2303ENG Microprocessor Techniques Project Report

# Introduction

The purpose of the project is to demonstrate knowledge of microcontroller programming using embedded C by producin a working Pong game on an LCD screen.

In order to achieve a successful result with this programing project it is necessary to interface the Tiva-C series TM4C123G board to the TG12864H3-05A LCD board using GPIO pins for the communication between the Tiva chip and the user terminal using the UART subsystem.

After the mentioned system initialisations are tested. In order to make this project successful interrupt routines need to be utilised. Interrupts in microcontroller systems are more effiffient than continuous polling.

This project uses and interrupt function based on tiva-c files rather than lower level assembly language.

In order to track the progress of the project an incremental test driven approach was used. The program starts at the main() function being a C program. To test the successful configuration of GPIO pins with the LCD, test functions named *TestDrawing* and *TestDynamics()* were created to establish the presence of communication between the Tiva and the LCD, now commented out at lines 602 and 603 in pong.c, test driven development was used as emphasised throughout the labs to ensure communication between devices are established.

To test successful connection and correct wiring between the Tiva board and the correct setup of the interrupt function. This project uses the LED on PortF pin 2 to indicate successful character transfer. For the LED to flash the interrupt function must be working, interrupts enables as well as GPIOs enabled.

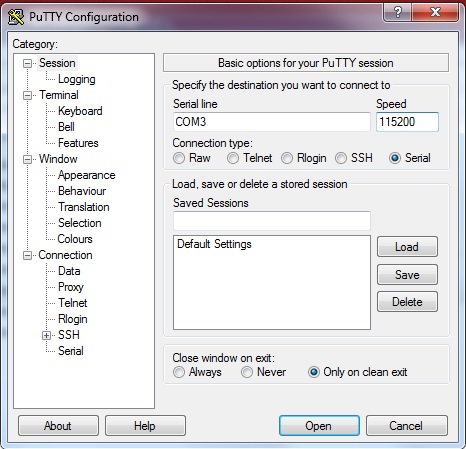
# Putty Setup

For the serial connection between of the UART system needs to be set to the following specifications:

Baud Rate: 115200

Data bits: 8

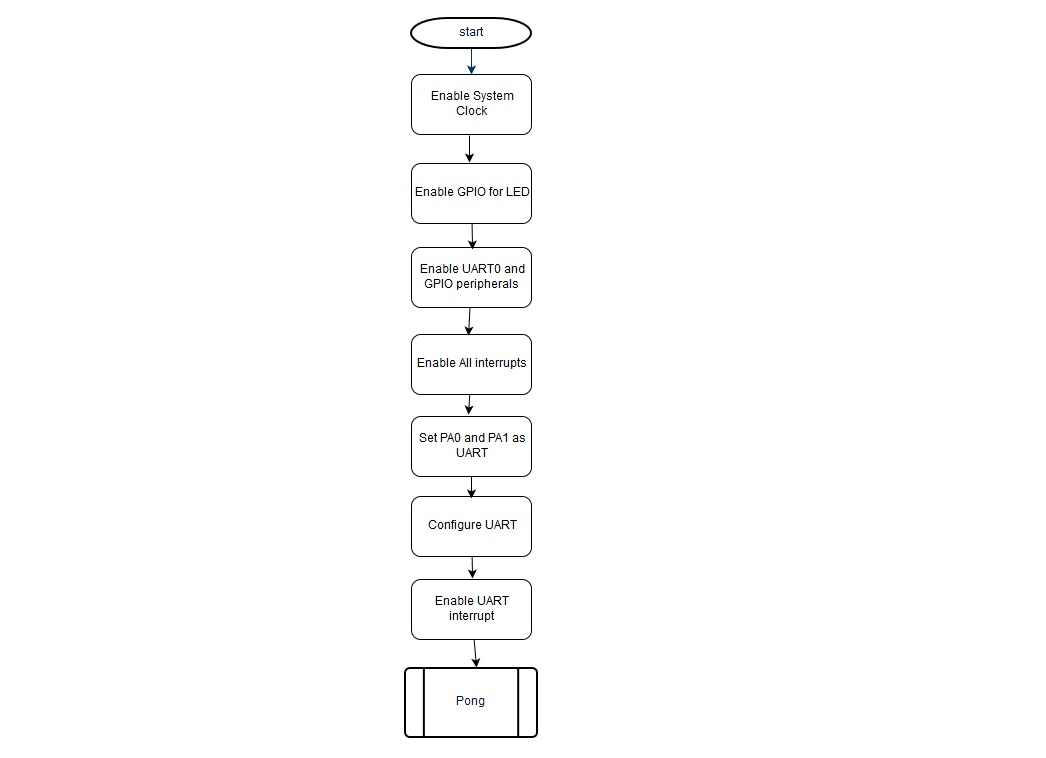
No parity.



