```c
///////////////////////////////////////
/// 640x480 version!
/// This code will segfault the original
/// DE1 computer
/// compile with
/// gcc NTSC_streamwPIO.c -o life -O2
/// -- no optimization yields ??? execution time
/// -- opt -O1 yields ??? mS execution time
/// -- opt -O2 yields ??? mS execution time
/// -- opt -O3 yields ??? mS execution time
///////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/mman.h>
#include <sys/time.h>
#include "address_map_arm_brl4.h"

// PIO Offsets - color thresholds for detecting notes
#define GREEN_MAX_R     0x00000000
#define GREEN_MIN_G     0x00000010
#define GREEN_MAX_B     0x00000020  // Prevents detecting white
#define RED_MIN_R      0x00000030
#define RED_MAX_G     0x00000040
#define RED_MAX_B        0x00000050  // Prevents detecting white
#define YELLOW_MIN_R     0x00000060
#define YELLOW_MIN_G    0x00000070
#define YELLOW_MAX_B       0x00000080  // Prevents detecting white
#define BLUE_MAX_R      0x00000090  // Prevents detecting white
#define BLUE_MAX_G     0x000000a0
#define BLUE_MIN_B        0x000000b0
#define ORANGE_MIN_R     0x000000c0
#define ORANGE_MIN_G       0x000000d0
#define ORANGE_MAX_B       0x000000e0  // Prevents detecting white



/* function prototypes */
void VGA_text (int, int, char *);
void VGA_text_clear();
void VGA_box (int, int, int, int, short);
int  VGA_read_pixel(int, int) ;
int  video_in_read_pixel(int, int);


// the light weight buss base
void *h2p_lw_virtual_base;
volatile unsigned int *h2p_lw_video_in_control_addr=NULL;
volatile unsigned int *h2p_lw_video_edge_control_addr=NULL;

// pixel buffer
volatile unsigned int * vga_pixel_ptr = NULL ;
void *vga_pixel_virtual_base;

//sram pixel
volatile unsigned int * sram_pixel_ptr = NULL ;
void *sram_pixel_virtual_base;

// video input buffer
volatile unsigned int * video_in_ptr = NULL ;
void *video_in_virtual_base;
```

```
// character buffer
volatile unsigned int * vga_char_ptr = NULL ;
void *vga_char_virtual_base;

// /dev/mem file id
int fd;

// shared memory
key_t mem_key=0xf0;
int shared_mem_id;
int *shared_ptr;
int shared_time;
int shared_note;
char shared_str[64];

// pixel macro
#define VGA_PIXEL(x,y,color) do{\
 char  *pixel_ptr ;\
 pixel_ptr = (char *)vga_pixel_ptr + ((y)<<10) + (x) ;\
 *(char *)pixel_ptr = (color);\
} while(0)

#define SRAM_PIXEL(x,y,color) do{\
 char  *pixel_ptr ;\
 pixel_ptr = (char *)sram_pixel_ptr + ((y)<<9) + (x) ;\
 *(char *)pixel_ptr = (color);\
} while(0)

#define VIDEO_IN_PIXEL(x,y,color) do{\
 char  *pixel_ptr ;\
 pixel_ptr = (char *)video_in_ptr + ((y)<<9) + (x) ;\
 *(char *)pixel_ptr = (color);\
} while(0)


// measure time
struct timeval t1, t2;
double elapsedTime;

int main(void)
{

 // Declare volatile pointers to I/O registers (volatile  // means that IO load and store instructions will be used  // to access these
pointer locations,
 // instead of regular memory loads and stores)

 // === need to mmap: =======================
 // FPGA_CHAR_BASE
 // FPGA_ONCHIP_BASE
 // HW_REGS_BASE

 // === get FPGA addresses ==================
    // Open /dev/mem
 if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 )  {
  printf( "ERROR: could not open \"/dev/mem\"...\n" );
  return( 1 );
 }

    // get virtual addr that maps to physical
 h2p_lw_virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
HW_REGS_BASE );
 if( h2p_lw_virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap1() failed...\n" );
  close( fd );
  return(1);
 }
    h2p_lw_video_in_control_addr=(volatile unsigned int *)(h2p_lw_virtual_base+VIDEO_IN_BASE+0x0c);
 *(h2p_lw_video_in_control_addr) = 0x04 ; // turn on video capture
 h2p_lw_video_edge_control_addr=(volatile unsigned int *)(h2p_lw_virtual_base+VIDEO_IN_BASE+0x10);
 *h2p_lw_video_edge_control_addr = 0x01 ; // 1 means edges
 *h2p_lw_video_edge_control_addr = 0x00 ; // 1 means edges


 // === get VGA char addr =====================
 // get virtual addr that maps to physical
 vga_char_virtual_base = mmap( NULL, FPGA_CHAR_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
FPGA_CHAR_BASE );
 if( vga_char_virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap2() failed...\n" );
  close( fd );
  return(1);
 }

   // Get the address that maps to the character
 vga_char_ptr =(unsigned int *)(vga_char_virtual_base);

 // === get VGA pixel addr ====================
 // get virtual addr that maps to physical
 // SDRAM
 vga_pixel_virtual_base = mmap( NULL, SDRAM_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
SDRAM_BASE); //SDRAM_BASE

 if( vga_pixel_virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap3() failed...\n" );
  close( fd );
  return(1);
 }
   // Get the address that maps to the FPGA pixel buffer
 vga_pixel_ptr =(unsigned int *)(vga_pixel_virtual_base);

 // === get SRAM pixel addr ====================
 // get virtual addr that maps to physical
 // SRAM
 sram_pixel_virtual_base = mmap( NULL, FPGA_ONCHIP_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
SRAM_BASE); //on chip SRAM base

 if( sram_pixel_virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap3() failed...\n" );
  close( fd );
  return(1);
 }
 sram_pixel_ptr =(unsigned int *)(sram_pixel_virtual_base);


 // === get video input =======================
 // on-chip RAM
 video_in_virtual_base = mmap( NULL, FPGA_ONCHIP_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED, fd,
FPGA_ONCHIP_BASE);
 if( video_in_virtual_base == MAP_FAILED ) {
  printf( "ERROR: mmap3() failed...\n" );
  close( fd );
  return(1);
 }
 // format the pointer
 video_in_ptr =(unsigned int *)(video_in_virtual_base);

 volatile unsigned int * green_max_r_ptr = NULL;
   volatile unsigned int * green_min_g_ptr = NULL;
   volatile unsigned int * green_max_b_ptr = NULL;
 volatile unsigned int * red_min_r_ptr = NULL;
   volatile unsigned int * red_max_g_ptr = NULL;
   volatile unsigned int * red_max_b_ptr = NULL;
 volatile unsigned int * yellow_min_r_ptr = NULL;
```

```
    volatile unsigned int * yellow_min_g_ptr = NULL;
    volatile unsigned int * yellow_max_b_ptr = NULL;
