS_TIMER],
//          addr         => reg_addr_array[S0_SYS_TIMER],
//          rd_data      => rd_data_array[S0_SYS_TIMER],
//          wr_data      => wr_data_array[S0_SYS_TIMER]
//       );
//
//    -- slot 1: uart1
//    uart1_slot1 : entity work.chu_uart
//       generic map(FIFO_DEPTH_BIT => 6)
//       port map(
//          clk     => clk,
//          reset   => reset,
//          cs      => cs_array[S1_UART],
//          read    => mem_rd_array[S1_UART],
//          write   => mem_wr_array[S1_UART],
//          addr    => reg_addr_array[S1_UART],
//          rd_data => rd_data_array[S1_UART],
//          wr_data => wr_data_array[S1_UART],
//          -- external signals
//          tx      => tx,
//          rx      => rx
//       );
//
//    -- slot 2: GPO for LEDs
//    gpo_slot2 : entity work.chu_gpo
//       generic map(W => N_LED)
//       port map(
//          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],
//          -- external signal
//          dout     => led
//       );
//
//    -- slot 3: input port for switches
//    gpi_slot3 : entity work.chu_gpi
//       generic map(W => N_SW)
//       port map(
//          clk      => clk,
//          reset    => reset,
//          cs       => cs_array[S3_SW],
//          read     => mem_rd_array[S3_SW],
//          write    => mem_wr_array[S3_SW],
//          addr     => reg_addr_array[S3_SW],
//          rd_data => rd_data_array[S3_SW],
//          wr_data => wr_data_array[S3_SW],
//          -- external signal
//          din      => sw
//       );
//
//    -- assign 0's to all unused slot rd_data signals
//    gen_unused_slot : for i in 4 to 63 generate
//       rd_data_array(i) <= (others => '0');
//    end generate gen_unused_slot;
//end arch;
```

Listing 2: edited top module

```
module mcs_top_vanilla
#(parameter BRG_BASE = 32'hc000_0000)
(
   input logic clk,
   input logic reset_n,
   // switches and LEDs
   input logic [15:0] sw,
   output logic [15:0] led,
   // uart
   input logic rx,
   output logic tx,
   input  logic [4:0] btn,
   // 8-digit 7-seg LEDs
   output logic [7:0] an,
   output logic [7:0] sseg,
   // i2c temperature sensor
   output logic tmp_i2c_scl,
   inout  tri tmp_i2c_sda
);
```

```
// declaration
logic clk_100M;
logic reset_sys;
// MCS IO bus
logic io_addr_strobe;
logic io_read_strobe;
logic io_write_strobe;
logic [3:0] io_byte_enable;
logic [31:0] io_address;
logic [31:0] io_write_data;
logic [31:0] io_read_data;
logic io_ready;
// fpro bus
logic fp_mmio_cs;
logic fp_wr;
logic fp_rd;
logic [20:0] fp_addr;
logic [31:0] fp_wr_data;
logic [31:0] fp_rd_data;

// body
assign clk_100M = clk;                  // 100 MHz external clock
assign reset_sys = !reset_n;

//instantiate uBlaze MCS
cpu cpu_unit (
 .Clk(clk_100M),                      // input wire Clk
 .Reset(reset_sys),                   // input wire Reset
 .IO_addr_strobe(io_addr_strobe),     // output wire IO_addr_strobe
 .IO_address(io_address),             // output wire [31 : 0] IO_address
 .IO_byte_enable(io_byte_enable),     // output wire [3 : 0]
    IO_byte_enable
 .IO_read_data(io_read_data),         // input wire [31 : 0]
    IO_read_data
 .IO_read_strobe(io_read_strobe),     // output wire IO_read_strobe
 .IO_ready(io_ready),                 // input wire IO_ready
 .IO_write_data(io_write_data),       // output wire [31 : 0]
    IO_write_data
 .IO_write_strobe(io_write_strobe)    // output wire IO_write_strobe
);

// instantiate bridge
chu_mcs_bridge #(.BRG_BASE(BRG_BASE)) bridge_unit (.*, .fp_video_cs());

// instantiated i/o subsystem
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),
   //btn
   .btn(btn),
   // 8-digit 7-seg LEDs
   .an(an),
   .sseg(sseg),
   // i2c temperature sensor
   .tmp_i2c_scl(tmp_i2c_scl),
   .tmp_i2c_sda(tmp_i2c_sda)
  );
endmodule
```

Listing 3: main file and application functions

```
/*****************************************************************//**
 * @file main_vanilla_test.cpp
 *
 * @brief Basic test of 4 basic i/o cores
 *
 * @author p chu
 * @version v1.0: initial release
 *********************************************************************/

