LAB # 6
CSE-379
Introduction To Microprocessors

Partner 1: Josh Jeong (djjeong)
Partner 2: Alex Wojewoda (alexwoje)
Lab Section: R2
4/23/2021

Table of Contents:

Section 1: Division of work

Partner 1: output_board, choose_board, ascii_to_int, int_to_ascii, illuminate_RGB_LED, colored_symbol, lab6, Switch_Handler, UART0_Handler, Timer_Handler

Partner 2: simple_read_character, board_color_copy, board_symbol_copy, remove_all_colored_symbols, color_Complete, lab6, Switch_Handler, UART0_Handler, Timer_Handler

Section 2: Program Overview

How to use the program

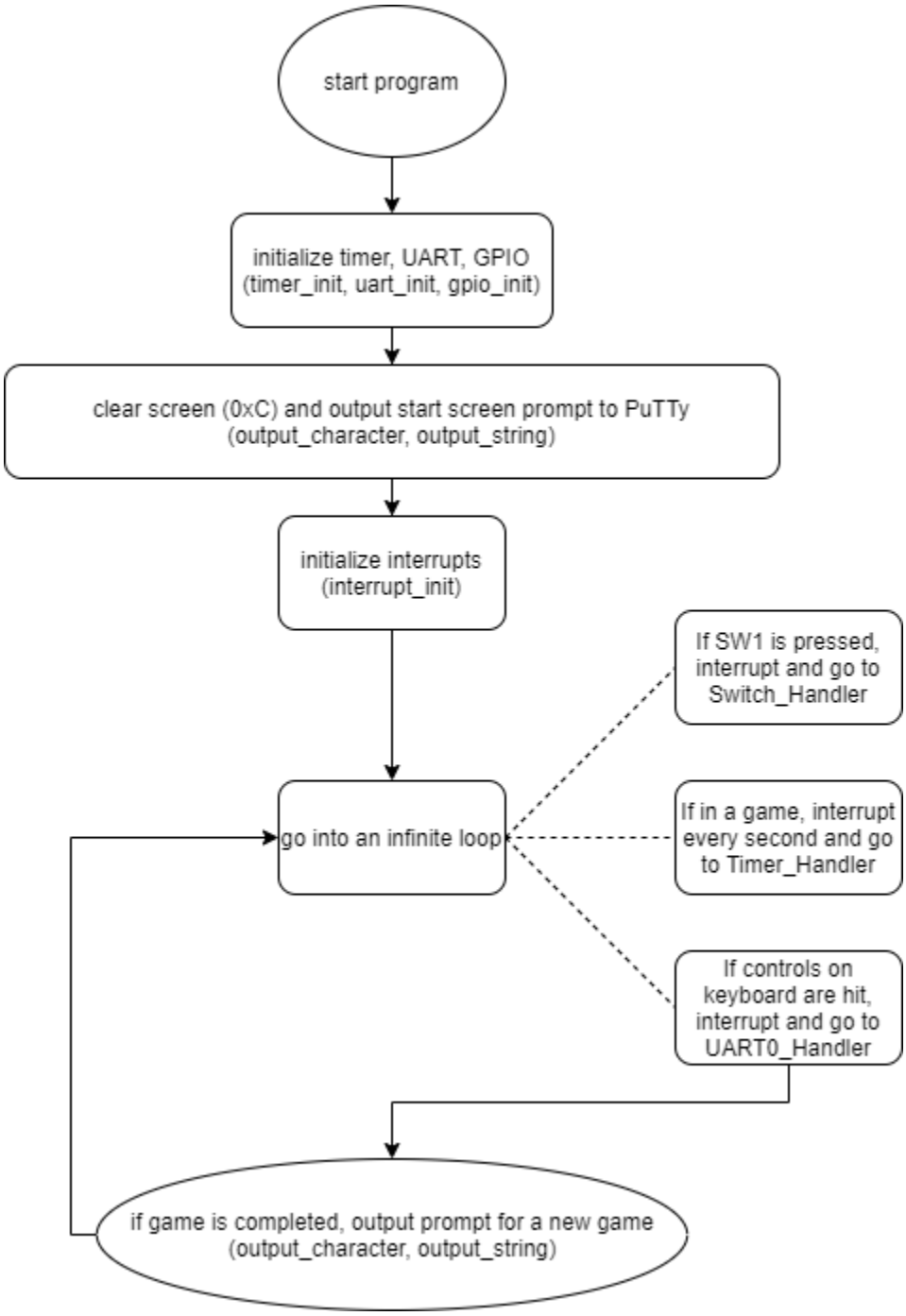
1. Change default background color of PuTTy to grey (red 128, green 128, blue 128) (to make the prompt texts, colors, and board visible)
2. Start program
3. Follow screen prompt on PuTTy
4. Controls
    a. Use the w, a, s, d keys to move up, left, down, right respectively
    b. Hit spacebar when over a ‘O’ to begin drawing
    c. Hit spacebar again to stop drawing
    d. When a line is completed, drawing is automatically stopped
    e. Press SW1 to pause game
        i. Press SW1 again to unpause
        ii. Press r to restart
        iii. Press n to start a new game
5. Connect every pair of colored ‘O’s with a line!
6. Try to fill all spaces and complete all lines on the board!

