Mid-Term Exam 2017

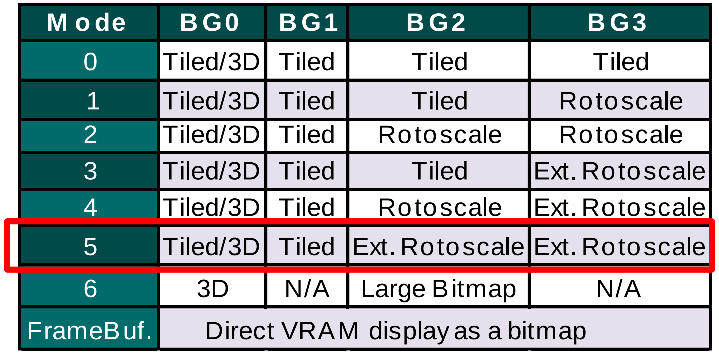
Code review

Systèmes Embarqués Microprogrammés

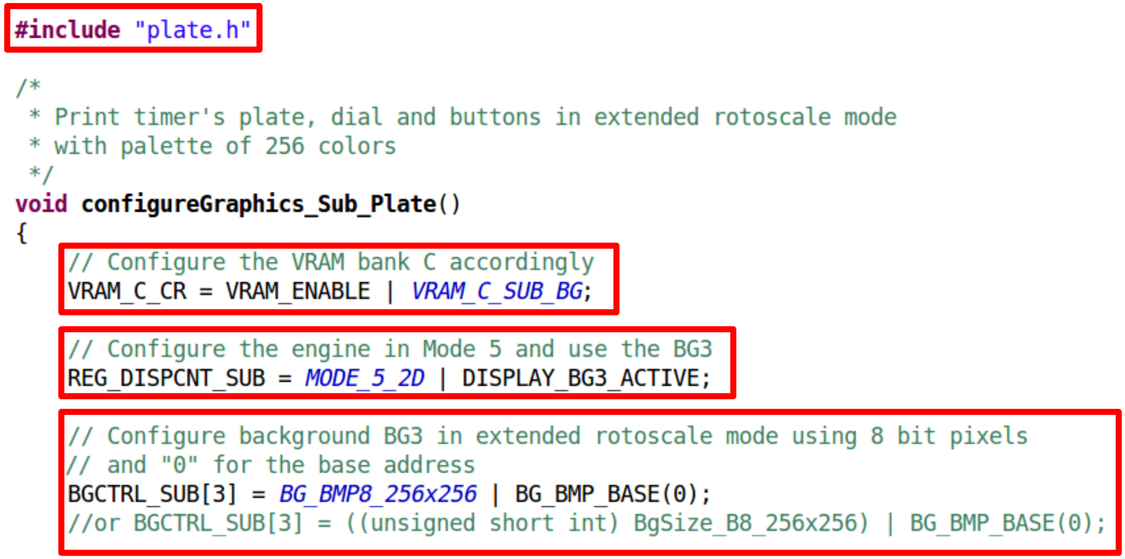
Prof. David Atienza Alonso, SEL-STI

Exercise 1: Configure SUB engine

- Complete the function **configureGraphics_Sub_Plate()** in the file **graphics_sub.c** following the given comments to configure the SUB engine in mode 5.
  1. Enable and configure VRAM bank accordingly
  2. Configure the engine in Mode 5 and use background 3
  3. Configure the background in extended rotoscale mode
  4. Include the “plate.h” file generated by grit



Mode	BG0	BG1	BG2	BG3
0	Tiled/3D	Tiled	Tiled	Tiled
1	Tiled/3D	Tiled	Tiled	Rotoscale
2	Tiled/3D	Tiled	Rotoscale	Rotoscale
3	Tiled/3D	Tiled	Tiled	Ext. Rotoscale
4	Tiled/3D	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled/3D	Tiled	Ext. Rotoscale	Ext. Rotoscale
6	3D	N/A	Large Bitmap	N/A
FrameBuf.	Direct VRAM display as a bitmap			

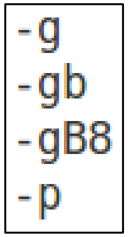


```
#include "plate.h"

/*
 * Print timer's plate, dial and buttons in extended rotoscale mode
 * with palette of 256 colors
 */
void configureGraphics_Sub_Plate()
{
    // Configure the VRAM bank C accordingly
    VRAM_C_CR = VRAM_ENABLE | VRAM_C_SUB_BG;

    // Configure the engine in Mode 5 and use the BG3
    REG_DISPCNT_SUB = MODE_5_2D | DISPLAY_BG3_ACTIVE;

    // Configure background BG3 in extended rotoscale mode using 8 bit pixels
    // and "0" for the base address
    BGCTRL_SUB[3] = BG_BMP8_256x256 | BG_BMP_BASE(0);
    //or BGCTRL_SUB[3] = ((unsigned short int) BgSize_B8_256x256) | BG_BMP_BASE(0);
```

Exercise 1: Download image on screen

- Download the image **plate.png** into the **data** folder of the project and create the configuration grit file to obtain the bitmap, and the corresponding 256 colors palette (using pixels represented with 8 bits).