# LCD Connection

|  |  |
| --- | --- |
| LCD Connection | Tiva Connection |
| B- | GND |
| G- | PE1 |
| A+ | 3.3V |
| R- | PE3 |
| GND | GND |
| VDD | +3.3 |
| SD | PD3 |
| SCLK | PD0 |
| A0 | PD2 |
| /RST | PF1 |
| /CS | PD1 |

# Program Flow Chart



Pong here refers to the “while(1)” loop in the main function.

# Interrupts

Interrupts for UART input via the keyboard is enabled by using functions from prewritten header files provided by texas instruments.

In this project the interrupt status is obtained from the predefined function “ROM\_UARTIntStatus(UART0\_BASE, true)” line 449 of pong.c , the function is defined in the “rom.h” file. Where the function wait for UART0\_BASE (address 0x4000C000), defined in hw\_memmap.h line 61 to contain data, that is a character from the keyboard via the serial UART.

With this design the system’s interrupt handler stores any character that is passed from the keyboard to the receive FIFO. In order to deal with the interrupt, after an interrupt has occurred it is cleared with the “ROM\_UARTIntClear” function defined in the rom.h file, line 6681.

The function “ROM\_UARTCharsAvail(UART0\_BASE)” is true if there is any data in the receive FIFO. These values are put into the declared variable “value” in pong.c. If the charatcers match any of the characters (‘a’,’A’,’z’,’Z’,’;’,’) in the switch statement the “g\_pX\_move\_dir”, is assigned a value where either a 1 or 2 to determine paddles x-axis displacement, a space or ‘r’ character received over the serial connection the ball is released and the pong game begins.

Enabling the clocks for UART0 using sysctl\_RCGCUART and the GPIO port clock using sysctl\_RCGCGPIO in this project has been in effect implemented through:

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // line 579 in pong.c

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // line 580 in pong.c

This implementation of using interrupts does not show the asm implementation, which usually first of all disables all system interrupts (CPSID I) in the main program before initialisng GPIO and UART peripherals using system control registers. Where register values are saved to the stack. Interrupt types edge or level trigger as wel as high/low for edge and rising/falling or both are set.

Interrupts in asm would then be enabled for Port A, by enabling the clock for GPIOA using the SYSCTL\_RCGCGPIO register, then enabling alternate function for pins 0 and 1 using the base address of GPIO A and GPIO\_AFSEL register offset and logical ORRing bits 0 and 1 for UART0 and then selecting the alternate function for, that is the UART using GPIO\_PCTL offsetand assigning the he value for pins 0 and 1.

In this project this has been implemented as:

GPIOPinConfigure(GPIO\_PA0\_U0RX); // line 585 in pong.c

GPIOPinConfigure(GPIO\_PA1\_U0TX); // line 586 in pong.c

ROM\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); // line 587 in pong.c

Finally enabling digital access for input and output pins using the digital enable register offset GPIO\_DEN.