volatile unsigned int * blue_max_r_ptr = NULL;
    volatile unsigned int * blue_max_g_ptr = NULL;
    volatile unsigned int * blue_min_b_ptr = NULL;
volatile unsigned int * orange_min_r_ptr = NULL;
    volatile unsigned int * orange_min_g_ptr = NULL;
    volatile unsigned int * orange_max_b_ptr = NULL;

green_max_r_ptr = (unsigned int *)(h2p_lw_virtual_base + GREEN_MAX_R);
green_min_g_ptr = (unsigned int *)(h2p_lw_virtual_base + GREEN_MIN_G);
green_max_b_ptr = (unsigned int *)(h2p_lw_virtual_base + GREEN_MAX_B);

red_min_r_ptr = (unsigned int *)(h2p_lw_virtual_base + RED_MIN_R);
red_max_g_ptr = (unsigned int *)(h2p_lw_virtual_base + RED_MAX_G);
red_max_b_ptr = (unsigned int *)(h2p_lw_virtual_base + RED_MAX_B);

yellow_min_r_ptr = (unsigned int *)(h2p_lw_virtual_base + YELLOW_MIN_R);
yellow_min_g_ptr = (unsigned int *)(h2p_lw_virtual_base + YELLOW_MIN_G);
yellow_max_b_ptr = (unsigned int *)(h2p_lw_virtual_base + YELLOW_MAX_B);

blue_max_r_ptr = (unsigned int *)(h2p_lw_virtual_base + BLUE_MAX_R);
blue_max_g_ptr = (unsigned int *)(h2p_lw_virtual_base + BLUE_MAX_G);
blue_min_b_ptr = (unsigned int *)(h2p_lw_virtual_base + BLUE_MIN_B);

orange_min_r_ptr = (unsigned int *)(h2p_lw_virtual_base + ORANGE_MIN_R);
orange_min_g_ptr = (unsigned int *)(h2p_lw_virtual_base + ORANGE_MIN_G);
orange_max_b_ptr = (unsigned int *)(h2p_lw_virtual_base + ORANGE_MAX_B);

//set color thresholds
*(green_max_r_ptr) = 20;
*(green_min_g_ptr) = 22;
*(green_max_b_ptr) = 7;

*(red_min_r_ptr) = 22;
*(red_max_g_ptr) = 20;
*(red_max_b_ptr) = 5;

*(yellow_min_r_ptr) = 22;
*(yellow_min_g_ptr) = 22;
*(yellow_max_b_ptr) =  5;

*(blue_max_r_ptr) = 20;
*(blue_max_g_ptr) = 35;
*(blue_min_b_ptr) = 11;

*(orange_min_r_ptr) = 22;
*(orange_min_g_ptr) = 2;
*(orange_max_b_ptr) = 4;



// ===========================================

/* create a message to be displayed on the VGA
        and LCD displays */
char text_top_row[40] = "DE1-SoC ARM/FPGA\0";
char text_bottom_row[40] = "Cornell ece5760\0";
char num_string[20], time_string[50] ;

// a pixel from the video
int pixel_color;
// video input index
int i,j;

// clear the screen
```

```
 VGA_box (0, 0, 639, 479, 0x03);
 // clear the text
 VGA_text_clear();
 VGA_text (1, 56, text_top_row);
 VGA_text (1, 57, text_bottom_row);

 int player_mode;
 // the actual enter
 printf("How many players? (1 or 2): ");
 scanf("%d", &player_mode);

 while(1)
 {
  gettimeofday(&t1, NULL); // start timer
  // read/write video input -- copy to VGA display
  VIDEO_IN_PIXEL(160,120,0xff); //middle pixel of input

  if ( player_mode == 1 ){
   //Center Highway
   for (i=100; i<220; i++) {
    for (j=175; j<210; j++) {
     pixel_color = video_in_read_pixel(i,j); //read 320x240 NTSC
     SRAM_PIXEL(i,j,pixel_color); //Write to second SRAM for note detector to read
     VGA_PIXEL(i,j,pixel_color); //write to VGA coordinates
    }
   }
   // Single Player Mode Boxes
   VGA_box(114,188,128,179, 0xffffff); //green box, draw in white
   VGA_box(133,188,147,179, 0xffffff); //red box, draw in white
   VGA_box(154,188,168,179, 0xffffff); //yellow box, draw in white
   VGA_box(174,188,188,179, 0xffffff); //blue box, draw in white
   VGA_box(194,188,208,179, 0xffffff); //orange box, draw in white
  }
  else{
   //Left Highway
   for (i=20; i<140; i++) {
    for (j=175; j<210; j++) {
     pixel_color = video_in_read_pixel(i,j); //read 320x240 NTSC
     SRAM_PIXEL(i,j,pixel_color); //Write to second SRAM for note detector to read
     VGA_PIXEL(i,j,pixel_color); //write to VGA coordinates
    }
   }
   // Two Player Mode Boxes
   VGA_box(28,188,42,179, 0xffffff);   //green box, draw in white
   VGA_box(47,188,61,179, 0xffffff);   //red box, draw in white
   VGA_box(68,188,82,179, 0xffffff);   //yellow box, draw in white
   VGA_box(88,188,102,179, 0xffffff);  //blue box, draw in white
   VGA_box(108,188,122,179, 0xffffff); //orange box, draw in white
  }

  // // stop timer
   gettimeofday(&t2, NULL);
   elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000.0;      // sec to ms
   elapsedTime += (t2.tv_usec - t1.tv_usec) / 1000.0;   // us to ms
   pixel_color = VGA_read_pixel(160,120);
   sprintf(time_string, "T=%3.0fmS  color=%x    ", elapsedTime, pixel_color);
   VGA_text (1, 58, time_string);

 } // end while(1)
} // end main

/****************************************************************************************
 * Subroutine to read a pixel from the video input
****************************************************************************************/
int  video_in_read_pixel(int x, int y){
 char  *pixel_ptr ;
 pixel_ptr = (char *)video_in_ptr + ((y)<<9) + (x) ;
```

```c
 return *pixel_ptr ;
}


/****************************************************************************************
 * Subroutine to read a pixel from the VGA monitor
****************************************************************************************/
int  VGA_read_pixel(int x, int y){
 char  *pixel_ptr ;
 pixel_ptr = (char *)vga_pixel_ptr + ((y)<<10) + (x) ;
 return *pixel_ptr ;
}

/****************************************************************************************
 * Subroutine to send a string of text to the VGA monitor
****************************************************************************************/
void VGA_text(int x, int y, char * text_ptr)
{
   volatile char * character_buffer = (char *) vga_char_ptr ; // VGA character buffer
 int offset;
 /* assume that the text string fits on one line */
 offset = (y << 7) + x;
 while ( *(text_ptr) )
 {
  // write to the character buffer
  *(character_buffer + offset) = *(text_ptr);
  ++text_ptr;
  ++offset;
 }
}

/****************************************************************************************
 * Subroutine to clear text to the VGA monitor
****************************************************************************************/
void VGA_text_clear()
{
   volatile char * character_buffer = (char *) vga_char_ptr ; // VGA character buffer
 int offset, x, y;
 for (x=0; x<79; x++){