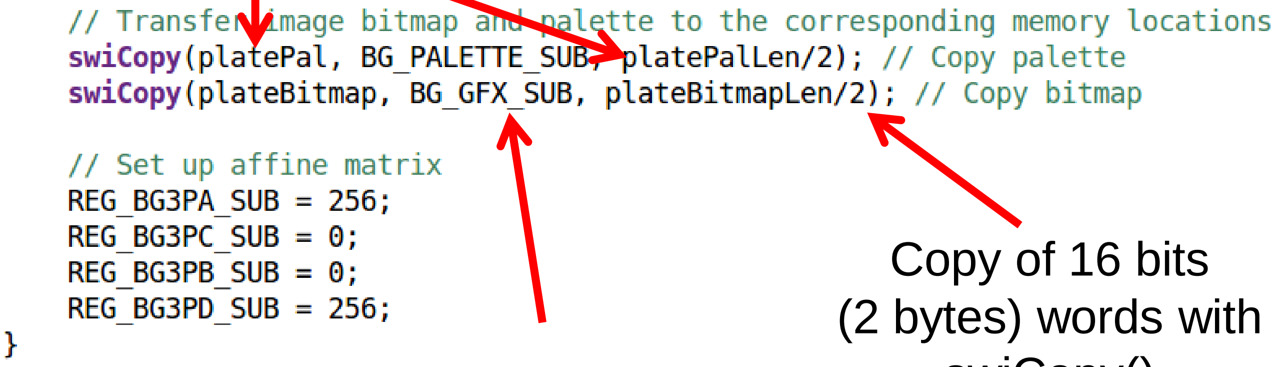
“plate.grit” file



```
-g
-gb
-gB8
-p
```

- -g: Include graphic data
- -gb: Generate Bitmap
- -gB8: Set pixel size to 8 bits (for palette)
- -p: Include palette

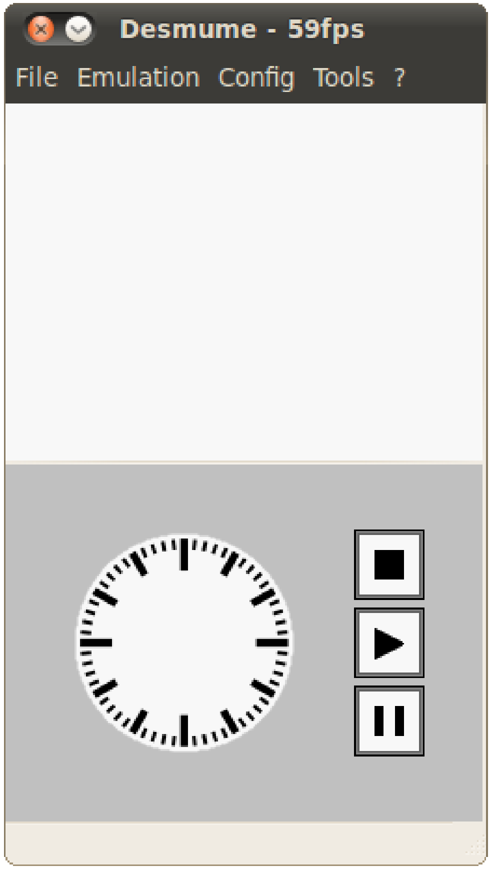
Generated by grit (declared inside “plate.h”)



```
    // Transfer image bitmap and palette to the corresponding memory locations
    swiCopy(platePal, BG_PALETTE_SUB, platePalLen/2); // Copy palette
    swiCopy(plateBitmap, BG_GFX_SUB, plateBitmapLen/2); // Copy bitmap

    // Set up affine matrix
    REG_BG3PA_SUB = 256;
    REG_BG3PC_SUB = 0;
    REG_BG3PB_SUB = 0;
    REG_BG3PD_SUB = 256;
}
```

BG_BMP_RAM_SUB(0) also valid

Copy of 16 bits (2 bytes) words with swiCopy()

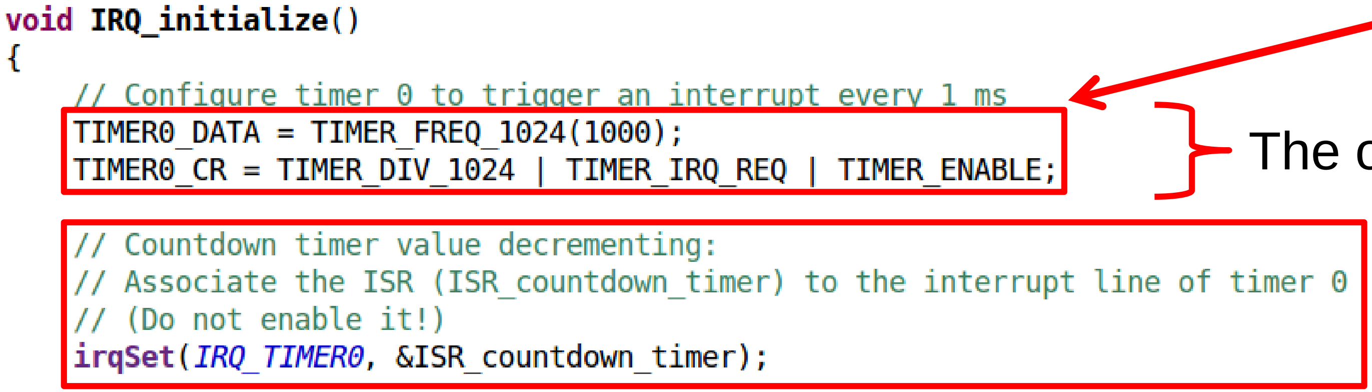


Exercise 2: Configure Timer

- The upper screen will be used to display the time value of the countdown timer. This value will be decremented **every 1 ms**, using interrupts of the hardware timer. The functions inside the file **irq_management.c**, which are prototyped in the header file **irq_management.h** must be completed.
  1. Configure the timer to trigger an interrupt at 1000 Hz (1000 times per second)
  2. Associate the ISR (*ISR_countdown_timer*) to the interrupt line

All the DIV_**X** are valid!