In this project this has been implemented through:

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0) // Line 579 in pong.c

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA) // Line 580 in pong.c

In this project the enable all interrupts CPSIE is done before configuring Port A as UART and UART interrupts:

ROM\_IntMasterEnable(); // line 583 in pong.c

In the C implementation of interrupts using the functions:

ROM\_IntEnable(INT\_UART0);

ROM\_UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

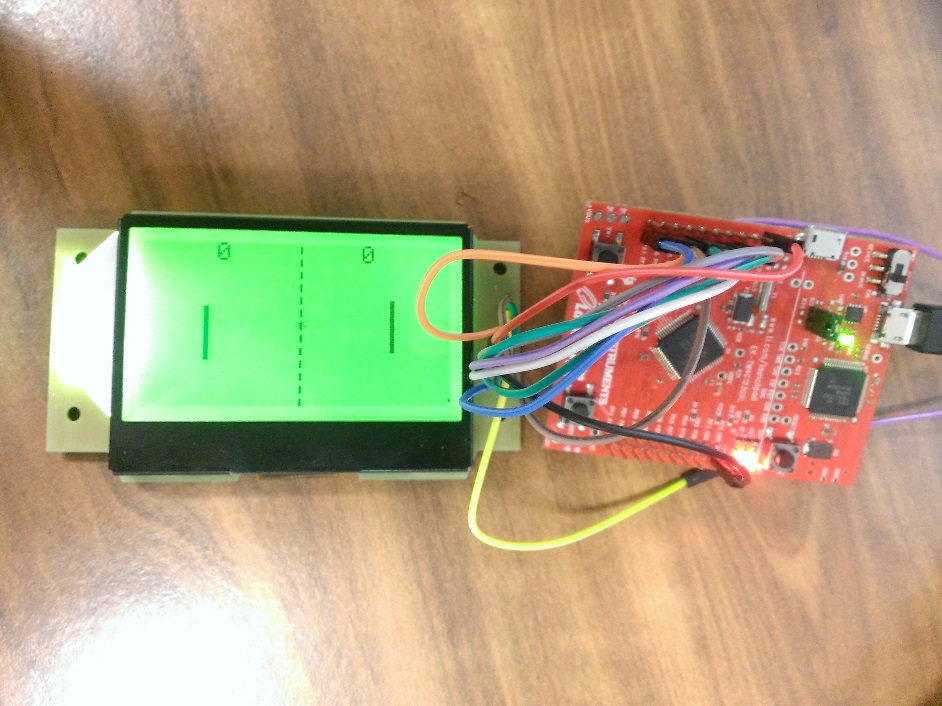
There is no explicit renaming of the associated interrupt vector table name at the associated position as done in ASM with vector tables. Using tiva-c files alowed setting the UART interrupts using the provided functions associates the UART0 interrupt to the associated interrupt number. The interrupt implementation in this project does not use an interrupt service routine, rather it uses a function called UARTHandler() and runs when a character is received.