  for (y=0; y<59; y++){
 /* assume that the text string fits on one line */
   offset = (y << 7) + x;
   // write to the character buffer
   *(character_buffer + offset) = ' ';
  }
 }
}

/****************************************************************************************
 * Draw a filled rectangle on the VGA monitor
****************************************************************************************/
#define SWAP(X,Y) do{int temp=X; X=Y; Y=temp;}while(0)

void VGA_box(int x1, int y1, int x2, int y2, short pixel_color)
{
 char  *pixel_ptr ;
 int row, col;

 /* check and fix box coordinates to be valid */
 if (x1>639) x1 = 639;
 if (y1>479) y1 = 479;
 if (x2>639) x2 = 639;
 if (y2>479) y2 = 479;
 if (x1<0) x1 = 0;
 if (y1<0) y1 = 0;
 if (x2<0) x2 = 0;
```

```
  if (y2<0) y2 = 0;
  if (x1>x2) SWAP(x1,x2);
  if (y1>y2) SWAP(y1,y2);
  for (row = y1; row <= y2; row++)
   for (col = x1; col <= x2; ++col)
   {
    //640x480
    pixel_ptr = (char *)vga_pixel_ptr + (row<<10)    + col ;
    // set pixel color
    *(char *)pixel_ptr = pixel_color;
   }
}
/////////////////////////////////////////////
/// end /////////////////////////////////////
```

```
module DE1_SoC_Computer (
 ///////////////////////////////////
 // FPGA Pins
 ///////////////////////////////////

 // Clock pins
 CLOCK_50,
 CLOCK2_50,
 CLOCK3_50,
 CLOCK4_50,

 // ADC
 ADC_CS_N,
 ADC_DIN,
 ADC_DOUT,
 ADC_SCLK,

 // Audio
 AUD_ADCDAT,
 AUD_ADCLRCK,
 AUD_BCLK,
 AUD_DACDAT,
 AUD_DACLRCK,
 AUD_XCK,

 // SDRAM
 DRAM_ADDR,
 DRAM_BA,
 DRAM_CAS_N,
 DRAM_CKE,
 DRAM_CLK,
 DRAM_CS_N,
 DRAM_DQ,
 DRAM_LDQM,
 DRAM_RAS_N,
 DRAM_UDQM,
 DRAM_WE_N,

 // I2C Bus for Configuration of the Audio and Video-In Chips
 FPGA_I2C_SCLK,
 FPGA_I2C_SDAT,

 // 40-Pin Headers
 GPIO_0,
 GPIO_1,

 // Seven Segment Displays
 HEX0,
 HEX1,
 HEX2,
 HEX3,
 HEX4,
 HEX5,

 // IR
 IRDA_RXD,
 IRDA_TXD,

 // Pushbuttons
 KEY,

 // LEDs
 LEDR,

 // PS2 Ports
 PS2_CLK,
 PS2_DAT,
 PS2_CLK2,
 PS2_DAT2,

 // Slider Switches
 SW,

 // Video-In
 TD_CLK27,
 TD_DATA,
 TD_HS,
 TD_RESET_N,
 TD_VS,

 // VGA
 VGA_B,
 VGA_BLANK_N,
 VGA_CLK,
 VGA_G,
 VGA_HS,
 VGA_R,
 VGA_SYNC_N,
 VGA_VS,

 ///////////////////////////////////////
 // HPS Pins
 ///////////////////////////////////////

 // DDR3 SDRAM
 HPS_DDR3_ADDR,
 HPS_DDR3_BA,
 HPS_DDR3_CAS_N,
 HPS_DDR3_CKE,
 HPS_DDR3_CK_N,
 HPS_DDR3_CK_P,
 HPS_DDR3_CS_N,
 HPS_DDR3_DM,
 HPS_DDR3_DQ,
 HPS_DDR3_DQS_N,
 HPS_DDR3_DQS_P,
 HPS_DDR3_ODT,
 HPS_DDR3_RAS_N,
 HPS_DDR3_RESET_N,
 HPS_DDR3_RZQ,
 HPS_DDR3_WE_N,

 // Ethernet
 HPS_ENET_GTX_CLK,
 HPS_ENET_INT_N,
 HPS_ENET_MDC,
 HPS_ENET_MDIO,
 HPS_ENET_RX_CLK,
 HPS_ENET_RX_DATA,
 HPS_ENET_RX_DV,
 HPS_ENET_TX_DATA,
 HPS_ENET_TX_EN,

 // Flash
 HPS_FLASH_DATA,
 HPS_FLASH_DCLK,
 HPS_FLASH_NCSO,

 // Accelerometer
 HPS_GSENSOR_INT,

 // General Purpose I/O
 HPS_GPIO,
 // I2C
 HPS_I2C_CONTROL,
 HPS_I2C1_SCLK,
 HPS_I2C1_SDAT,
 HPS_I2C2_SCLK,
 HPS_I2C2_SDAT,

 // Pushbutton
 HPS_KEY,

 // LED
 HPS_LED,

 // SD Card
 HPS_SD_CLK,
 HPS_SD_CMD,
 HPS_SD_DATA,

 // SPI
 HPS_SPIM_CLK,
 HPS_SPIM_MISO,
 HPS_SPIM_MOSI,
 HPS_SPIM_SS,

 // UART
 HPS_UART_RX,
 HPS_UART_TX,

 // USB
 HPS_CONV_USB_N,
 HPS_USB_CLKOUT,
 HPS_USB_DATA,
 HPS_USB_DIR,
 HPS_USB_NXT,
 HPS_USB_STP
);

//=======================================================
//  PARAMETER declarations
//=======================================================


//=======================================================
//  PORT declarations
//=======================================================

////////////////////////////////////
// FPGA Pins
////////////////////////////////////

// Clock pins
input      CLOCK_50;
input      CLOCK2_50;
input      CLOCK3_50;
input      CLOCK4_50;

// ADC
inout      ADC_CS_N;
output     ADC_DIN;
input      ADC_DOUT;
output     ADC_SCLK;

// Audio
input      AUD_ADCDAT;
inout      AUD_ADCLRCK;
inout     AUD_BCLK;
output    AUD_DACDAT;
inout     AUD_DACLRCK;
output    AUD_XCK;

// SDRAM
output  [12: 0] DRAM_ADDR;
output [ 1: 0] DRAM_BA;
output    DRAM_CAS_N;
output    DRAM_CKE;
output    DRAM_CLK;
output    DRAM_CS_N;
inout   [15: 0] DRAM_DQ;
output    DRAM_LDQM;
output    DRAM_RAS_N;
output    DRAM_UDQM;
output    DRAM_WE_N;

// I2C Bus for Configuration of the Audio and Video-In Chips
output    FPGA_I2C_SCLK;
inout     FPGA_I2C_SDAT;

// 40-pin headers
inout   [35: 0] GPIO_0;
inout   [35: 0] GPIO_1;

// Seven Segment Displays
output [ 6: 0] HEX0;
output [ 6: 0] HEX1;
output [ 6: 0] HEX2;
output [ 6: 0] HEX3;
output [ 6: 0] HEX4;
output [ 6: 0] HEX5;

// IR
input     IRDA_RXD;
output    IRDA_TXD;

// Pushbuttons
input   [ 3: 0] KEY;

// LEDs
output [ 9: 0] LEDR;

// PS2 Ports
inout     PS2_CLK;
inout     PS2_DAT;

inout     PS2_CLK2;
inout     PS2_DAT2;

// Slider Switches
input   [ 9: 0] SW;

// Video-In
input     TD_CLK27;
input   [ 7: 0] TD_DATA;
input     TD_HS;
output    TD_RESET_N;
input     TD_VS;

// VGA
output [ 7: 0] VGA_B;
output    VGA_BLANK_N;
output     VGA_CLK;
output  [ 7: 0] VGA_G;
output     VGA_HS;
output  [ 7: 0] VGA_R;
output     VGA_SYNC_N;
output     VGA_VS;



///////////////////////////////////
// HPS Pins
///////////////////////////////////

// DDR3 SDRAM
output  [14: 0] HPS_DDR3_ADDR;
output  [ 2: 0]  HPS_DDR3_BA;
output     HPS_DDR3_CAS_N;
output     HPS_DDR3_CKE;
output     HPS_DDR3_CK_N;
output     HPS_DDR3_CK_P;
output     HPS_DDR3_CS_N;
output  [ 3: 0] HPS_DDR3_DM;
inout   [31: 0] HPS_DDR3_DQ;
inout   [ 3: 0] HPS_DDR3_DQS_N;
inout   [ 3: 0] HPS_DDR3_DQS_P;
output     HPS_DDR3_ODT;
output     HPS_DDR3_RAS_N;
output     HPS_DDR3_RESET_N;
input      HPS_DDR3_RZQ;
output     HPS_DDR3_WE_N;

// Ethernet
output     HPS_ENET_GTX_CLK;
inout      HPS_ENET_INT_N;
output     HPS_ENET_MDC;
inout      HPS_ENET_MDIO;
input      HPS_ENET_RX_CLK;
input   [ 3: 0] HPS_ENET_RX_DATA;
input      HPS_ENET_RX_DV;
output  [ 3: 0] HPS_ENET_TX_DATA;
output     HPS_ENET_TX_EN;

// Flash
inout   [ 3: 0] HPS_FLASH_DATA;
output     HPS_FLASH_DCLK;
output     HPS_FLASH_NCSO;

// Accelerometer
inout      HPS_GSENSOR_INT;