$(2^{16}-1) * \mathbf{X} / 33514000 > 1$ ms



```
void IRQ_initialize()
{
    // Configure timer 0 to trigger an interrupt every 1 ms
    TIMER0_DATA = TIMER_FREQ_1024(1000);
    TIMER0_CR = TIMER_DIV_1024 | TIMER_IRQ_REQ | TIMER_ENABLE;

    // Countdown timer value decrementing:
    // Associate the ISR (ISR_countdown_timer) to the interrupt line of timer 0
    // (Do not enable it!)
    irqSet(IRQ_TIMER0, &ISR_countdown_timer);
```

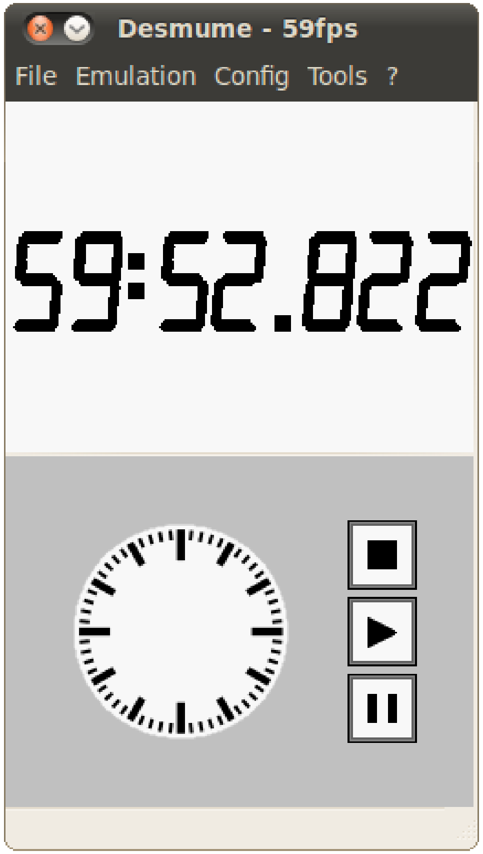
The order matters!

As touchscreen will be used later, **irqInit()** must NOT be called!

Exercise 2: Implement Interrupt Service Routine (ISR)

- Refresh the main screen
  1. Associate the ISR (**ISR_VBlank**) to the interrupt line VBLANK and enable it

```
    // Main screen refreshing:
    // Associate the ISR (ISR_VBlank) to the interrupt line VBLANK and enable it
    irqSet(IRQ_VBLANK, &ISR_VBlank);
    irqEnable(IRQ_VBLANK);
}
```

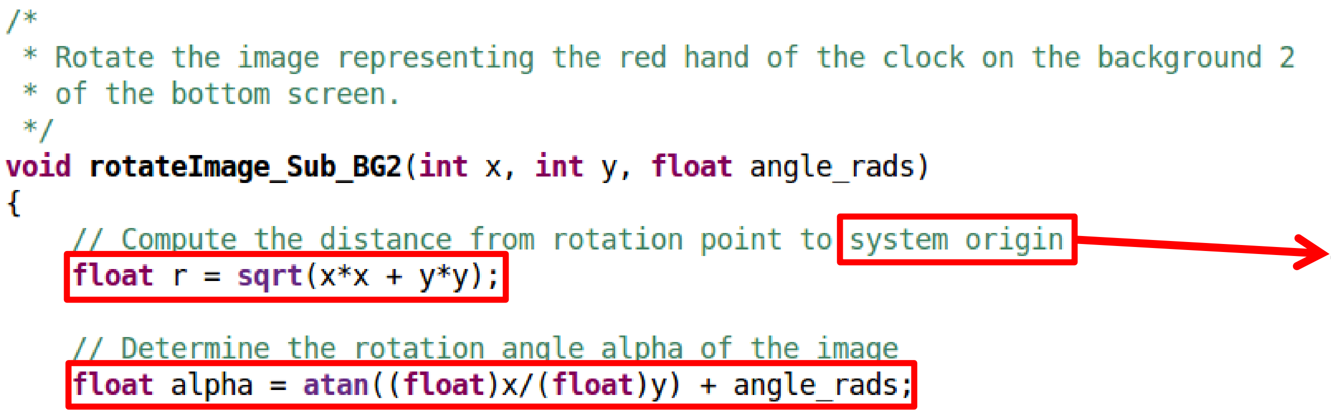


59:52.822

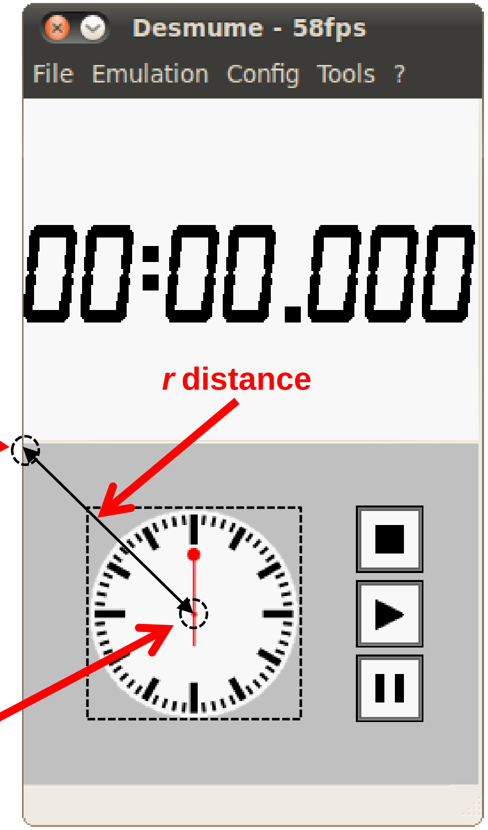
Exercise 3: Configure 2nd Background in SUB Engine

- During the execution of the countdown timer, the red hand of the clock must be rotating (clockwise). Complete the function **rotateImage_Sub_BG2()** in the file **clk_hand.c** following the given comments.

