Dynamic Threads & LCD Interfacing

EEL 4745C: Microprocessor Applications II

Fall 2022

Md Jahidul Islam

Lecture 6

G8RTOS: Interfacing LCDs

Lab4: Dynamic Threads and LCD Interfacing

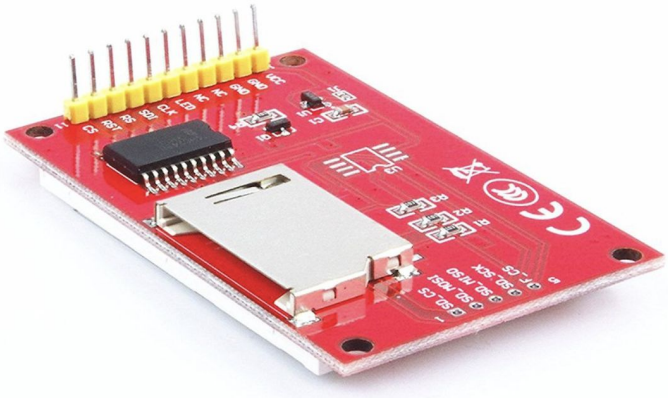
- Part A: Interfacing LCD
  - Complete driver functions
- Part B.1: Priority Scheduler
  - Incorporate priority features into the round-robin algorithm
- Part B.2: Dynamic Thread Features
  - Thread creation and destruction
- Part B.3: Aperiodic Event Threads
  - Relocate ISRs interrupt vector

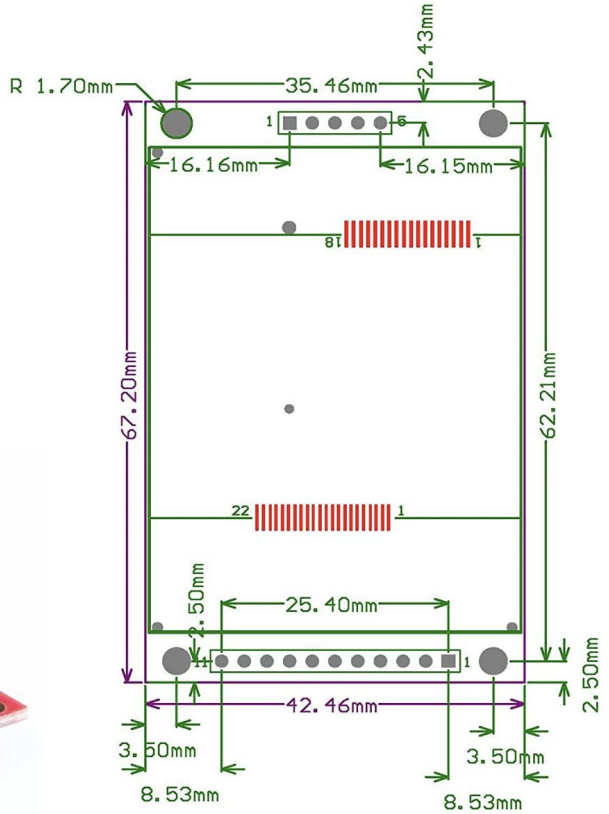
Demos are due: Oct 24 - Nov 03

Solutions upload: Nov 4th

Our TFT LCD

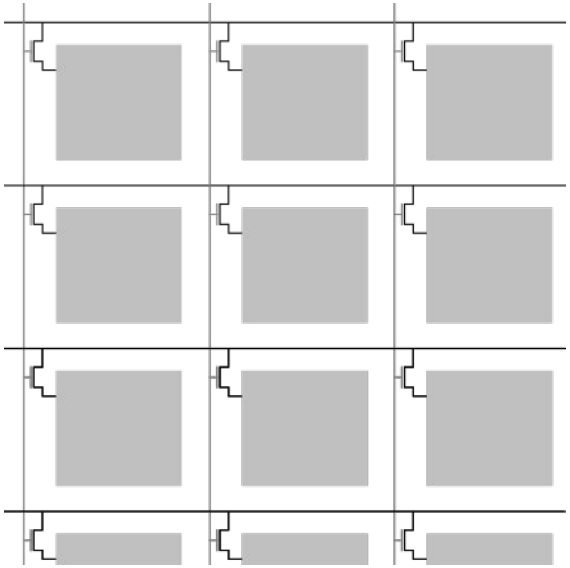
- Single chip TFT LCD display
- 240x320 dot resolution (RGB)
- Internal 17.28KB graphic RAM
- System interfaces
  - parallel 8-/9-/16-/18-bit data bus MCU interface
  - 6-/16-/18-bit data bus RGB interface
  - 3-/4-line serial peripheral interface (SPI)
- Touch screen interface: SPI





TFT LCD: Overview

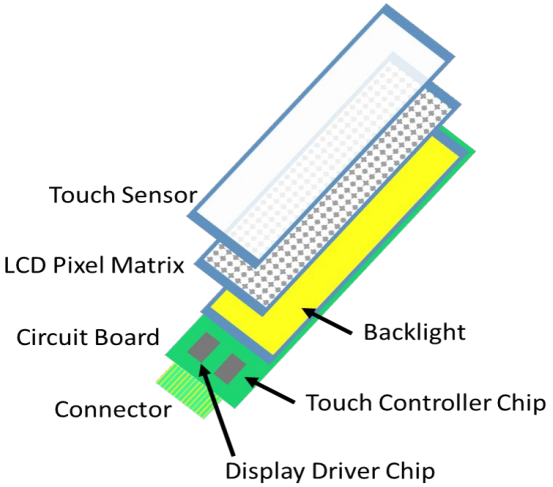
- LCD Displays stands for Liquid Crystal Displays
- TFT Displays stands for Thin Film Transistor
  - Mature technology with capacitors and transistors
  - Categorically referred to as active-matrix LCDs.
  - These LCDs can hold back some pixels while using other pixels, hence they operate at a very low power
- Cannot release color themselves; rely on extra light source in order to display (backlight)
- Widely used in embedded systems