// General Purpose I/O
inout   [ 1: 0] HPS_GPIO;

// I2C
inout      HPS_I2C_CONTROL;
inout      HPS_I2C1_SCLK;
inout      HPS_I2C1_SDAT;
inout      HPS_I2C2_SCLK;
inout      HPS_I2C2_SDAT;

// Pushbutton
inout      HPS_KEY;

// LED
```

```
inout     HPS_LED;

// SD Card
output    HPS_SD_CLK;
inout     HPS_SD_CMD;
inout   [ 3: 0] HPS_SD_DATA;

// SPI
output    HPS_SPIM_CLK;
input     HPS_SPIM_MISO;
output    HPS_SPIM_MOSI;
inout     HPS_SPIM_SS;

// UART
input     HPS_UART_RX;
output    HPS_UART_TX;

// USB
inout     HPS_CONV_USB_N;
input     HPS_USB_CLKOUT;
inout   [ 7: 0] HPS_USB_DATA;
input     HPS_USB_DIR;
input     HPS_USB_NXT;
output    HPS_USB_STP;

//=======================================================
//  REG/WIRE declarations
//=======================================================

wire  [23: 0] hex5_hex0;

HexDigit Digit0(HEX0, hex5_hex0[3:0]);
HexDigit Digit1(HEX1, hex5_hex0[7:4]);
HexDigit Digit2(HEX2, hex5_hex0[11:8]);
HexDigit Digit3(HEX3, hex5_hex0[15:12]);
HexDigit Digit4(HEX4, hex5_hex0[19:16]);
HexDigit Digit5(HEX5, hex5_hex0[23:20]);

// MAY need to cycle this switch on power-up to get video
assign TD_RESET_N = SW[0];

// get some signals exposed
// connect bus master signals to i/o for probes
assign GPIO_0[0] = TD_HS ;
assign GPIO_0[1] = TD_VS ;
assign GPIO_0[2] = TD_DATA[6] ;
assign GPIO_0[3] = TD_CLK27 ;
assign GPIO_0[4] = TD_RESET_N ;

// attach color presses to GPIO pins
assign GPIO_0[5] = G;
assign GPIO_0[6] = R;
assign GPIO_0[7] = Y;
assign GPIO_0[8] = B;
assign GPIO_0[9] = O;
assign GPIO_0[10]= strum;

assign LEDR[9]= G;
assign LEDR[8]= R;
assign LEDR[7]= Y;
assign LEDR[6]= B;
assign LEDR[5]= O;
// Skip LED 4 to separate strum from colors
assign LEDR[3]= strum;
assign hex5_hex0[23:20] = G ? 4'ha : 4'h1;
assign hex5_hex0[19:16] = R ? 4'ha : 4'h1;
assign hex5_hex0[15:12] = Y ? 4'ha : 4'h1;
assign hex5_hex0[11:8]  = B ? 4'ha : 4'h1;
assign hex5_hex0[7:4]   = O ? 4'ha : 4'h1;
assign hex5_hex0[3:0]   = strum ? 4'ha : 4'h1;

logic negedge_vs;
logic posedge_vs;
logic [7:0] onchip_sram_data;
logic [16:0] onchip_sram_addr;

logic G;
logic R;
logic Y;
logic B;
logic O;
logic strum;

//Edge Detection of VS
Synchronizer sync(
  .clk(CLOCK_50),
  .reset(~KEY[0]),
  .in_(TD_VS),
  .negedge_(negedge_vs),
  .out(), //DOnt need
  .posedge_(posedge_vs)
);


logic [7:0] green_max_r;
logic [7:0] green_min_g;
logic [7:0] green_max_b;
logic [7:0] red_min_r;
logic [7:0] red_max_g;
logic [7:0] red_max_b;
logic [7:0] yellow_min_r;
logic [7:0] yellow_min_g;
logic [7:0] yellow_max_b;
logic [7:0] blue_max_r;
logic [7:0] blue_max_g;
logic [7:0] blue_min_b;
logic [7:0] orange_min_r;
logic [7:0] orange_min_g;
logic [7:0] orange_max_b;

CloneVRTL clone (
 .clk(CLOCK_50),
 .reset(~KEY[0]),
 .two_player(SW[1]),
 .posedge_TD_VS(posedge_vs),
 .negedge_TD_VS(negedge_vs),
 .box_offset({14'b0,SW[9:2],10'b0}), // From PIO Port or switches, Switches give us 10.24us granularity. Minimum when SW=1, is 1ms delay until press. Max is 5.2ms
 .onchip_sram_data(onchip_sram_data), // Connect to every note detector (pixel_data_val tells each note detector if the data is for it)

 .green_max_r(green_max_r),
 .green_min_g(green_min_g),
 .green_max_b(green_max_b),
 .red_min_r(red_min_r),
 .red_max_g(red_max_g),
 .red_max_b(red_max_b),
 .yellow_min_r(yellow_min_r),
 .yellow_min_g(yellow_min_g),
 .yellow_max_b(yellow_max_b),
 .blue_max_r(blue_max_r),
 .blue_max_g(blue_max_g),
 .blue_min_b(blue_min_b),
 .orange_min_r(orange_min_r),
 .orange_min_g(orange_min_g),
 .orange_max_b(orange_max_b),


   .G(G),     //  GPIO signals
   .R(R),     //  GPIO signals
   .Y(Y),     //  GPIO signals
   .B(B),     //  GPIO signals
   .O(O),     //  GPIO signals
   .strum(strum), //  GPIO signals
   .onchip_sram_addr(onchip_sram_addr) // output from fsm to the sram
);

logic [31:0] dma_address;
logic        dma_waitrequest;
logic        dma_write;
logic [7:0]  dma_writedata;

//=======================================================
//  Structural coding
//=======================================================

Computer_System The_System (
 ////////////////////////////////////
 // FPGA Side
 ////////////////////////////////////

 // Global signals
 .system_pll_ref_clk_clk     (CLOCK_50),
 .system_pll_ref_reset_reset   (1'b0),

 // AV Config
 .av_config_SCLK       (FPGA_I2C_SCLK),
 .av_config_SDAT       (FPGA_I2C_SDAT),

 // Audio Subsystem
 .audio_pll_ref_clk_clk     (CLOCK3_50),
 .audio_pll_ref_reset_reset    (1'b0),
 .audio_clk_clk        (AUD_XCK),
 .audio_ADCDAT        (AUD_ADCDAT),
 .audio_ADCLRCK        (AUD_ADCLRCK),
 .audio_BCLK         (AUD_BCLK),
 .audio_DACDAT        (AUD_DACDAT),
 .audio_DACLRCK        (AUD_DACLRCK),


 // VGA Subsystem
 .vga_pll_ref_clk_clk      (CLOCK2_50),
 .vga_pll_ref_reset_reset    (1'b0),
 .vga_CLK          (VGA_CLK),
 .vga_BLANK         (VGA_BLANK_N),
 .vga_SYNC         (VGA_SYNC_N),
 .vga_HS          (VGA_HS),
 .vga_VS          (VGA_VS),
 .vga_R          (VGA_R),
 .vga_G          (VGA_G),
 .vga_B          (VGA_B),

 // Video In Subsystem
 .video_in_TD_CLK27       (TD_CLK27),
 .video_in_TD_DATA       (TD_DATA),
 .video_in_TD_HS       (TD_HS),
 .video_in_TD_VS       (TD_VS),
 .video_in_clk27_reset     (),
 .video_in_TD_RESET      (),
 .video_in_overflow_flag     (),


 // SDRAM
 .sdram_clk_clk        (DRAM_CLK),
   .sdram_addr        (DRAM_ADDR),
 .sdram_ba        (DRAM_BA),
 .sdram_cas_n        (DRAM_CAS_N),
 .sdram_cke        (DRAM_CKE),
 .sdram_cs_n        (DRAM_CS_N),
 .sdram_dq        (DRAM_DQ),
 .sdram_dqm        ({DRAM_UDQM,DRAM_LDQM}),
 .sdram_ras_n        (DRAM_RAS_N),
 .sdram_we_n        (DRAM_WE_N),

 // On chip SRAM s2 port
 .onchip_sram_s2_address(onchip_sram_addr), // comes from FSM
 .onchip_sram_s2_chipselect(1),             // read enable? Set to 1 bc we always want to read
 .onchip_sram_s2_clken(1),
 .onchip_sram_s2_write(),                  // can leave unconnected since we don't need to write to SRAM
 .onchip_sram_s2_readdata(onchip_sram_data),