  1. Compute the distance from rotation point to system origin.
  2. Determine the rotation angle alpha of the image.



```
/*
 * Rotate the image representing the red hand of the clock on the background 2
 * of the bottom screen.
 */
void rotateImage_Sub_BG2(int x, int y, float angle_rads)
{
    // Compute the distance from rotation point to system origin
    float r = sqrt(x*x + y*y);

    // Determine the rotation angle alpha of the image
    float alpha = atan((float)x/(float)y) + angle_rads;
```



r distance

Rotation center (x , y)

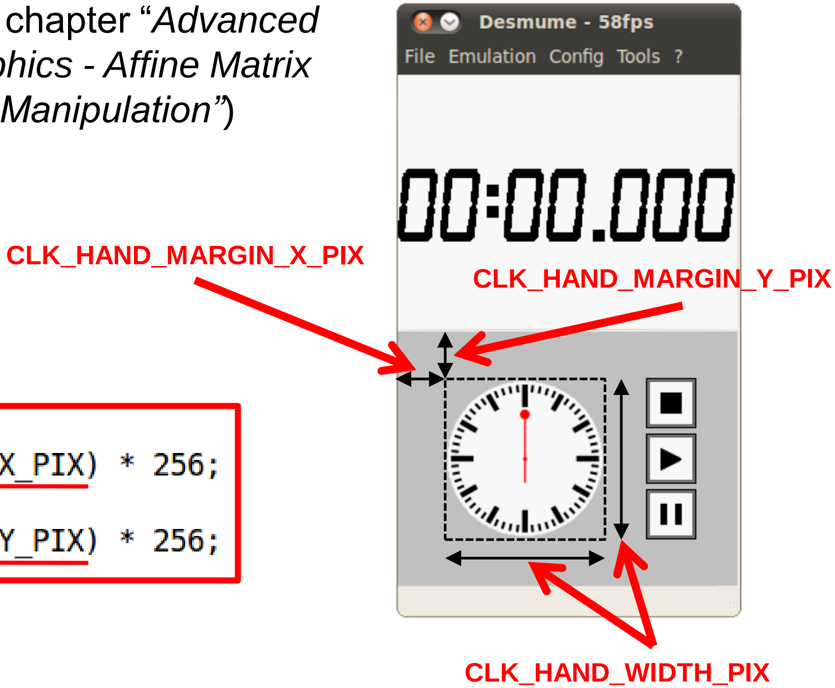
(See chapter "*Advanced Graphics - Affine Matrix Manipulation*")

Exercise 3: Configure Affine Matrix

- To obtain the picture on the right, the background 2 of the bottom screen has be configured, by using affine matrix transformations.
  1. Define the image rotation matrix
  2. Determine the image translation

(See chapter "*Advanced Graphics - Affine Matrix Manipulation*")

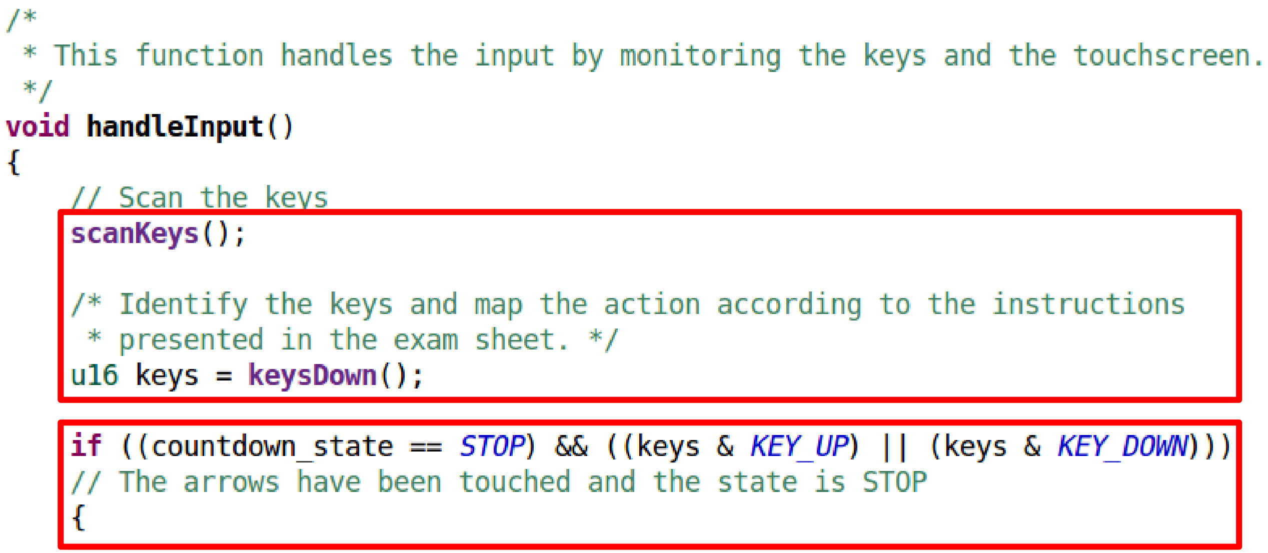
```
  // Image rotation matrix
  REG_BG2PA_SUB = cos(angle_rads) * 256; //xdx
  REG_BG2PB_SUB = sin(angle_rads) * 256; //xdy
  REG_BG2PC_SUB = -sin(angle_rads) * 256; //ydx
  REG_BG2PD_SUB = cos(angle_rads) * 256; //ydy

  // Image translation
  // dx with offset CLK_HAND_MARGIN_X_PIX
  REG_BG2X_SUB = (x - r*sin(alpha) - CLK_HAND_MARGIN_X_PIX) * 256;
  // dy with offset CLK_HAND_MARGIN_Y_PIX
  REG_BG2Y_SUB = (y - r*cos(alpha) - CLK_HAND_MARGIN_Y_PIX) * 256;
}
```