- References
  - https://en.wikipedia.org/wiki/Thin-film-transistor_liquid-crystal_display
  - https://www.orientdisplay.com/knowledge-base/lcd-basics/lcd-vs-tft-ips-led-oled-display/
  - https://grobotronics.com/images/datasheets/xpt2046-datasheet.pdf

TFT LCD: Building Blocks

- The display is constructed on top of a circuit board which houses the connector and any controller chips that are necessary.
- The backlight is located on top of the circuit board, with the pixel matrix sitting on top of the backlight.
  - The backlight is necessary for TFT LCD displays to allow the display to be seen.
  - Without a backlight, a color TFT LCD will show no image.
- Pixel matrix is comprised of an array of pixels in height and width of a certain color depth that make up the display.
- The touch sensor is optional and is located at the top of the stackup.
- References
  - https://www.digikey.com/htmldatasheets/production/1640716/0/0/1/ili9341-datasheet.html
  - https://cdn-shop.adafruit.com/datasheets/ILI9325.pdf
  - https://www.adafruit.com/product/1770



Part A: Important Driver Functions

These functions are already implemented for you; see BSP drivers: **`ILI9341_Lib.c`** and **`ILI9341_Lib.h`**

- **LCD_Init()**: Enable/Initialize SPI/GPIO Peripherals
- **PutChar()**: Outputs a character to the display at some coordinate

  // This utilizes the ASCII library
- **LCD_Text()**: Outputs a string to the display at some coordinate
- **LCD_WriteIndex()**: Sets the address for the register we want to write.
- **LCD_WriteData()**: Writes 16-bit data to the register that is specified by the LCD_WriteIndex() function
- **LCD_Write_Data_Only()**: Sends only data (useful for continuous transmission)
- **TP_ReadXY()**: Reads the tapped X and Y coordinates from the LCD.

Part A: Important Driver Functions

You will need to implement these functions:

- **LCD_DrawRectangle()**: Draw a rectangle with a specified color.
- **LCD_Clear()**: Clear the screen with a specified color
- **LCD_SetPoint()**: Draw one pixel with specified coordinate and color.
- **LCD_WriteReg()**: Write data to the specified register.
- **LCD_SetCursor()**: Place the cursor at the specified coordinate.
- **LCD_PushColor()**: Set a pixel on the LCD to a specific color.
- **LCD_SetAddress()**: Set the draw area of the LCD.

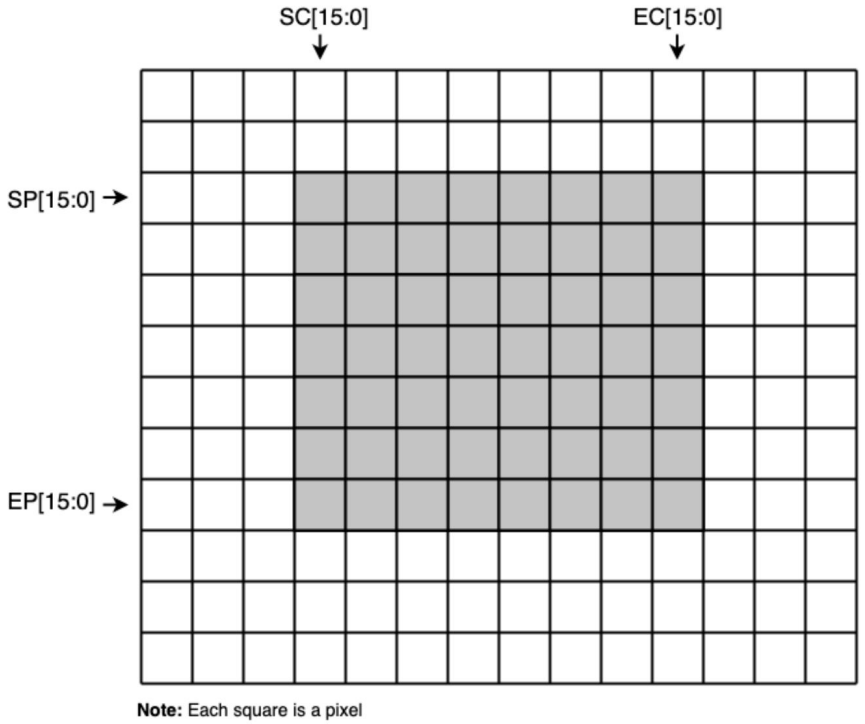
Please look into the manual/datasheet to correctly implement these.

Feel free to implement any other functions you might need!

Example: Writing Pixels

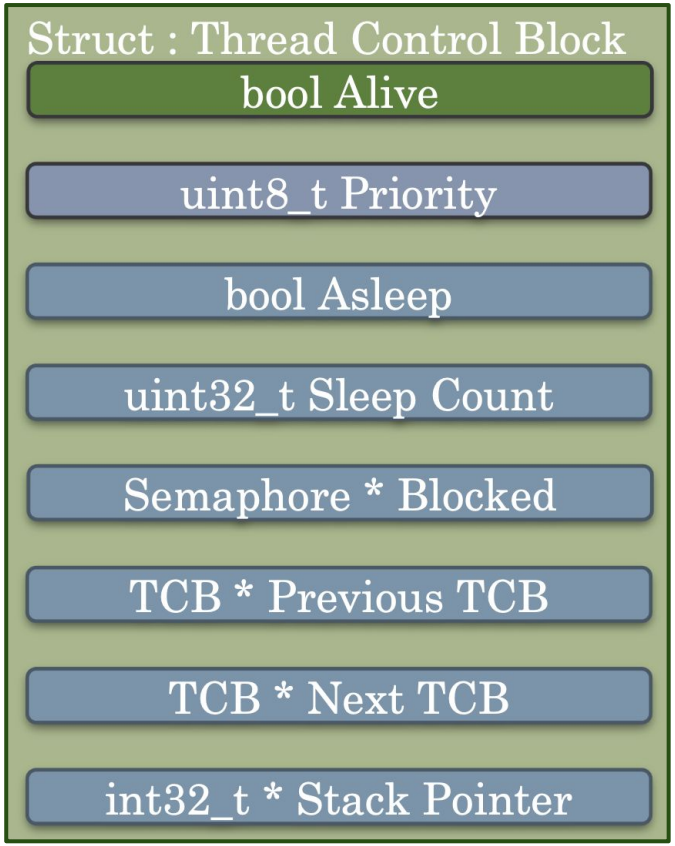
- Define a rectangular window of frame memory
- Use *Column Address Set (0x2A)* and *Page Address Set (0x2B)* for setting: Start Column (SC), End Column (EC); Start Page (SP) and End Page (EP).

```
LCD_SetAddress(x1, y1, x2, y2){
      LCD_WriteIndex(0x2A);
      LCD_WriteData(x1>>8);
      LCD_WriteData(x1);
      LCD_WriteData(x2>>8);
      LCD_WriteData(x2);

      LCD_WriteIndex(0x2B);
      LCD_WriteData(y1>>8);
      LCD_WriteData(y1);
      LCD_WriteData(y2>>8);
      LCD_WriteData(y2);

      LCD_WriteIndex(0x2C); //memory write
}
```