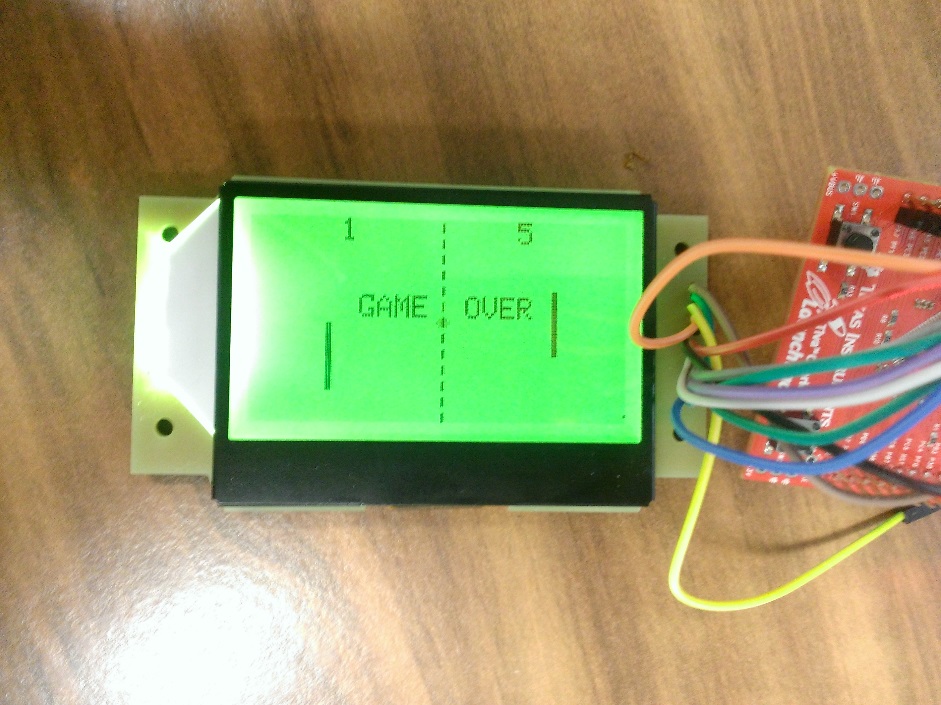
# Program Evaluation

The Project Program runs successfully fulfilling the outcomes in the System requirements document. However the objectives of this project could have been better satisfied by using the assembly language inline with the C code for an asm interrupt service routine and enable interrupts using interrupt registers as covered in the cause of MPT2303ENG.

The product of this project was focused more on the pong game dynamics to have it completely working in time rather, and less on interrupt enabling in using the register procedures as shown in the course. A more thorough understanding of interrupts could have been demonstrated by using interrupt registers directly using assembly or C rather than tiva-c files.

# Appendix A: Project Pictures





# Appendix B: Project Functions

**DrawCentreDottedLine()**

**void DrawCentreDottedLine()**

Input: Nil

Returns: Nil

Called by:

* PongInit()

Task

Draws a line vertically down the Screen to separate player 1 and player 2 screen.

**DrawBall**

**void DrawBall(uint8\_t x, uint8\_t y, bool clear)**

Input: x and y coordinates of ball position, type: signed char

Draw or clear parameter, type: boolean

Returns: Nil

Called by:

* PongInit()
* ResetBall()
* MoveBall(uint8\_t x\_direction, uint8\_t y\_direction)

Task

Determines the position of the ball to be drawn or cleared depending on the clear parameter.

True = clear ball.

False = draw ball.

Function uses ClearPixel(x, y) and DrawPixel(x, y) from lcd.h

**DrawPaddleP1(uint8\_t y)**

**void DrawPaddleP1(uint8\_t y)**

Input: y coordinates of paddle 1, type: signed char

Returns: Nil

Called by:

* TestDrawing()
* PongInit()
* MovePaddleP1(uint8\_t direction)

Task

Takes the y-coordinate from the called function. Draws paddle at provided y-axis.

Function uses DrawPixel(x, y) from lcd.h

**DrawPaddleP2(uint8\_t y)**

**void DrawPaddleP2(uint8\_t y)**

Input: y coordinates of paddle 2, type: signed char

Returns: Nil

Called by:

* TestDrawing()
* PongInit()
* MovePaddleP2(uint8\_t direction)

Task

Takes the y-coordinate from the called function. Draws paddle at provided y-axis.

Function uses DrawPixel(x, y) from lcd.h

**DrawScores(uint8\_t score\_p1, uint8\_t score\_p2)**

**void DrawScores(uint8\_t score\_p1, uint8\_t score\_p2)**

Input: Scores of player 1 and 2, type: signed char

Returns: Nil

Called by:

* PongInit()

Task

Draws the scores player 1 and player 2.