NOTE: Do not forget to use the constants: CLK_HAND_MARGIN_X_PIX and CLK_HAND_MARGIN_Y_PIX, defined inside the header file *clk_hand.h*, in order to center the hand inside the dial of the clock

Exercise 4: Use of Control Pad

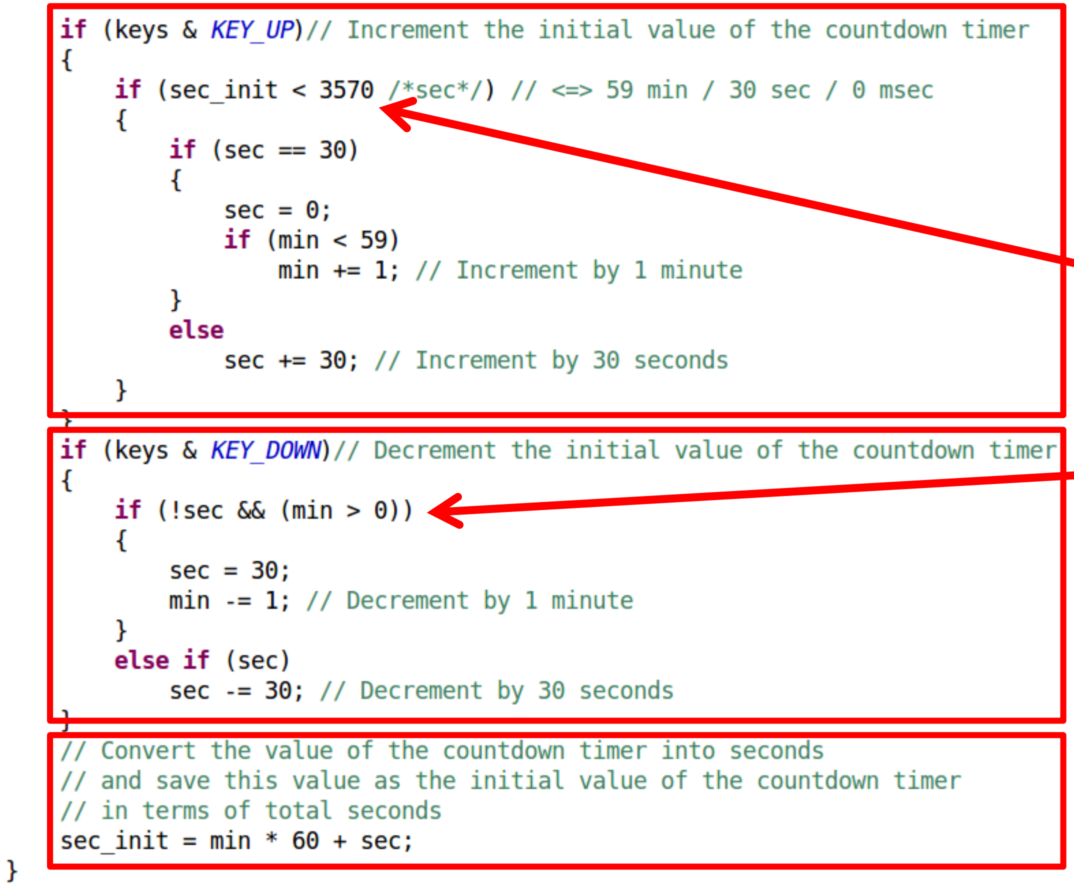
- The two vertical arrows (UP and DOWN keys) will be used to increment or decrement the initial value of the countdown timer. This increase/decrease has to be performed on the minutes and seconds with a step of **+/-30 seconds**, when the countdown timer is in **STOP mode** (only). Complete the function **handleInput()**, in the file **irq_management.c** following the given comments.
  1. Scan, identify the keys and map the action
  2. The arrows have been touched and the state is STOP



```c
/*
 * This function handles the input by monitoring the keys and the touchscreen.
 */
void handleInput()
{
    // Scan the keys
    scanKeys();

    /* Identify the keys and map the action according to the instructions
     * presented in the exam sheet. */
    u16 keys = keysDown();

    if ((countdown_state == STOP) && ((keys & KEY_UP) || (keys & KEY_DOWN)))
    // The arrows have been touched and the state is STOP
    {
```

Exercise 4: Map the Control Pad Actions

- Increment or decrement the initial value of the countdown timer:
  1. Increment the initial value of the countdown timer
  2. Decrement the initial value of the countdown timer
  3. Convert the value of the countdown timer into seconds and save it