Part B.1: Priority Scheduling

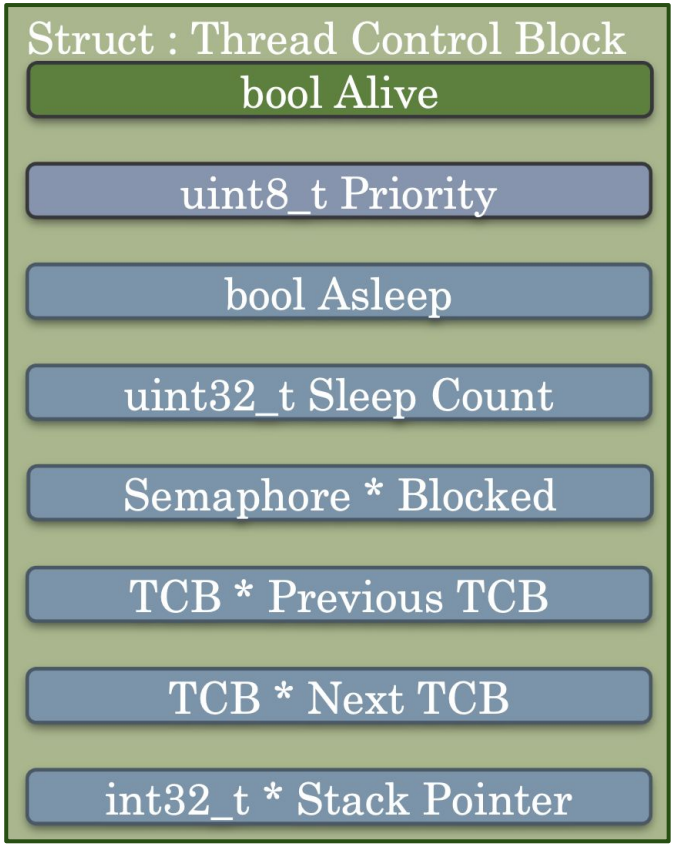
- Why priority scheduling Instead of round-robin:
  - You might need something like a background thread that always needs to execute as soon as possible.
  - For example, after tapping the LCD screen, a background thread might need to be executed first to update the global status as soon as possible.
- How to implement:
  - Maintain a variable `currentMaxPriority` to store the priority level of the current running thread.
    - Initialize to 256 (lowest priority); highest priority is 0.
  - While scheduling another thread, check if its priority is less than `currentMaxPriority`.
  - Modify G8RTOS_Launch to choose the thread with the highest priority to run first.



Part B.1: Priority Scheduling

Priority check pseudo code

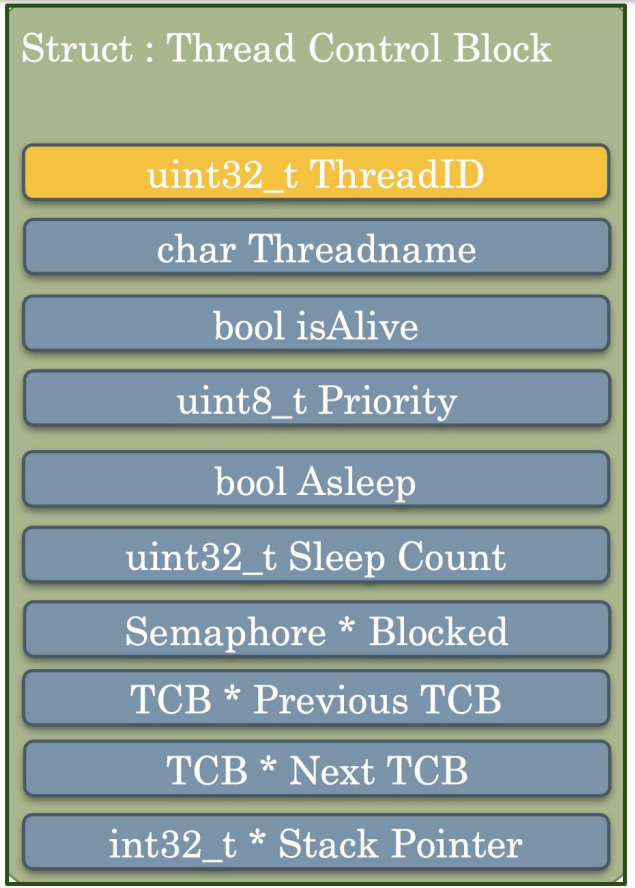
```
/* Priority of potential next thread to run */
uint8_t nextThreadPriority = UINT8_MAX;

for(loop)
{
     /* Check if Thread is blocked or asleep */
     if !nextThread.issleep() && !nextThread.isblocked()
     {
          /* Check if priority is higher than current max */
          if nextThread.Priority less than nextThreadPriority
          {
               /* Set CurrentlyRunning thread to the next thread to run */
               CurrentlyRunningThread = nextThread
               nextThreadPriority = CurrentlyRunningThread.Priority;
          }
     }

     nextThread = nextThread.nextTCB;
}
```



Part B.2: Dynamic Thread Features

- Dynamic thread creation and destruction
  - Modification: **`AddThread()`**
  - New function: **`KillThread()`** and **`KillSelf()`**

- Implementation:
  - Boolean (`isAlive`) to keep track of status: alive/dead
  - Integer (`threadID`) and character array (`threadName`):
    - To keep track of threads inside of the variable explorer and overall debugging process.
    - Allow every thread to have its unique ID so that the user can request the ID of the thread to be killed.