//#define _DEBUG
#include "chu_init.h"
#include "gpio_cores.h"
#include "sseg_core.h"
#include "i2c_core.h"
/**
 * blink once per second for 5 times.
 * provide a sanity check for timer (based on SYS_CLK_FREQ)
 * @param led_p pointer to led instance
 */
void timer_check(GpoCore *led_p) {
   int i;

   for (i = 0; i < 5; i++) {
      led_p->write(0xffff);
      sleep_ms(500);
      led_p->write(0x0000);
      sleep_ms(500);
      debug("timer check - (loop #)/now: ", i, now_ms());
   }
}

/**
 * check individual led
 * @param led_p pointer to led instance
 * @param n number of led
 */
void led_check(GpoCore *led_p, int n) {
   int i;

   for (i = 0; i < n; i++) {
      led_p->write(1, i);
      sleep_ms(200);
      led_p->write(0, i);
      sleep_ms(200);
   }
}

/**
 * leds flash according to switch positions.
 * @param led_p pointer to led instance
 * @param sw_p pointer to switch instance
 */
void sw_check(GpoCore *led_p, GpiCore *sw_p) {
   int i, s;

   s = sw_p->read();
   for (i = 0; i < 30; i++) {
      led_p->write(s);
      sleep_ms(50);
      led_p->write(0);
      sleep_ms(50);
   }
}

/**
 * uart transmits test line.
 * @note uart instance is declared as global variable in chu_io_basic.h
 */
void uart_check() {
   static int loop = 0;

   uart.disp("uart test #");
   uart.disp(loop);
   uart.disp("\n\r");
   loop++;
}

/**
 * Test debounced buttons
 *   - count transitions of normal and debounced button
 * @param db_p pointer to debouceCore instance
 */

void debounce_check(DebounceCore *db_p, GpoCore *led_p) {
   long start_time;
   int btn_old, db_old, btn_new, db_new;
   int b = 0;
   int d = 0;
   uint32_t ptn;

   start_time = now_ms();
   btn_old = db_p->read();
   db_old = db_p->read_db();
   do {
      btn_new = db_p->read();
      db_new = db_p->read_db();
      if (btn_old != btn_new) {
         b = b + 1;
         btn_old = btn_new;
      }
      if (db_old != db_new) {
         d = d + 1;
         db_old = db_new;
      }
      ptn = d & 0x0000000f;
      ptn = ptn | (b & 0x0000000f) << 4;
      led_p->write(ptn);
   } while ((now_ms() - start_time) < 5000);
}


/**
 * Test pattern in 7-segment LEDs
 * @param sseg_p pointer to 7-seg LED instance
 */

void sseg_check(SsegCore *sseg_p) {
   int i, n;
   uint8_t dp;

   //turn off led
   for (i = 0; i < 8; i++) {
      sseg_p->write_1ptn(0xff, i);
   }
   //turn off all decimal points
   sseg_p->set_dp(0x00);

   // display 0x0 to 0xf in 4 epochs
   // upper 4  digits mirror the lower 4
```

```
    for (n = 0; n < 4; n++) {
        for (i = 0; i < 4; i++) {
            sseg_p->write_1ptn(sseg_p->h2s(i + n * 4), 3 - i);
            sseg_p->write_1ptn(sseg_p->h2s(i + n * 4), 7 - i);
            sleep_ms(300);
        } // for i
    }  // for n
        // shift a decimal point 4 times
    for (i = 0; i < 4; i++) {
        bit_set(dp, 3 - i);
        sseg_p->set_dp(1 << (3 - i));
        sleep_ms(300);
    }
    //turn off led
    for (i = 0; i < 8; i++) {
        sseg_p->write_1ptn(0xff, i);
    }
    //turn off all decimal points
    sseg_p->set_dp(0x00);

}


/*
 * read temperature from adt7420
 * @param adt7420_p pointer to adt7420 instance
 */
void adt7420_check(I2cCore *adt7420_p, GpoCore *led_p) {
    const uint8_t DEV_ADDR = 0x4b;
    uint8_t wbytes[2], bytes[2];
    //int ack;
    uint16_t tmp;
    float tmpC;

    // read adt7420 id register to verify device existence
    // ack = adt7420_p->read_dev_reg_byte(DEV_ADDR, 0x0b, &id);


    wbytes[0] = 0x0b;
    adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
    adt7420_p->read_transaction(DEV_ADDR, bytes, 1, 0);
    uart.disp("read ADT7420 id (should be 0xcb): ");
    uart.disp(bytes[0], 16);
    uart.disp("\n\r");
    //debug("ADT check ack/id: ", ack, bytes[0]);
    // read 2 bytes
    //ack = adt7420_p->read_dev_reg_bytes(DEV_ADDR, 0x0, bytes, 2);
    wbytes[0] = 0x00;
    adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
    adt7420_p->read_transaction(DEV_ADDR, bytes, 2, 0);