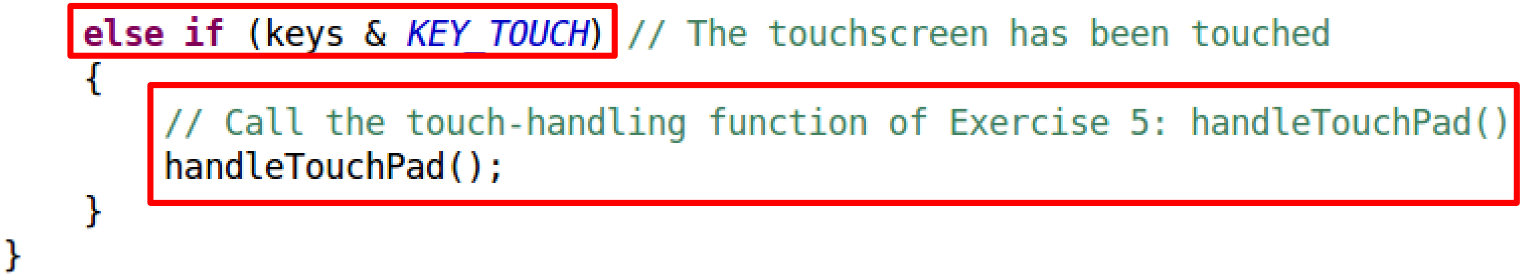


```
if (keys & KEY_UP)// Increment the initial value of the countdown timer
{
    if (sec_init < 3570 /*sec*/) // <=> 59 min / 30 sec / 0 msec
    {
        if (sec == 30)
        {
            sec = 0;
            if (min < 59)
                min += 1; // Increment by 1 minute
        }
        else
            sec += 30; // Increment by 30 seconds
    }
}
if (keys & KEY_DOWN)// Decrement the initial value of the countdown timer
{
    if (!sec && (min > 0))
    {
        sec = 30;
        min -= 1; // Decrement by 1 minute
    }
    else if (sec)
        sec -= 30; // Decrement by 30 seconds
}
// Convert the value of the countdown timer into seconds
// and save this value as the initial value of the countdown timer
// in terms of total seconds
sec_init = min * 60 + sec;
}
```

Be careful with the boundaries/limits up and down [00:00.000 ; 59:30.000]

Exercise 4: Map the Touchscreen Actions

- The key pressed detection on the touchpad has to be also implemented:
  1. Detect if the touchscreen has been touched
  2. Call the touch-handling function of Exercise 5: handleTouchPad()
     (This function is used in the next exercise)

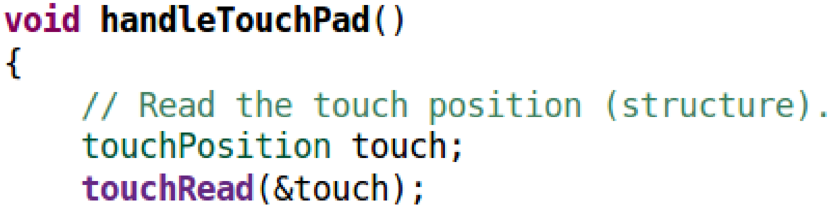


```
    else if (keys & KEY_TOUCH) // The touchscreen has been touched
    {
        // Call the touch-handling function of Exercise 5: handleTouchPad()
        handleTouchPad();
    }
}
```

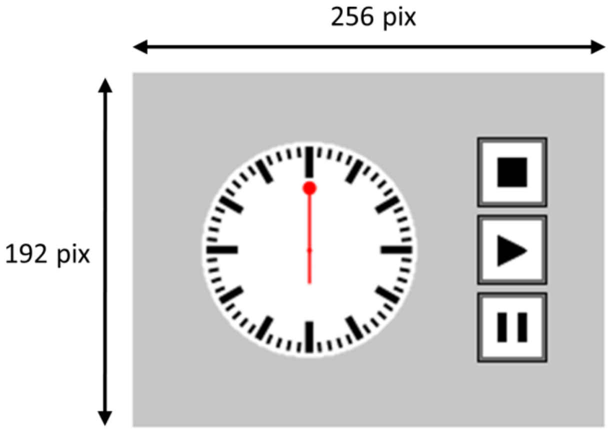
Exercise 5: Touchscreen management

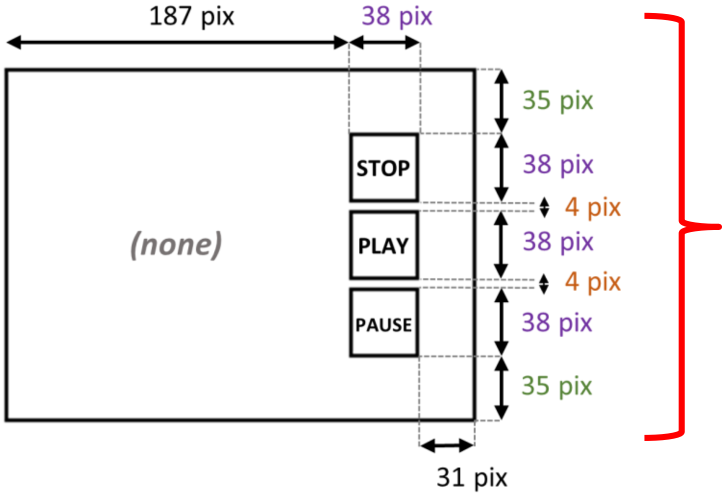
- Complete the function **handleTouchPad()** in the file **irq_management.c** following the given comments to read the touchscreen and map the touched coordinates to the corresponding control: **STOP**, **PLAY**, **PAUSE**.