Part B.2: Dynamic Thread Features

Modifications to `AddThread()`

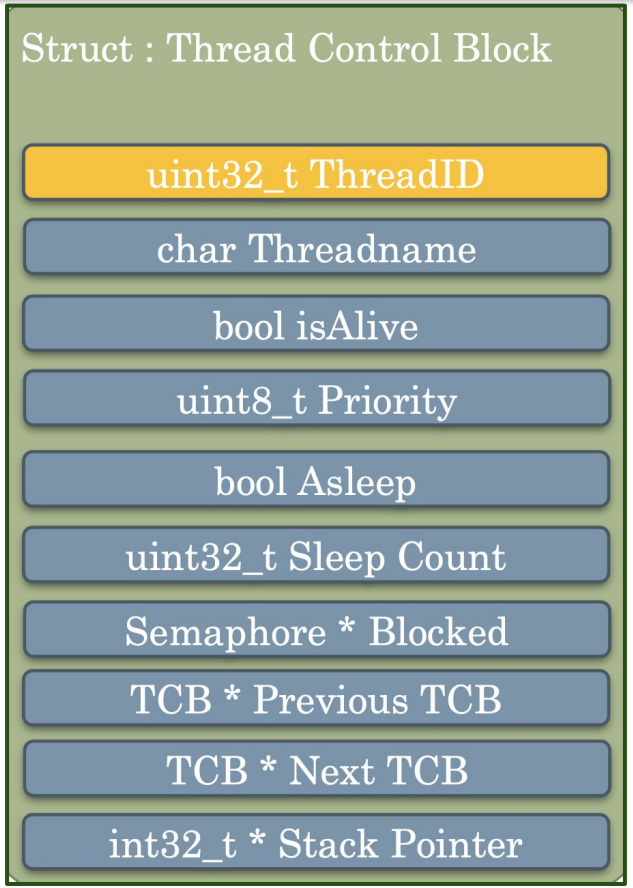
- The `AddThread` function will now take in not only a thread's priority, but also its name/id to initialize.
- Since we want to be able to add a thread while our OS is running, we will need to enter a critical section and exit it prior to returning.

`KillThread(threadId)`

- This function will take in a `threadId`, indicating the thread to kill. It takes care of the boundary conditions (e.g.,: if no threads exist with that ID, only one thread running).

`KillSelf()`

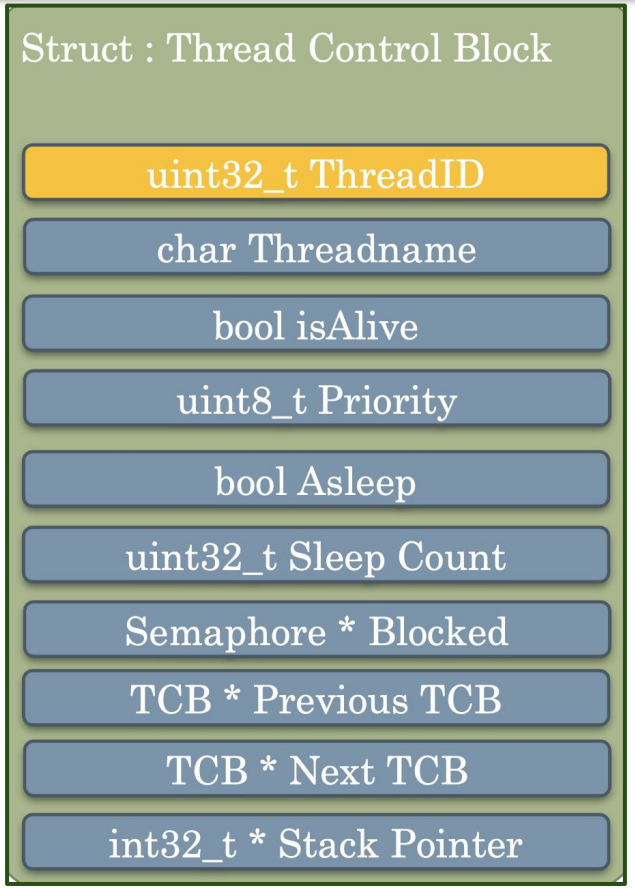
- This function will simply kill the currently running thread.



Example: Killing Threads

`KillThread(threadId)`

- This function will take in a `threadId`, indicating the thread to kill. It takes care of the boundary conditions (e.g.,: if no threads exist with that ID, only one thread running).
- **Procedure**:
  - Enter a critical section
  - Return right error code if there's only one thread running
  - Search for thread with the same threadId
  - Return error code if the thread does not exist
  - Set the threads isAlive bit to false
  - Update thread pointers
  - If thread being killed is the currently running thread, we need to context switch once critical section is ended
  - Decrement number of threads
  - End critical section



Part B.3: Aperiodic Event Thread

Definition: *An event thread with an arrival pattern that lacks a bounded minimum interval between subsequent instances.*

Implementation:

- Whenever you tap (use the LCD touchpad), we need to run an ISR. which is essentially an aperiodic event thread.
- We will need to initialize the appropriate **NVIC (Nested Vectored Interrupt Controller)** registers.
- To add an aperiodic event, we provide it with