 .onchip_sram_s2_writedata(),              // can leave unconnected since we don't need to write to SRAM

 .green_max_r_external_connection_export(green_max_r),
 .green_min_g_external_connection_export(green_min_g),
 .green_max_b_external_connection_export(green_max_b),
 .red_min_r_external_connection_export(red_min_r),
 .red_max_g_external_connection_export(red_max_g),
 .red_max_b_external_connection_export(red_max_b),
 .yellow_min_r_external_connection_export(yellow_min_r),
 .yellow_min_g_external_connection_export(yellow_min_g),
 .yellow_max_b_external_connection_export(yellow_max_b),
 .blue_max_r_external_connection_export(blue_max_r),
 .blue_max_g_external_connection_export(blue_max_g),
 .blue_min_b_external_connection_export(blue_min_b),
 .orange_min_r_external_connection_export(orange_min_r),
 .orange_min_g_external_connection_export(orange_min_g),
 .orange_max_b_external_connection_export(orange_max_b),

 ////////////////////////////////////
 // HPS Side
 ////////////////////////////////////
 // DDR3 SDRAM
 .memory_mem_a   (HPS_DDR3_ADDR),
 .memory_mem_ba   (HPS_DDR3_BA),
 .memory_mem_ck   (HPS_DDR3_CK_P),
 .memory_mem_ck_n  (HPS_DDR3_CK_N),
 .memory_mem_cke  (HPS_DDR3_CKE),
 .memory_mem_cs_n  (HPS_DDR3_CS_N),
 .memory_mem_ras_n  (HPS_DDR3_RAS_N),
 .memory_mem_cas_n  (HPS_DDR3_CAS_N),
 .memory_mem_we_n  (HPS_DDR3_WE_N),
 .memory_mem_reset_n (HPS_DDR3_RESET_N),
 .memory_mem_dq   (HPS_DDR3_DQ),
 .memory_mem_dqs  (HPS_DDR3_DQS_P),
 .memory_mem_dqs_n  (HPS_DDR3_DQS_N),
 .memory_mem_odt  (HPS_DDR3_ODT),
 .memory_mem_dm   (HPS_DDR3_DM),
 .memory_oct_rzqin  (HPS_DDR3_RZQ),

 // Ethernet
 .hps_io_hps_io_gpio_inst_GPIO35 (HPS_ENET_INT_N),
 .hps_io_hps_io_emac1_inst_TX_CLK (HPS_ENET_GTX_CLK),
 .hps_io_hps_io_emac1_inst_TXD0 (HPS_ENET_TX_DATA[0]),
 .hps_io_hps_io_emac1_inst_TXD1 (HPS_ENET_TX_DATA[1]),
 .hps_io_hps_io_emac1_inst_TXD2 (HPS_ENET_TX_DATA[2]),
 .hps_io_hps_io_emac1_inst_TXD3 (HPS_ENET_TX_DATA[3]),
 .hps_io_hps_io_emac1_inst_RXD0 (HPS_ENET_RX_DATA[0]),
 .hps_io_hps_io_emac1_inst_MDIO (HPS_ENET_MDIO),
 .hps_io_hps_io_emac1_inst_MDC  (HPS_ENET_MDC),
 .hps_io_hps_io_emac1_inst_RX_CTL (HPS_ENET_RX_DV),
 .hps_io_hps_io_emac1_inst_TX_CTL (HPS_ENET_TX_EN),
 .hps_io_hps_io_emac1_inst_RX_CLK (HPS_ENET_RX_CLK),
 .hps_io_hps_io_emac1_inst_RXD1 (HPS_ENET_RX_DATA[1]),
 .hps_io_hps_io_emac1_inst_RXD2 (HPS_ENET_RX_DATA[2]),
 .hps_io_hps_io_emac1_inst_RXD3 (HPS_ENET_RX_DATA[3]),

 // Flash
 .hps_io_hps_io_qspi_inst_IO0 (HPS_FLASH_DATA[0]),
 .hps_io_hps_io_qspi_inst_IO1 (HPS_FLASH_DATA[1]),
 .hps_io_hps_io_qspi_inst_IO2 (HPS_FLASH_DATA[2]),
 .hps_io_hps_io_qspi_inst_IO3 (HPS_FLASH_DATA[3]),
 .hps_io_hps_io_qspi_inst_SS0 (HPS_FLASH_NCSO),
 .hps_io_hps_io_qspi_inst_CLK (HPS_FLASH_DCLK),

 // Accelerometer
 .hps_io_hps_io_gpio_inst_GPIO61 (HPS_GSENSOR_INT),

 // General Purpose I/O
 .hps_io_hps_io_gpio_inst_GPIO40 (HPS_GPIO[0]),
 .hps_io_hps_io_gpio_inst_GPIO41 (HPS_GPIO[1]),

 // I2C
 .hps_io_hps_io_gpio_inst_GPIO48 (HPS_I2C_CONTROL),
 .hps_io_hps_io_i2c0_inst_SDA  (HPS_I2C1_SDAT),
 .hps_io_hps_io_i2c0_inst_SCL  (HPS_I2C1_SCLK),
 .hps_io_hps_io_i2c1_inst_SDA  (HPS_I2C2_SDAT),
 .hps_io_hps_io_i2c1_inst_SCL  (HPS_I2C2_SCLK),

 // Pushbutton
 .hps_io_hps_io_gpio_inst_GPIO54 (HPS_KEY),

 // LED
 .hps_io_hps_io_gpio_inst_GPIO53 (HPS_LED),

 // SD Card
 .hps_io_hps_io_sdio_inst_CMD (HPS_SD_CMD),
 .hps_io_hps_io_sdio_inst_D0 (HPS_SD_DATA[0]),
 .hps_io_hps_io_sdio_inst_D1 (HPS_SD_DATA[1]),
 .hps_io_hps_io_sdio_inst_CLK (HPS_SD_CLK),
 .hps_io_hps_io_sdio_inst_D2 (HPS_SD_DATA[2]),
 .hps_io_hps_io_sdio_inst_D3 (HPS_SD_DATA[3]),

 // SPI
 .hps_io_hps_io_spim1_inst_CLK  (HPS_SPIM_CLK),
 .hps_io_hps_io_spim1_inst_MOSI (HPS_SPIM_MOSI),
 .hps_io_hps_io_spim1_inst_MISO (HPS_SPIM_MISO),
 .hps_io_hps_io_spim1_inst_SS0  (HPS_SPIM_SS),

 // UART
 .hps_io_hps_io_uart0_inst_RX (HPS_UART_RX),
 .hps_io_hps_io_uart0_inst_TX (HPS_UART_TX),

 // USB
 .hps_io_hps_io_gpio_inst_GPIO09 (HPS_CONV_USB_N),
 .hps_io_hps_io_usb1_inst_D0  (HPS_USB_DATA[0]),
 .hps_io_hps_io_usb1_inst_D1  (HPS_USB_DATA[1]),
 .hps_io_hps_io_usb1_inst_D2  (HPS_USB_DATA[2]),
 .hps_io_hps_io_usb1_inst_D3  (HPS_USB_DATA[3]),
 .hps_io_hps_io_usb1_inst_D4  (HPS_USB_DATA[4]),
 .hps_io_hps_io_usb1_inst_D5  (HPS_USB_DATA[5]),
 .hps_io_hps_io_usb1_inst_D6  (HPS_USB_DATA[6]),
 .hps_io_hps_io_usb1_inst_D7  (HPS_USB_DATA[7]),
 .hps_io_hps_io_usb1_inst_CLK  (HPS_USB_CLKOUT),
 .hps_io_hps_io_usb1_inst_STP  (HPS_USB_STP),
 .hps_io_hps_io_usb1_inst_DIR  (HPS_USB_DIR),
 .hps_io_hps_io_usb1_inst_NXT  (HPS_USB_NXT)
);


endmodule
```

```
//-----------------------------------------------------------------------
// Clone Hero top level module
//-----------------------------------------------------------------------

typedef struct packed {
   logic val ; //0 if irrelevant
   logic [31:0] cycles; //Num cycles until press
} NoteSchedPkt;

module CloneVRTL
#(
   parameter keyboard_press_time = 1000000,
   parameter box_height = 10, //number of pixels to look for notes in
   parameter num_pixels = 15,
   parameter frame_period = 1668335,
   //Do not set these parameters from outside this module
   parameter box_height_nbits = $clog2(box_height)
)
(
   input logic clk,
   input logic reset,
   input logic two_player,
   input logic posedge_TD_VS, // sync for scheduler, tells sched to request data from detectors
   input logic negedge_TD_VS, // sync for FSM
   input logic [31:0] box_offset, // From PIO Port or switches

   input logic [7:0] onchip_sram_data, // Connect to every note detector (pixel_data_val tells each note detector if the data is for
it)

   // Color Constraint signals coming from PIO ports
   input logic [7:0] green_max_r,
   input logic [7:0] green_min_g,
   input logic [7:0] green_max_b,

   input logic [7:0] red_min_r,
   input logic [7:0] red_max_g,
   input logic [7:0] red_max_b,

   input logic [7:0] yellow_min_r,
   input logic [7:0] yellow_min_g,
   input logic [7:0] yellow_max_b,

   input logic [7:0] blue_max_r,
   input logic [7:0] blue_max_g,
   input logic [7:0] blue_min_b,

   input logic [7:0] orange_min_r,