1. Read the touchscreen if it has been touched
2. Get the coordinates of the touched position



```
void handleTouchPad()
{
    // Read the touch position (structure).
    touchPosition touch;
    touchRead(&touch);
```



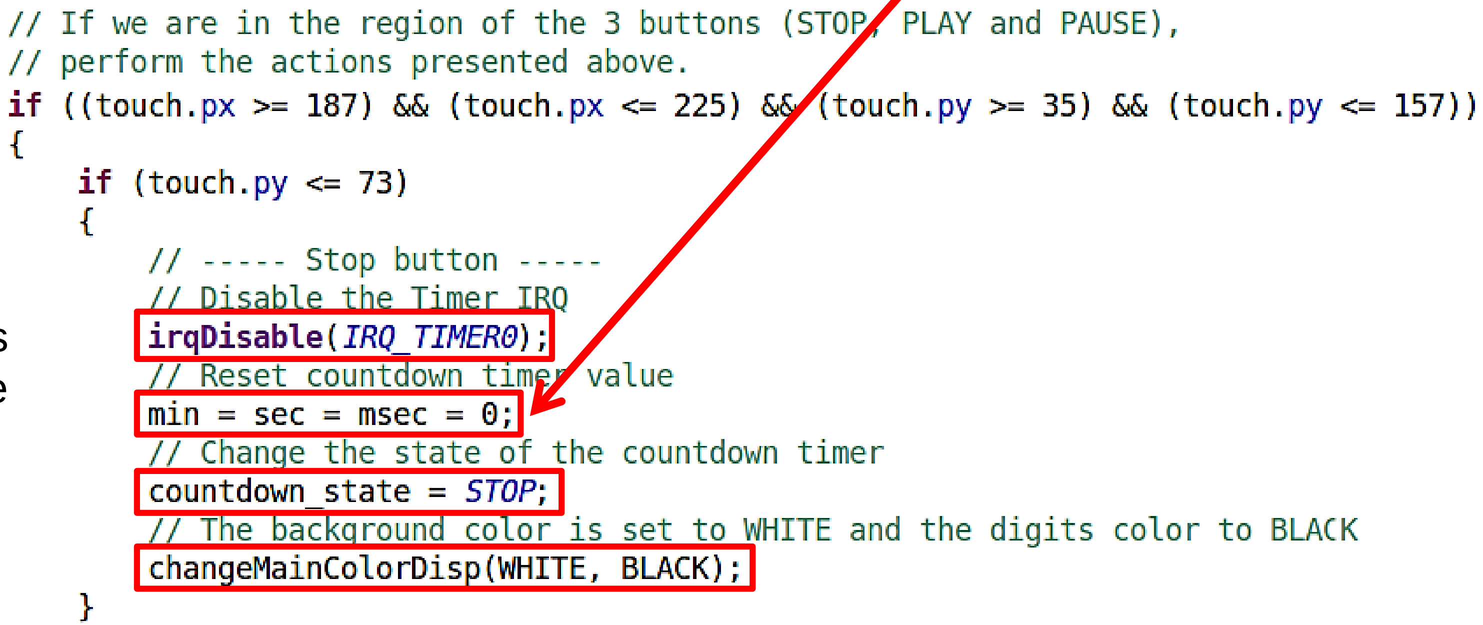


Relative position within the 256x192-pixel image

Exercise 5: Touchscreen management

- Check the region touched by the user and follow the indicated actions.

  1. If **STOP** (square) button/region touched:
     - Disable the countdown timer IRQ (IRQ_TIMER0).
     - Reset timer value (min, sec, msec) to zero. (WARNING: Do not reset **sec_init**)
     - Change the state of **countdown_state**.
     - Set background color to WHITE and the digits color to BLACK on the MAIN screen, with the function: **changeMainColorDisp(uint16 bg_color, uint16 digit_color).**

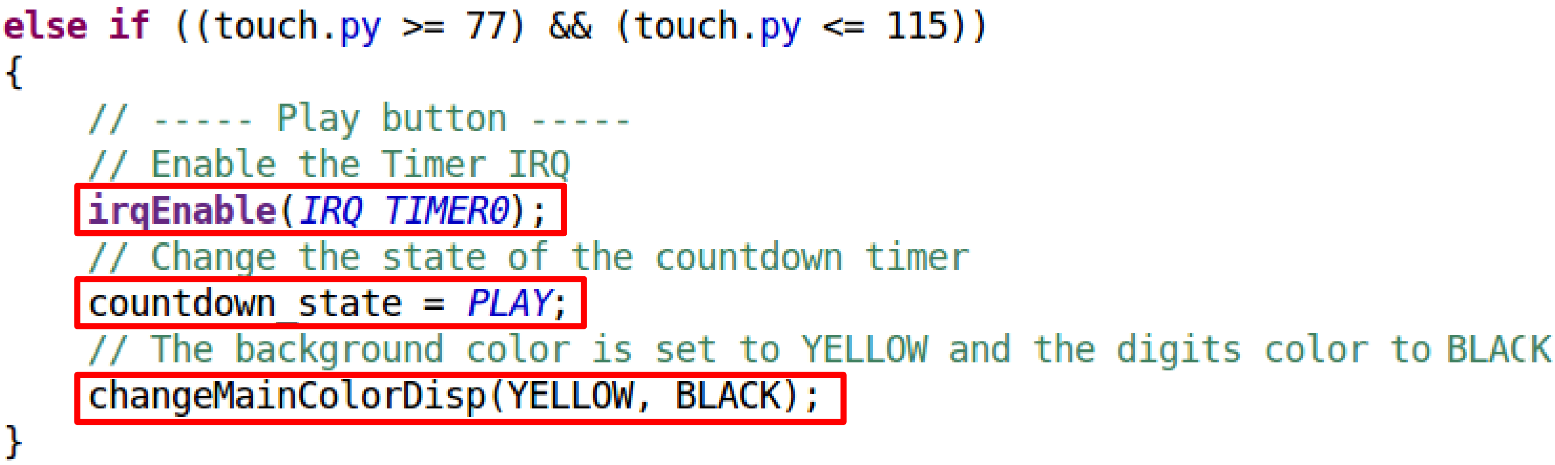


```
// If we are in the region of the 3 buttons (STOP, PLAY and PAUSE),
// perform the actions presented above.
if ((touch.px >= 187) && (touch.px <= 225) && (touch.py >= 35) && (touch.py <= 157))
{
    if (touch.py <= 73)
    {
        // ----- Stop button -----
        // Disable the Timer IRQ
        irqDisable(IRQ_TIMER0);
        // Reset countdown timer value
        min = sec = msec = 0;
        // Change the state of the countdown timer
        countdown_state = STOP;
        // The background color is set to WHITE and the digits color to BLACK
        changeMainColorDisp(WHITE, BLACK);
    }
```