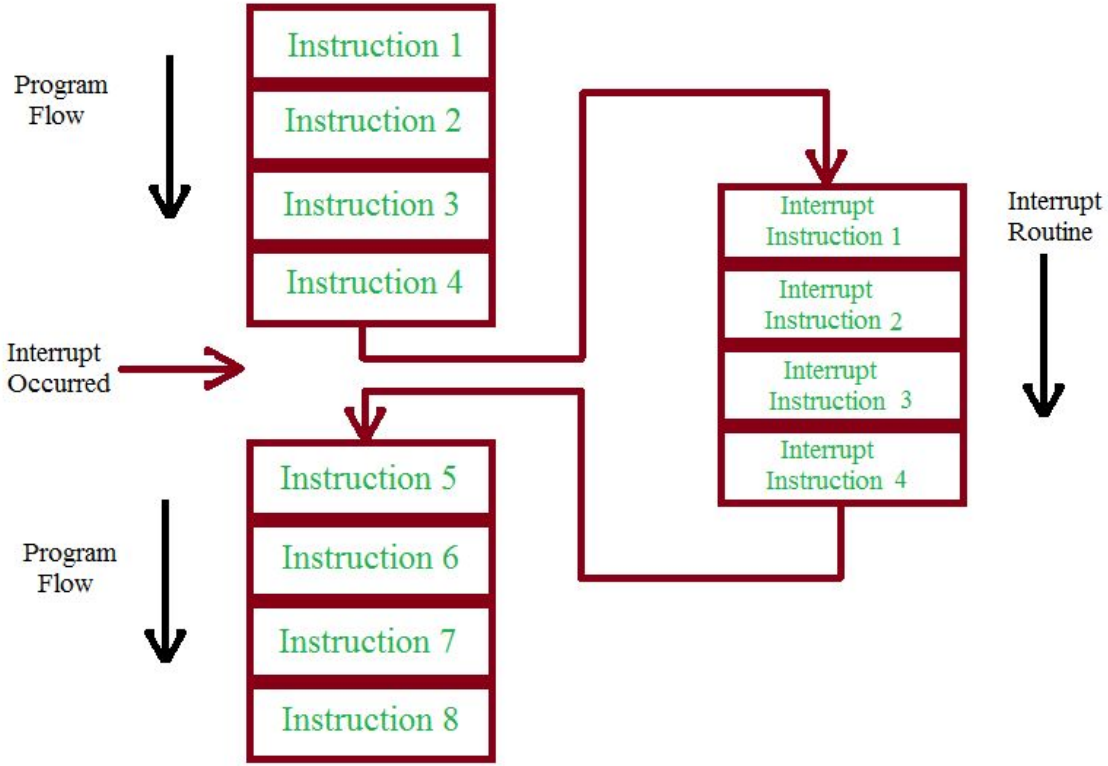
```
AddAPeriodicEvent(void(*AthreadToAdd)(void), uint8_t priority, int32_t IRQn)
```

  - A function pointer that will serve as the ISR, a priority, and
  - The IRQ (interrupt request) number: an assigned location where the computer can expect a particular device to interrupt
- The ISR interrupt vector table must be relocated to SRAM.
  - You can You can do this in G8RTOS_Init()
  - See the Lab-4 manual

What happens when Interrupt Occurs?

Whenever a hard/soft exception occurs

- *A function call and the required response is executed in the form of a piece of code known as a Service routine (SR) or Interrupt Service Routine (ISR).*
- *After that set of instructions in the service, the routine is executed the control shifts back to the main program in which the interrupt occurred.*

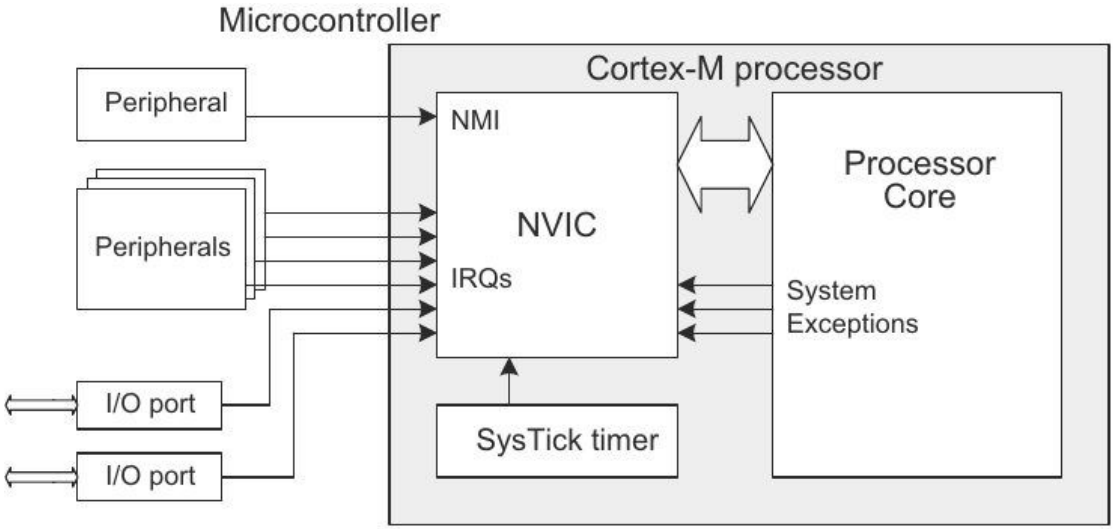


NVIC in ARM Cortex-M

***'Nested'**: processing an interrupt (with higher priority) with in another interrupt (with lower priority).*

- *ARM Cortex-M microcontrollers have 0-255 exceptions/interrupts.*
  - *Each exception has a priority*
  - *System exceptions: 16 (0-15)*
  - *User interrupts 240 (16-255)*

- *The higher priority interrupts always gets to execute before a lower priority interrupts even if the lower priority interrupts occurs earlier.*