    // conversion
    tmp = (uint16_t) bytes[0];
    tmp = (tmp << 8) + (uint16_t) bytes[1];
    if (tmp & 0x8000) {
       tmp = tmp >> 3;
       tmpC = (float) ((int) tmp - 8192) / 16;
    } else {
       tmp = tmp >> 3;
       tmpC = (float) tmp / 16;
    }
    uart.disp("temperature (C): ");
    uart.disp(tmpC);
    uart.disp("\n\r");
    led_p->write(tmp);
    sleep_ms(1000);
    led_p->write(0);


}

uint8_t int2sseg(uint32_t in){
        uint8_t out;
        switch(in){
        case 0:
                out = 0xc0;
                break;
        case 1:
                out = 0xf9;
                break;
        case 2:
                out = 0xa4;
                break;
        case 3:
                out = 0xb0;
                break;
        case 4:
                out = 0x99;
                break;
        case 5:
                out = 0x92;
                break;
        case 6:
                out = 0x82;
                break;
        case 7:
                out=0xf8;
                break;
            case 8:
                out=0x80;
                break;
            case 9:
                out=0x90;
                break;
        }
        return out;
}
void adt7420_sseg(I2cCore *adt7420_p, GpoCore *led_p,SsegCore *sseg_p) {
   const uint8_t DEV_ADDR = 0x4b;
   uint8_t wbytes[2], bytes[2];
   //int ack;
   uint16_t tmp;
   float tmpC;
   int i;
   uint8_t dp;
   uint8_t f = 0x8e;
   uint8_t c = 0xc6;
   int ones, tens, tenth, hundredth;


     //turn off decimal points
     sseg_p->set_dp(0x08);

   // read adt7420 id register to verify device existence
   // ack = adt7420_p->read_dev_reg_byte(DEV_ADDR, 0x0b, &id);

   wbytes[0] = 0x0b;
   adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
   adt7420_p->read_transaction(DEV_ADDR, bytes, 1, 0);
   uart.disp("read ADT7420 id (should be 0xcb): ");
   uart.disp(bytes[0], 16);
   uart.disp("\n\r");
   //debug("ADT check ack/id: ", ack, bytes[0]);
   // read 2 bytes
   //ack = adt7420_p->read_dev_reg_bytes(DEV_ADDR, 0x0, bytes, 2);
   wbytes[0] = 0x00;
   adt7420_p->write_transaction(DEV_ADDR, wbytes, 1, 1);
   adt7420_p->read_transaction(DEV_ADDR, bytes, 2, 0);

   // conversion
   tmp = (uint16_t) bytes[0];
   tmp = (tmp << 8) + (uint16_t) bytes[1];
   if (tmp & 0x8000) {
      tmp = tmp >> 3;
```

```
        tmpC = (float) ((int) tmp - 8192) / 16;
    } else {
        tmp = tmp >> 3;
        tmpC = (float) tmp / 16;
    }
    uart.disp("temperature (C): ");
    uart.disp(tmpC);
    uart.disp("\n\r");
//    led_p->write(tmp);
//    sleep_ms(1000);
//    led_p->write(0);

    tens = int(tmpC/10.0);
    ones = int(tmpC-(tens*10.0));
    tenth = int((tmpC-(tens*10.0)-ones)*10.0);
    hundredth=int((tmpC-(tens*10.0)-ones-(tenth * 0.1))*100.0);



    // ##.##L
        sseg_p->write_1ptn(0xff, 7);
        sseg_p->write_1ptn(0xff, 6);
        sseg_p->write_1ptn(0xff, 5);
        sseg_p->write_1ptn(int2sseg(tens), 4);//#
        sseg_p->write_1ptn(int2sseg(ones), 3);//#
        sseg_p->write_1ptn(int2sseg(tenth), 2);//#
        sseg_p->write_1ptn(int2sseg(hundredth), 1);//#
        sseg_p->write_1ptn(c, 0);//L
}

// instantiate switch, led
GpoCore led(get_slot_addr(BRIDGE_BASE, S2_LED));
GpiCore sw(get_slot_addr(BRIDGE_BASE, S3_SW));
DebounceCore btn(get_slot_addr(BRIDGE_BASE, S7_BTN));
SsegCore sseg(get_slot_addr(BRIDGE_BASE, S8_SSEG));
I2cCore adt7420(get_slot_addr(BRIDGE_BASE, S10_I2C));
int main() {

    while (1) {
//        timer_check(&led);
//        led_check(&led, 16);
//        sw_check(&led, &sw);
//        uart_check();
//        debug("main - switch value / up time : ", sw.read(), now_ms());
//        debounce_check(&btn, &led);
//        sseg_check(&sseg);
//        adt7420_check(&adt7420, &led);
        adt7420_sseg(&adt7420,&led,&sseg);
    } //while
} //main
```