Identify buttons & discard white region

Exercise 5: Touchscreen management

- Check the region touched by the user and follow the indicated actions.
  2. If **PLAY** (triangle) button/region touched:
     - Start countdown timer by enabling its IRQ (IRQ_TIMER0).
     - Change the state of **countdown_state**.
     - Set background color to YELLOW and the digits color to BLACK on the MAIN screen, with the function: **changeMainColorDisp(uint16 bg_color, uint16 digit_color).**

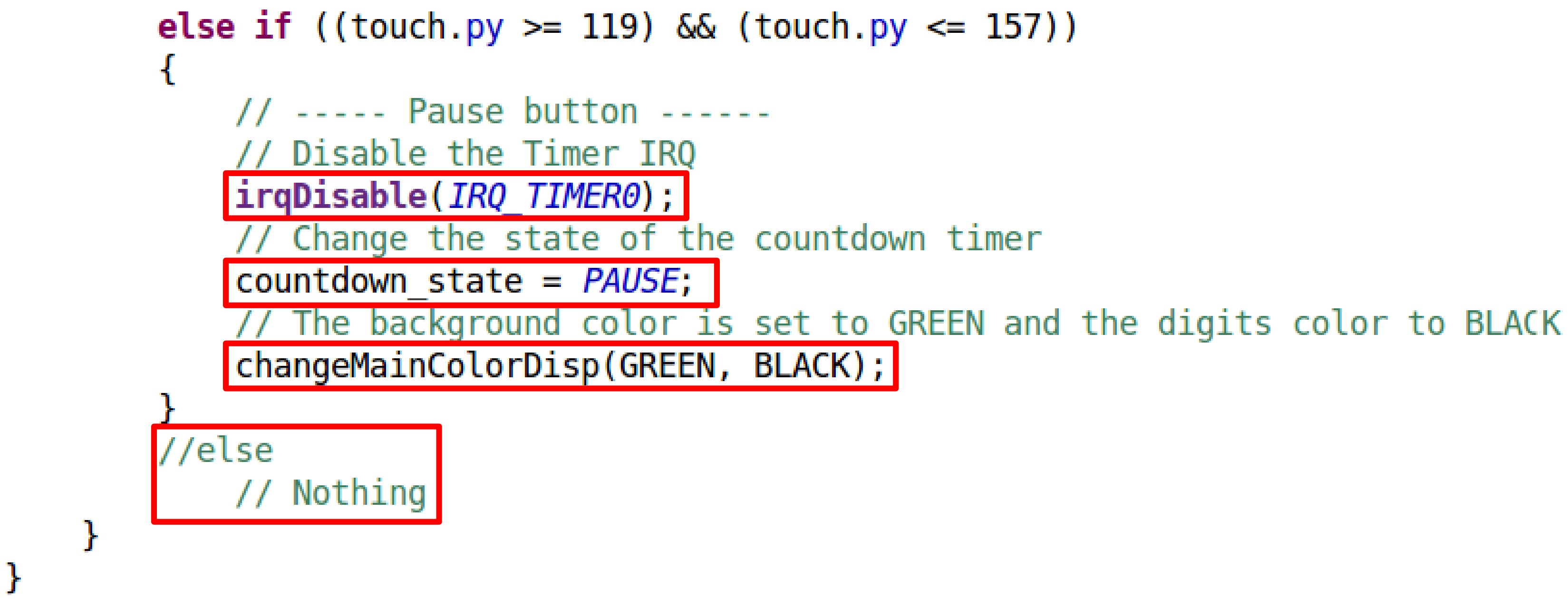


```
else if ((touch.py >= 77) && (touch.py <= 115))
{
    // ----- Play button -----
    // Enable the Timer IRQ
    irqEnable(IRQ_TIMER0);
    // Change the state of the countdown timer
    countdown_state = PLAY;
    // The background color is set to YELLOW and the digits color to BLACK
    changeMainColorDisp(YELLOW, BLACK);
}
```

Exercise 5: Touchscreen management

- Check the region touched by the user and follow the indicated actions.

3. If **PAUSE** (double bars) button/region touched:
   - Pause countdown timer by disabling its IRQ (IRQ_TIMER0).
   - Change the state of **countdown_state**.
   - Set background color to GREEN and the digits color to BLACK on the MAIN screen, with the function: **changeMainColorDisp(uint16 bg_color, uint16 digit_color).**
4. If the touch position is not in one of those 3 regions, do nothing



```
        else if ((touch.py >= 119) && (touch.py <= 157))
        {
            // ----- Pause button ------
            // Disable the Timer IRQ
            irqDisable(IRQ_TIMER0);
            // Change the state of the countdown timer
            countdown_state = PAUSE;
            // The background color is set to GREEN and the digits color to BLACK
            changeMainColorDisp(GREEN, BLACK);
        }
        //else
            // Nothing
    }
}
```