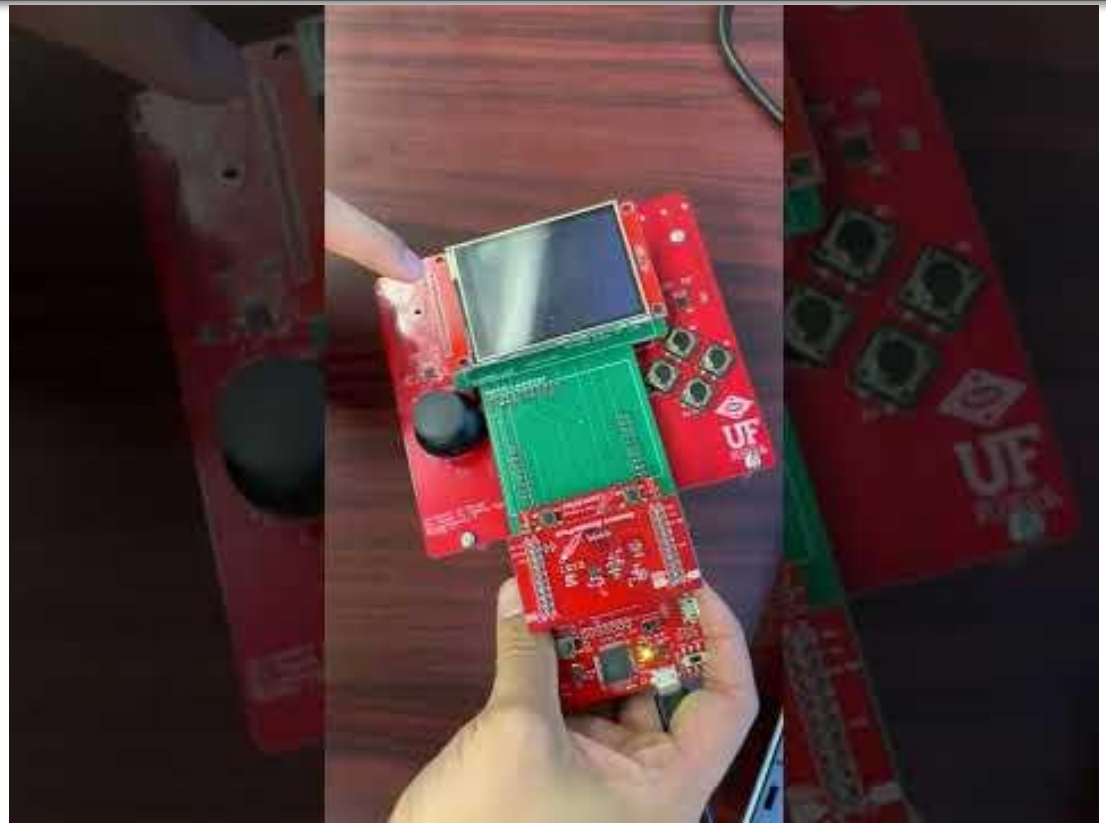


*Read more at: **here** and **here***

G8RTOS Lab4

Lab4: Dynamic Threads and LCD Interfacing

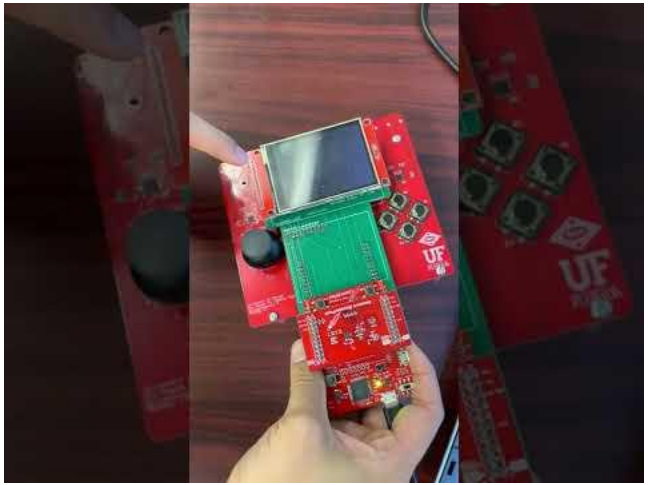
- Part A: Interfacing LCD
  - Complete driver functions
- Part B.1: Priority Scheduler
  - Incorporate priority features into the round-robin algorithm
- Part B.2: Dynamic Thread Features
  - Thread creation and destruction
- Part B.3: Aperiodic Event Threads
  - Relocate ISRs interrupt vector



https://youtu.be/umWUxbx3qZc

Lab4: Rules

- Program will launch with a blank screen waiting for a tap.
- Once touched
  - A ball (4x4 rectangle in our case) should be drawn on the screen with a random color.
  - You may use the `time.h` library for randomness
- Depending on the accelerometer X and Y values, the ball will change directions smoothly.
- Every new ball created should have a random speed
  - Just use a scaling factor for its velocity.
- If any ball is touched, it should be deleted.
- There will be a maximum of 20 balls allowed at any point of time.
- If a ball hits an edge, it should wrap around to the other side.

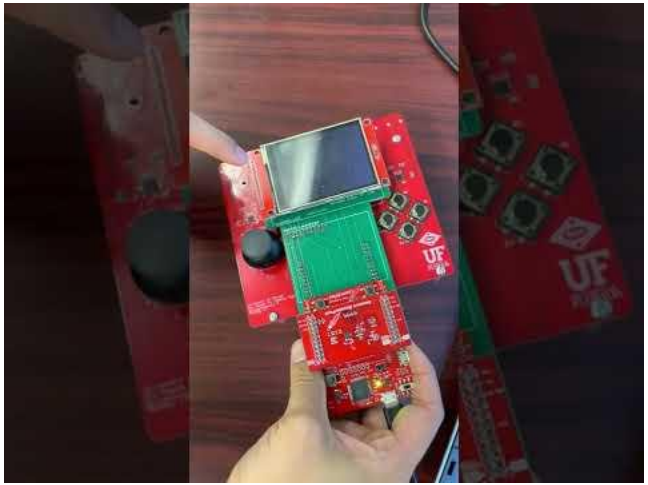


https://youtu.be/umWUxbx3qZc

Lab4: Workflow

Initially, you will have the following threads active:

- **Read Accelerometer:** background thread
- **LCD tap:** aperiodic thread
- **Wait for tap:** background thread
  - Waits for ISR flag, reads touch coordinates, then determines whether to delete or add a ball.
  - If a ball is to be created: write the coordinates to a FIFO and then create a Ball thread.
  - If a ball is to be deleted: wait for any semaphores the ball thread might be using and call `G8RTOS_KillThread` with the ball's `threadID`.
  - Delay for some time to account for screen bouncing before checking the touch flag again.