   input logic [7:0] orange_min_g,
   input logic [7:0] orange_max_b,


   output logic G,     //  GPIO signals
   output logic R,     //  GPIO signals
   output logic Y,     //  GPIO signals
   output logic B,     //  GPIO signals
   output logic O,     //  GPIO signals
   output logic strum, //  GPIO signals
   output logic [16:0] onchip_sram_addr // output from fsm to the sram
);

   // Internal Signals

   logic [4:0] pixel_data_val;
   logic [4:0] sched_recv_val;
   logic [4:0] sched_recv_press;
   logic [box_height_nbits-1:0] sched_recv_edge_height [4:0];
   logic sched_recv_rdy;

   // FSM
   FSMVRTL #(box_height, num_pixels) fsm(
       .clk(clk),
       .two_player(two_player),
       .reset(reset || negedge_TD_VS),
       .onchip_sram_addr(onchip_sram_addr),
       .pixel_data_val(pixel_data_val)
   );

   // Note Detectors

   noteDetectorVRTL #(box_height, num_pixels) green(
       .clk(clk),
       .reset(reset),
       .pixel_data(onchip_sram_data),
       .pixel_data_val(pixel_data_val[4]),
       .min_r(0),
       .max_r(green_max_r),
       .min_g(green_min_g),
       .max_g(112),
       .min_b(0),
       .max_b(green_max_b),
       .send_rdy(sched_recv_rdy),                  // input from scheduler, 1 every 33ms
       .send_val(sched_recv_val[4]),               // output to scheduler, 1 when finished computation
       .send_press(sched_recv_press[4]),           // output to scheduler, 1 if note detected
       .send_edge_height(sched_recv_edge_height[4])
   );

   noteDetectorVRTL #(box_height, num_pixels) red(
       .clk(clk),
       .reset(reset),
       .pixel_data(onchip_sram_data),
       .pixel_data_val(pixel_data_val[3]),
       .min_r(red_min_r),
       .max_r(112),
       .min_g(0),
       .max_g(red_max_g),
       .min_b(0),
       .max_b(red_max_b),
       .send_rdy(sched_recv_rdy),
       .send_val(sched_recv_val[3]),
       .send_press(sched_recv_press[3]),
       .send_edge_height(sched_recv_edge_height[3])
   );

   noteDetectorVRTL #(box_height, num_pixels) yellow(
       .clk(clk),
       .reset(reset),
       .pixel_data(onchip_sram_data),
       .pixel_data_val(pixel_data_val[2]),
       .min_r(yellow_min_r),
       .max_r(112),
       .min_g(yellow_min_g),
       .max_g(112),
       .min_b(0),
       .max_b(yellow_max_b),
       .send_rdy(sched_recv_rdy),
       .send_val(sched_recv_val[2]),
       .send_press(sched_recv_press[2]),
       .send_edge_height(sched_recv_edge_height[2])
   );

   noteDetectorVRTL #(box_height, num_pixels) blue(
       .clk(clk),
       .reset(reset),
       .pixel_data(onchip_sram_data),
       .pixel_data_val(pixel_data_val[1]),
       .min_r(0),
       .max_r(blue_max_r),
       .min_g(0),
       .max_g(blue_max_g),
       .min_b(blue_min_b),
       .max_b(48),
       .send_rdy(sched_recv_rdy),
       .send_val(sched_recv_val[1]),
       .send_press(sched_recv_press[1]),
       .send_edge_height(sched_recv_edge_height[1])
   );

   noteDetectorVRTL #(box_height, num_pixels) orange(
       .clk(clk),
       .reset(reset),
       .pixel_data(onchip_sram_data),
       .pixel_data_val(pixel_data_val[0]),
       .min_r(orange_min_r),
       .max_r(112),
       .min_g(orange_min_g),
       .max_g(112),
       .min_b(0),
       .max_b(orange_max_b),
       .send_rdy(sched_recv_rdy),
       .send_val(sched_recv_val[0]),
       .send_press(sched_recv_press[0]),
       .send_edge_height(sched_recv_edge_height[0])
   );


   NoteSchedPkt  sched_green;
   NoteSchedPkt  sched_red;
   NoteSchedPkt  sched_yellow;
   NoteSchedPkt  sched_blue;
   NoteSchedPkt  sched_orange;


   // Scheduler
   schedulerVRTL #(box_height, frame_period) scheduler(
       .clk(clk),
       .reset(reset),
       .posedge_TD_VS(posedge_TD_VS),
       .box_offset(box_offset),
       .recv_val(sched_recv_val),
       .recv_press(sched_recv_press),
       .recv_edge_height(sched_recv_edge_height),
       .recv_rdy(sched_recv_rdy),
       .green(sched_green),
       .red(sched_red),
       .yellow(sched_yellow),
       .blue(sched_blue),
       .orange(sched_orange)
   );

   // keyboardController
   keyboardControllerVRTL #(keyboard_press_time) keyboardController(
       .clk(clk),
       .reset(reset),
       .green(sched_green),
       .red(sched_red),
       .yellow(sched_yellow),
       .blue(sched_blue),
       .orange(sched_orange),
       .G(G),              // Connect to GPIO pin
```

```verilog
        .R(R),              // Connect to GPIO pin
        .Y(Y),              // Connect to GPIO pin
        .B(B),              // Connect to GPIO pin
        .O(O),              // Connect to GPIO pin
        .strum(strum)
    );

endmodule
```

```
//==============================================================
// FSMVRTL.v
//==============================================================
// FSM for controlling noteDetector logic and memory accesses

module FSMVRTL
#(
   parameter box_height = 9, //number of pixels to look for notes in
   parameter num_pixels = 15
)
(
   input  logic                        clk,
   input  logic                        reset,
   input  logic                        two_player,

   // output logic [4:0]                   end_turn, // 4 green -> 0 orange
   output logic [16:0]                  onchip_sram_addr,
   output logic [4:0]                   pixel_data_val // 4 green , 0 orange
);

 // master box locations
 // 107-111 118-122  green x values
 // 130-134 141-145    red x values
 // 154-158 165-169 yellow x values
 // 177-181 188-192   blue x values
 // 200-204 211-215 orange x values
 // 179-188                y values

logic [1:0] state; // 0 warm up: send req to mem but not val data, 1 normal operation, 2 cooldown: no more req but val data, 3
done

logic [2:0] color_sel;
logic [2:0] color_sel_delayed; // This is due to the one cycle offset for pixel data val.
                              // The last pixel will need to go to the previous note
                              // detector but color_sel will point to the next one already.