Function calls CursorPos(unsigned char x, unsigned char y), PutcharLCD(char c),from lcd.h

**DrawGameOver(uint8\_t is\_p1\_winner)**

**void DrawGameOver(uint8\_t is\_p1\_winner)**

Input: Player status (P1 or P2), type: boolean

Returns: Nil

Called by:

* TestDrawing()
* HandleScores()

Task

Draws “GAME OVER” when called with a logical true argument.

Function calls CursorPos(unsigned char x, unsigned char y) and PutcharLCD(char c) from lcd.h

**PongInit()**

**void PongInit()**

Input: Nil

Returns: Nil

Called by:

* main()
* MoveBall(uint8\_t x\_direction, uint8\_t y\_direction)

Task

Initialises the pong graphics to the screen, used at the beginnig after the all initialisations.

Function calls DrawScores(g\_score\_p1, g\_score\_p2), DrawPaddleP1(g\_p1\_position), DrawPaddleP2(g\_p2\_position), DrawBall(g\_ball\_position\_x, g\_ball\_position\_y, false) in pong.c

**ResetBall()**

**void ResetBall()**

Input: Nil

Returns: Nil

Called by:

* HandleScores()

Task

Resets the ball to the centre of the screen after the ball is missed by any of the players and reaches the end of the screen (i.e: x == 0 or x == LCD\_WIDTH)

Function calls DrawBall(g\_ball\_position\_x, g\_ball\_position\_y, true) in pong.c

**HandleScores()**

**void HandleScores()**

Input: Nil

Returns: Nil

Called by:

* DynamicsTick ()

Task

Keep track of scores, if the ball reaches the left boundary (i.e. x == LCD\_WIDTH) player1 scores a point.If the ball reaches the right boundary (i.e. x==0) player 2 scores a point. If any of the players score 5 (i.e. NUM\_ROUNDS) the game ends.

Function calls DrawGameOver(uint8\_t is\_p1\_winner) and ResetBall().

**HandleScores()**

**void HandleScores()**

Input: Nil

Returns: Nil

Called by:

* DynamicsTick ()

Task

Keep track of scores, if the ball reaches the left boundary (i.e. x == LCD\_WIDTH) player1 scores a point.If the ball reaches the right boundary (i.e. x==0) player 2 scores a point. If any of the players score 5 (i.e. NUM\_ROUNDS) the game ends.

Function calls DrawGameOver(uint8\_t is\_p1\_winner) and ResetBall().

**MovePaddleP1(uint8\_t direction)**

**void MovePaddleP1(uint8\_t direction)**

Input: Direction based on user input

Returns: Nil

Called by:

* main()

Task

Moves paddle up or down based on input argument.

**MovePaddleP2(uint8\_t direction)**

Same as *MovePaddleP2(uint8\_t direction)* for paddle 2.

**MoveBall(uint8\_t x\_direction, uint8\_t y\_direction)**

**void MoveBall(uint8\_t x\_direction, uint8\_t y\_direction)**

Input: Direction of ball to be bounced towards, determined in DynamicsTick()

Returns: Nil

Called by:

* DynamicsTick()
* TestDynamics()

Task

Moves the direction of the ball based on the x and y values provided by DynamicsTick().

**DynamicsTick**

Input: Nil

Returns: Nil

Called by:

* DynamicsTick()
* TestDynamics()

Task

DynamicsTick is continuously called in the main loop to decide whether the ball will rebound vertically straight, diagonally up/down depending on which third of the panel has made contact and the top or bottompanel of the screen set the x (vertical) and y (horizontal) displacement direction, by setting the variables y\_direction and y\_direction.

**DynamicsTick**

Input: Nil

Returns: Nil

Called by:

* DynamicsTick()
* TestDynamics()

Task

DynamicsTick is continuously called in the main loop to decide whether the ball will rebound vertically straight, diagonally up/down depending on which third of the panel has made contact and the top or bottompanel of the screen set the x (vertical) and y (horizontal) displacement direction, by setting the variables y\_direction and y\_direction.