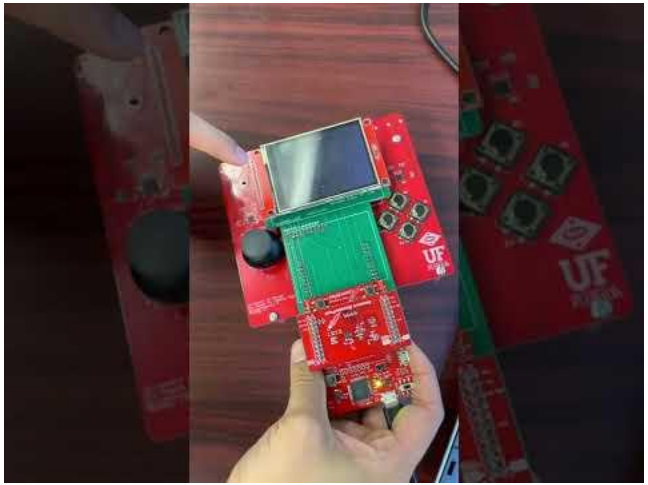


https://youtu.be/umWUxbx3qZc

Lab4: Workflow

Contd..

- **Ball thread:** background thread
  - Finds a dead ball and makes it *alive*.
  - Reads FIFO and initializes coordinates accordingly.
  - Get `threadID` and store it; it is better if you use a struct to hold all information about a ball
    - Color, ID, position, and velocity.
    - Alive or killed/blocked etc.
  - Within while(1):
    - Move its position depending on velocity/acceleration.
    - Update the ball on screen and sleep for some time.



https://youtu.be/umWUxbx3qZc

Lab4: Logistics

- Original deadline:
  - Original demo due: 10/24 - 10/27
  - Late demo (with -10% due): 10/24 - 11/03

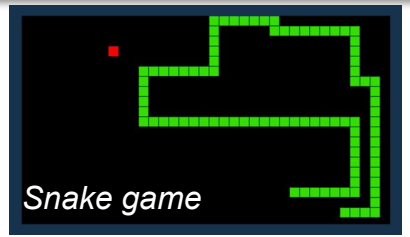
- ***No late penalty*** *till 11/03 (last day for lab-4 demo)*
- ***Quiz #2:*** *on the 10/31-11/03 dates in respective labs*

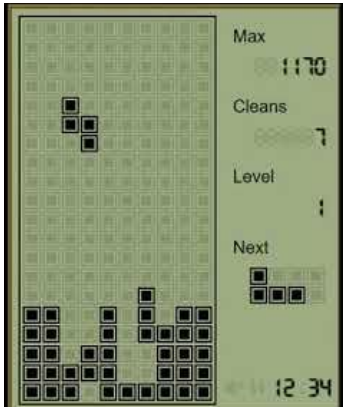
- Driver issues: `Point TP_ReadXY()` function
  - Giving wrong X/Y values
  - Alternative solutions:
    - *Delete the oldest ball, or*
    - *Delete the newest ball.*

- Beagle-boards will be distributed next week
- Some project ideas will be discussed today!

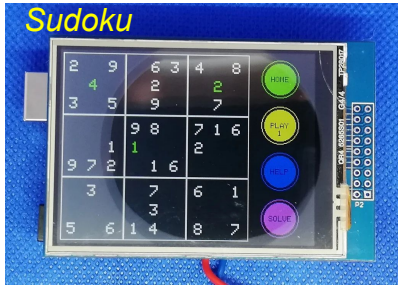
```
Point TP_ReadXY()
{
    Point coor;
    uint8_t highByte, lowByte;

    WriteTP_CS(0);
    SSI0_DR_R = (CHX);              //Reads X data
    while(SSIBusy(SSI0_BASE));
    highByte = SSI0_DR_R;
    while(SSIBusy(SSI0_BASE));
    lowByte = SSI0_DR_R;
    while(SSIBusy(SSI0_BASE));

    coor.x = highByte << 8;             /* Read D8..D15          */
    coor.x |= lowByte;              /* Read D0..D7           */
    coor.x >>= 4;                   //Accounts for offset and scales down
    coor.x -= 190;
    coor.x *= (ADC_X_INVERSE * MAX_SCREEN_X);

    SSI0_DR_R = (CHY);              //Reads Y data
    while(SSIBusy(SSI0_BASE));
    highByte = SSI0_DR_R;
    while(SSIBusy(SSI0_BASE));
    lowByte = SSI0_DR_R;
    while(SSIBusy(SSI0_BASE));

    coor.y = highByte << 8;             /* Read D8..D15          */
    coor.y |= lowByte;              /* Read D0..D7           */
    coor.y >>= 4;                   //Accounts for offset and scales down
    coor.y -= 140;
    coor.y *= (ADC_Y_INVERSE * MAX_SCREEN_Y);
    WriteTP_CS(1);

    return coor;
}
```