logic [3:0] x_pixel_ctr; // 0 to 15
logic [7:0] y_pixel_ctr; // 201 to 209
logic [8:0] x_offset_mux_out;
logic pixel_data_val_internal;

assign onchip_sram_addr = {y_pixel_ctr, x_offset_mux_out + x_pixel_ctr};
assign pixel_data_val_internal = (state == 1 || state == 2);

always_comb begin
   pixel_data_val = 0;
   pixel_data_val[color_sel_delayed-1] = pixel_data_val_internal; // set pixel_data_val based on delayed color_sel (sram takes
one cycle to send data back)
end

always_ff @(posedge clk) begin
   if (reset) begin
       state <= 0;
       y_pixel_ctr <= 8'd179;
       x_pixel_ctr <= 0;
       color_sel <= 5;
       color_sel_delayed <= 5;
   end
   else if (state != 3) begin

       if (state == 0 || state == 2) begin // only spend one cycle in the warmup and cooldown states
         state <= state + 1;
       end

       color_sel_delayed <= color_sel;
```

```
        if (color_sel==0 && state == 1) begin
            state <= 2; // state==2 means we are in cooldown, done getting all pixel data for this frame
        end

        if ((y_pixel_ctr == (8'd179 + (box_height-1))) && x_pixel_ctr==(num_pixels-1)) begin // set read address for all pixels in this
color, move to next color
            color_sel <= color_sel - 1;
            y_pixel_ctr <= 8'd179;
        end
        // y_pixel_ctr logic
        else if (x_pixel_ctr==(num_pixels-1)) begin // reset row, increment y_pixel_counter
            y_pixel_ctr <= y_pixel_ctr + 1;
        end

        if (x_pixel_ctr == (num_pixels-1)) begin // Move to next section of 5 pixels
            x_pixel_ctr <= 0;
        end
        else begin
            x_pixel_ctr <= x_pixel_ctr + 1;
        end
    end
end


always_comb begin
    case (color_sel) //look up table for starting memory location of each box
        4'd1:    x_offset_mux_out = two_player ? 108 : 194; // Orange L
        4'd2:    x_offset_mux_out = two_player ?  88 : 174; // Blue L
        4'd3:    x_offset_mux_out = two_player ?  68 : 154; // Yellow L
        4'd4:    x_offset_mux_out = two_player ?  47 : 133; // Red L
        4'd5:    x_offset_mux_out = two_player ?  28 : 114; // Green L
        default: x_offset_mux_out = 9'bx;
    endcase
end

endmodule
```

```
//=============================================================
// noteDetectorVRTL.v
//=============================================================
// Reads memory box on one lane of highway and determines if note present

module noteDetectorVRTL #(
   parameter box_height = 9, //number of pixels to look for notes in
   parameter num_pixels = 16,
   // Do not set from outside this module
   parameter box_height_nbits = $clog2(box_height)
)(
   input  logic                        clk,
   input  logic                        reset,
   input  logic [7:0]                  pixel_data,      // screen data from memory
   input  logic                        pixel_data_val,  // 1 when the FSM is sending this note detector valid data from SDRAM

   input logic [7:0] min_r,
   input logic [7:0] max_r,
   input logic [7:0] min_g,
   input logic [7:0] max_g,
   input logic [7:0] min_b,
   input logic [7:0] max_b,

   input  logic                        send_rdy,        // from scheduler, ready at 30 fps
   output logic                        send_val,        // finished evaluation. registered. Enable register and input 1 when FSM signals
end_turn to this solver (meaning it has looped through all its pixels)
   output logic                        send_press,      // if note to be pressed, registered. Enable if and only if we detected white
stripe (load with 1 in this case)
   output logic [box_height_nbits-1:0]   send_edge_height // to scheduler, (the height of the edge), registered. Enable at same
time as send_press register
);

  // Accumulate color amount in a row
  logic [8:0] accum_r; // 105 is max value of accum in a single row (7*15px)
  logic [8:0] accum_g; // 105 is max value of accum in a single row (7*15px)
  logic [7:0] accum_b; // 45 is max value of accum in a single row (3*15px)

  logic [7:0] counter; // counts up to num pixels in a row. When it equals num_pixels, reset the accums and the counter
  logic [7:0] row_counter; // start at box_height, count down. This is used to determine edge_height

  // accum registers
  always_ff @(posedge clk) begin
    if (reset || counter == num_pixels-1 || send_val && send_rdy) begin // reset when we reach end of row in boundary box
        accum_r <= 0;
        accum_g <= 0;
        accum_b <= 0;
        counter <= 0;
    end
    else if (pixel_data_val) begin // if valid pixel from memory
        accum_r <= accum_r + pixel_data[7:5];
        accum_g <= accum_g + pixel_data[4:2];
        accum_b <= accum_b + pixel_data[1:0];
        counter <= counter + 1;
    end
  end

  // update row_counter
  always_ff @(posedge clk) begin
    if (reset || (send_val && send_rdy)) begin
        row_counter <= box_height-1;
    end
    else if (counter == num_pixels-1) begin
        row_counter <= row_counter-1;
    end
  end
  // send_val register
  always_ff @(posedge clk) begin
    if (reset || (send_val && send_rdy)) begin
        send_val <= 0;
    end
    else if((row_counter == 0) && (counter == (num_pixels-1))) begin // we receive all the pixels in the boundary box
        send_val <= 1;
    end
  end

  // send_msg registers
  always_ff @(posedge clk) begin
    if (reset || (send_val && send_rdy)) begin
        send_press <= 0;
        send_edge_height <= 0;
    end
    // else if within color constraint, then detect a color
    else if((counter == (num_pixels-1)) &&
    ((accum_r + pixel_data[7:5]) >= min_r) && ((accum_r + pixel_data[7:5]) <= max_r) &&
    ((accum_g + pixel_data[4:2]) >= min_g) && ((accum_g + pixel_data[4:2]) <= max_g) &&
    ((accum_b + pixel_data[1:0]) >= min_b) && ((accum_b + pixel_data[1:0]) <= max_b))
    begin
        send_press <= 1;
        send_edge_height <= row_counter;
    end
  end

endmodule
```

```
//============================================================
// schedulerVRTL.v
//============================================================
// Translates note edge heights to number of cycles until key press, changes output at 30 fps

typedef struct packed {
   logic val ; //0 if irrelevant
   logic [31:0] cycles; //Num cycles until press
} NoteSchedPkt;

module schedulerVRTL
#(
   parameter box_height = 9, //number of pixels to look for notes in
   parameter frame_period = 1668335, // 29.97 fps
   //Do not set these parameters from outside this module
   parameter box_height_nbits = $clog2(box_height),
   parameter cycles_per_pixel = 1668335 / box_height //total cycles spent within box divided by box height (33ms/1 clock
period)/(box height in pixels)
)(
   input  logic        clk,
   input  logic        reset,
   input  logic        posedge_TD_VS, // This is a synchronizer output of the TD_VS signal.
   input  logic [31:0]  box_offset,       // units in cycles, bottom of our box to the center of the hit box, lets guess 50 ms
                                   // This is going to be configured over a PIO port.
   // [4] is for green and [0] is for orange
   input  logic [4:0]   recv_val,        // finished evaluation
   input  logic [4:0]   recv_press,      // if note to be pressed
   input  logic [box_height_nbits-1:0]   recv_edge_height [4:0], // the height of the edge (units in pixels)
   output logic         recv_rdy,        // to noteDetector, should be high every 33 ms

   output NoteSchedPkt  green,
   output NoteSchedPkt  red,
   output NoteSchedPkt  yellow,
   output NoteSchedPkt  blue,
   output NoteSchedPkt  orange
);

   // Get edge height and hard map it to num cycles until hit. If recv_press, then set NoteSchedPkt.val==1

   assign recv_rdy = posedge_TD_VS; // sync with frame update

   NoteSchedPkt schedpkt [4:0];

   assign green      = schedpkt[4];
   assign red        = schedpkt[3];
   assign yellow     = schedpkt[2];
   assign blue       = schedpkt[1];
   assign orange     = schedpkt[0];

   // Tie calibration (press_time and offset) to PIO.