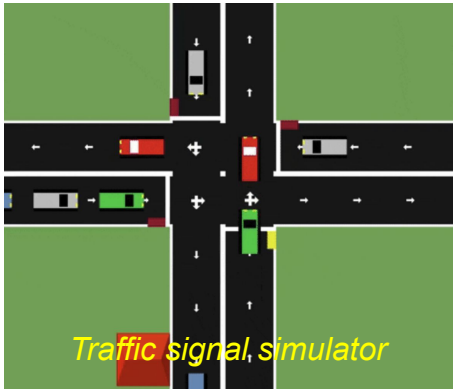
Some Project Ideas

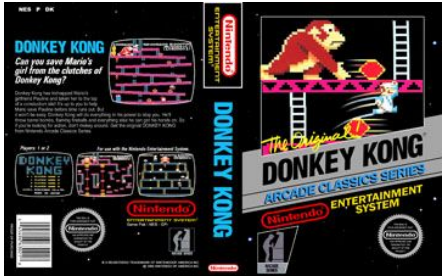




Tetris

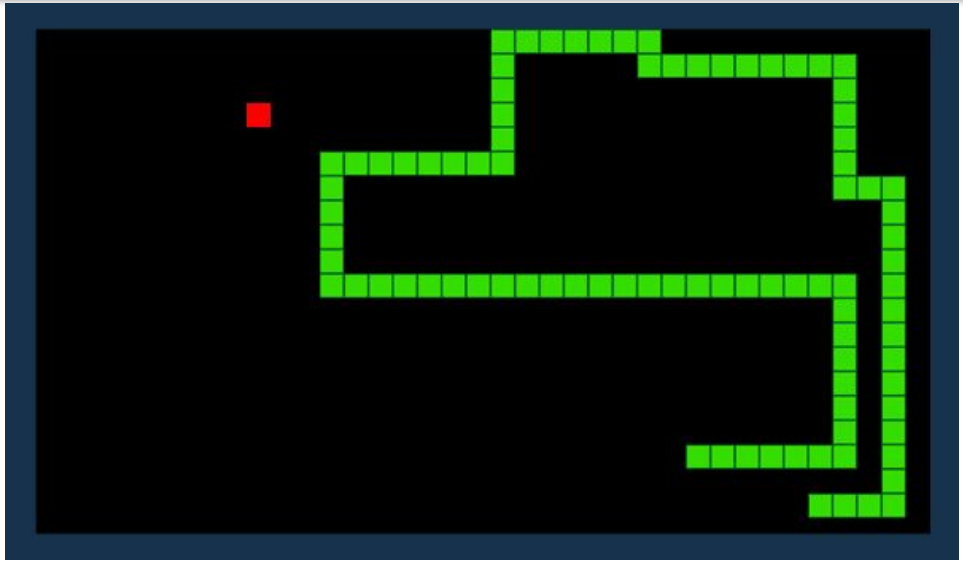






Snake Game: RTOS Design

- **Background Threads**
  - Joystick reader
  - Snake (linked list)
  - Board (2D array)
- **Aperiodic event thread**
  - Win condition
  - Lose condition
- **Periodic thread**
  - Draw canvas
- **Buffers**
  - Joystick values (FIFO)
  - Snake body points (linked list)
  - Point tables or difficulty levels?



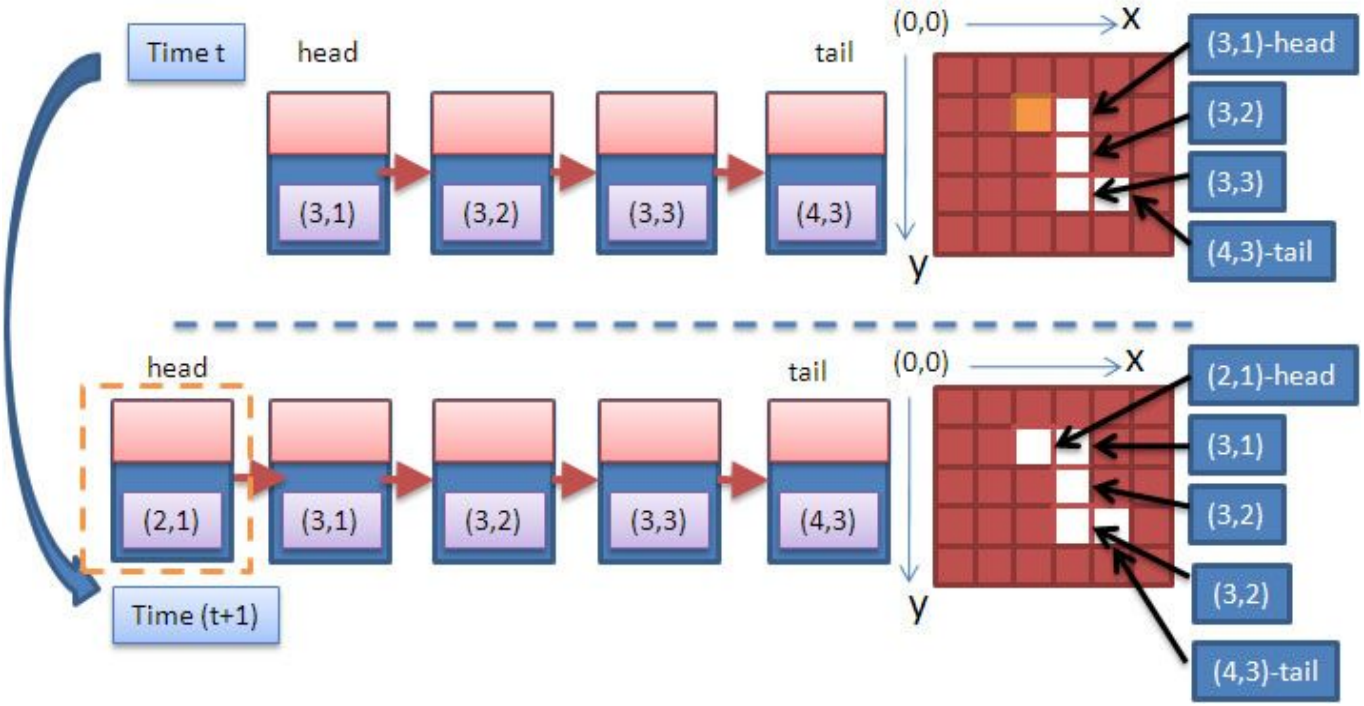
Semaphores

- Interfacing: joystick, leds, display
- Shared Memory: FIFOs, snake body buffers
- Database: levels, tables, scores

Scheduling algorithm: Round-robin + priority

Snake Game: Data Structures

- **Snake**
  - Linked list of (x, y)
  - Head grows based on joystick values
- **Board**
  - 2D array (0/1)
  - Updates snake and goal values
  - Updates goal
- **Move**
  - Joystick FIFO



Snake Game: Memory Management

- **Joystick buffer:** standard FIFO; same as Lab-3
- **Snake body points:** singly connected linked-list
  - Head grows, tail remains static
  - Opposite direction: reverse linked list?
- **Canvas**
  - Display board to the LCD through registers
  - Refresh and update periodically
  - Take care of win/lose situation (higher priority event)
- **Database**
  - Game levels, templates, point tables, etc.