   genvar i;
   generate
      for (i=0; i<5; i++) begin : SCHED
         assign schedpkt[i].val    = recv_val[i] && recv_rdy && recv_press[i];
         assign schedpkt[i].cycles = (recv_edge_height[i] * cycles_per_pixel) + box_offset;
      end
   endgenerate


endmodule
```

```
//==============================================================
// Module for controlling the keyboard over GPIO
//==============================================================
// Counts down number of cycles until press and presses/strums

typedef struct packed {
   logic val ; //0 if irrelevant
   logic [31:0] cycles; //Num cycles until press
} NoteSchedPkt;

module keyboardControllerVRTL #(
   parameter press_time = 500000 // The number of cycles to press a key for (10 ms)
)(
   input  logic        clk,
   input  logic        reset,
   input  NoteSchedPkt  green,
   input  NoteSchedPkt  red,
   input  NoteSchedPkt  yellow,
   input  NoteSchedPkt  blue,
   input  NoteSchedPkt  orange,
   output logic        G,             // Connect to GPIO pin
   output logic        R,             // Connect to GPIO pin
   output logic        Y,             // Connect to GPIO pin
   output logic        B,             // Connect to GPIO pin
   output logic        O,             // Connect to GPIO pin
   output logic        strum          // Connect to GPIO pin
);

   logic op_G;     // tell keyboard to execute a green operation (press green for 10ms, 500,000 cycles)
   logic op_R;     // tell keyboard to execute a red operation (press green for 10ms, 500,000 cycles)
   logic op_Y;     // tell keyboard to execute a yellow operation (press green for 10ms, 500,000 cycles)
   logic op_B;     // tell keyboard to execute a blue operation (press green for 10ms, 500,000 cycles)
   logic op_O;     // tell keyboard to execute a orange operation (press green for 10ms, 500,000 cycles)
   logic op_strum; // tell keyboard to execute an strum operation (press green for 10ms, 500,000 cycles)

   logic [31:0] g_counter;
   logic [31:0] r_counter;
   logic [31:0] y_counter;
   logic [31:0] b_counter;
   logic [31:0] o_counter;
   logic [31:0] strum_counter;

   // strum on any event
   assign op_strum = op_G || op_R || op_Y || op_B || op_O;

   // press key if the counter is not 0
   assign    G =    g_counter != 0;
   assign    R =    r_counter != 0;
   assign    Y =    y_counter != 0;
   assign    B =    b_counter != 0;
   assign    O =    o_counter != 0;
   assign strum = strum_counter != 0;

   // triggering an operation
   always_ff @ (posedge clk) begin
      if (reset) begin
         g_counter <= 0;
         r_counter <= 0;
         y_counter <= 0;
         b_counter <= 0;
         o_counter <= 0;
      end
      else begin //decrement counter
         g_counter    <= (op_G)    ? press_time : ((g_counter==0)    ? 0 : g_counter - 1);
```

```
        r_counter     <= (op_R)     ? press_time : ((r_counter==0)     ? 0 : r_counter - 1);
        y_counter     <= (op_Y)     ? press_time : ((y_counter==0)     ? 0 : y_counter - 1);
        b_counter     <= (op_B)     ? press_time : ((b_counter==0)     ? 0 : b_counter - 1);
        o_counter     <= (op_O)     ? press_time : ((o_counter==0)     ? 0 : o_counter - 1);
        strum_counter <= (op_strum) ? press_time : ((strum_counter==0) ? 0 : strum_counter - 1);
      end
    end

    CountdownQueueVRTL green_q (
      .clk(clk),
      .reset(reset),
      .recv_val(green.val),
      .recv_rdy(), // If queue is full we lose this packet, should always be 1
      .recv_msg(green.cycles),
      .send_val(op_G), //valid when countdown queue reaches 0
      .send_rdy(1),
      .num_free_entries()
    );

    CountdownQueueVRTL red_q (
      .clk(clk),
      .reset(reset),
      .recv_val(red.val),
      .recv_rdy(), // If queue is full we lose this packet, should always be 1
      .recv_msg(red.cycles),
      .send_val(op_R),  //valid when countdown queue reaches 0
      .send_rdy(1),
      .num_free_entries()
    );

    CountdownQueueVRTL yellow_q (
      .clk(clk),
      .reset(reset),
      .recv_val(yellow.val),
      .recv_rdy(), // If queue is full we lose this packet, should always be 1
      .recv_msg(yellow.cycles),
      .send_val(op_Y),  //valid when countdown queue reaches 0
      .send_rdy(1),
      .num_free_entries()
    );

    CountdownQueueVRTL blue_q (
      .clk(clk),
      .reset(reset),
      .recv_val(blue.val),
      .recv_rdy(), // If queue is full we lose this packet, should always be 1
      .recv_msg(blue.cycles),
      .send_val(op_B),  //valid when countdown queue reaches 0
      .send_rdy(1),
      .num_free_entries()
    );

    CountdownQueueVRTL orange_q (
      .clk(clk),
      .reset(reset),
      .recv_val(orange.val),
      .recv_rdy(), // If queue is full we lose this packet, should always be 1
      .recv_msg(orange.cycles),
      .send_val(op_O), //valid when countdown queue reaches 0
      .send_rdy(1),
      .num_free_entries()
    );

endmodule
```

```
//----------------------------------------------------------------------
// Queue with countdown for each entry
//----------------------------------------------------------------------
// hardcoded for 5 entries, assumes always send_rdy

module CountdownQueueVRTL
(
  input  logic                  clk,
  input  logic                  reset,

  input  logic                  recv_val, // connects to valid bit of NoteSchedPkt
  output logic                  recv_rdy,
  input  logic [31:0]           recv_msg, // connects to cycles bit of NoteSchedPkt

  output logic                  send_val, // when this is 1 we should initiate a key press operation in the keyboardControllerVRTL
module
  input  logic                  send_rdy, // from keyboard controller, always 1

  output logic [2:0]  num_free_entries
);

  localparam p_msg_nbits = 32;
  localparam p_num_msgs  = 5;
  localparam c_addr_nbits = 3;

  logic                    write_en;       // Wen to wire to regfile
  logic [c_addr_nbits-1:0] write_addr;     // Waddr to wire to regfile
  logic [c_addr_nbits-1:0] read_addr;      // Raddr to wire to regfile

  // Enqueue and dequeue pointers

  logic [c_addr_nbits-1:0] enq_ptr;
  logic [c_addr_nbits-1:0] enq_ptr_next;

  vc_ResetReg#(c_addr_nbits) enq_ptr_reg
  (
    .clk     (clk),
    .reset   (reset),
    .d       (enq_ptr_next),
    .q       (enq_ptr)
  );

  logic [c_addr_nbits-1:0] deq_ptr;
  logic [c_addr_nbits-1:0] deq_ptr_next;

  vc_ResetReg#(c_addr_nbits) deq_ptr_reg
  (
    .clk   (clk),
    .reset (reset),
    .d     (deq_ptr_next),
    .q     (deq_ptr)
  );

  assign write_addr = enq_ptr;
  assign read_addr  = deq_ptr;

  // Extra state to tell difference between full and empty

  logic full;
  logic full_next;

  vc_ResetReg#(1) full_reg
  (
    .clk   (clk),
    .reset (reset),
    .d     (full_next),
    .q     (full)
  );

  logic  do_enq;
  assign do_enq = recv_rdy && recv_val;

  logic  do_deq;
  assign do_deq = send_rdy && send_val;

  logic  empty;
  assign empty = ~full && (enq_ptr == deq_ptr);

  assign write_en = do_enq;

  assign recv_rdy = ~full;
  assign send_val = ~empty && (entries[deq_ptr] == 0); // Ready to dequeue if data is zero

  logic [c_addr_nbits-1:0] deq_ptr_plus1;
  assign deq_ptr_plus1 = deq_ptr + 1'b1;

  logic [c_addr_nbits-1:0] deq_ptr_inc;
  assign deq_ptr_inc = (deq_ptr_plus1 == p_num_msgs) ? {c_addr_nbits{1'b0}} : deq_ptr_plus1;

  logic [c_addr_nbits-1:0] enq_ptr_plus1;
  assign enq_ptr_plus1 = enq_ptr + 1'b1;

  logic [c_addr_nbits-1:0] enq_ptr_inc;
  assign enq_ptr_inc = (enq_ptr_plus1 == p_num_msgs) ? {c_addr_nbits{1'b0}} : enq_ptr_plus1;

  assign deq_ptr_next = ( do_deq ) ? ( deq_ptr_inc ) : deq_ptr;

  assign enq_ptr_next = ( do_enq ) ? ( enq_ptr_inc ) : enq_ptr;

  assign full_next
    = ( do_enq && ~do_deq && ( enq_ptr_inc == deq_ptr ) ) ? 1'b1
    : ( do_deq && full )                                  ? 1'b0
    :                                                       full;

  assign num_free_entries
    = full              ? {(c_addr_nbits+1){1'b0}}
    : empty             ? p_num_msgs[c_addr_nbits:0]
    : (enq_ptr > deq_ptr) ? p_num_msgs[c_addr_nbits:0] - (enq_ptr - deq_ptr)
    : (deq_ptr > enq_ptr) ? deq_ptr - enq_ptr
    :                     {(c_addr_nbits+1){1'bx}};


  // Dpath

  logic [31:0] entries [4:0];

  always_ff @ (posedge clk) begin
    if (reset) begin
       entries[0] <= 0;
       entries[1] <= 0;
       entries[2] <= 0;
       entries[3] <= 0;
       entries[4] <= 0;
    end
    else if (write_en) begin //other entries not decremented but should be negligible
      entries[write_addr] <= recv_msg;
    end
    else begin // decrement all entries if they are non zero (even if they are not valid it shouldn't matter)
       entries[0] <= (entries[0] == 0) ? 0 : entries[0] - 1;
       entries[1] <= (entries[1] == 0) ? 0 : entries[1] - 1;
       entries[2] <= (entries[2] == 0) ? 0 : entries[2] - 1;
```

```
      entries[3] <= (entries[3] == 0) ? 0 : entries[3] - 1;
      entries[4] <= (entries[4] == 0) ? 0 : entries[4] - 1;
    end
  end

endmodule
```