Program Summary

The program is a simple game consisting of a 7 x 7 board and ascii symbols printed out to PuTTy as a string using the cursor and ANSI escape sequences. The “movement” of the cursor is simulated via constant updates in the board string. The game starts with the cursor in the middle of the board. The user can control the cursor by hitting the w, a, s, d keys. Hitting the spacebar when over a colored ‘O’ will enable drawing of a line with the same color. Hitting the spacebar again disables drawing. The line is completed when it reaches the other same colored ‘O’ on the board. When a line is completed, drawing will stop automatically. The user may press the SW1 to pause and unpause a game. When paused, the user has the option to resume (press SW1), restart (press r),

or start a new game (press n). The game ends once every pair of colored ‘O’s have been connected with a line and the board is completely filled.

Flowsheet of entire program



Section 3: Subroutine Descriptions

UART0_Handler:
Clears the interrupt and modifies 2 board strings in memory (a board containing colors and a board containing symbols). The w, a, s, d keys are used to simulate movement on the board by updating the current and next position on the boards. Once a line is complete, the lines completed counter is incremented by 1. If a line is deleted, the lines completed counter is decremented by 1. The updates are then reflected in PuTTy by using ANSI escape sequences. The space bar enables and disables drawing. The r key is used to restart the game and the n key is used to start a new, random game.

Switch_Handler:
Clears the interrupt and modifies strings in memory containing the paused/unpaused state of the program. The paused variable string is either a “1” (not paused) or “2” (paused). The Handler changes the current string to the other. (from “1” to “2” or “2” to “1”)

Timer_Handler:
Clears the interrupt and modifies a time variable string in memory every second based on the paused/unpaused variable string. If the game is unpaused, the time variable string is incremented by 1 every second.

output_board:
Takes in the color board string address in r0, symbol board string address in r1. Iterates over both boards. According to 2 chars, 1 from each board, it outputs a board to PuTTy by using ANSI escape sequences.

uart_init:
Initializes the user UART for use.

timer_init:
Initializes Timer A for use, configured for interrupts every second.

gpio_init:
Initializes the GPIO for use.

interrupt_init:
Initializes the code to generate an interrupt when a keystroke is entered by the user or when the SW1 on the tiva board is pressed.

simple_read_character:
Helper subroutine in the library file that loads a byte from ARM memory register into r0

output_string:
Transmits a NULL-terminated ASCII string for display in PuTTy. The base address of the string is passed into the routine in r0.

output_character:
Transmits a character from the UART to PuTTy. The character is passed into the routine in r0 and stores it into the UART Data Register (UARTDR).

choose_board:
Takes in an int in r0 (board # 1 - 16). Modifies the color board string and symbol board string to match the board version.

board_color_copy:
Helper function for choose_board. Takes the base addresses of 2 boards in r0 and r1. Copies each color character from the board in r1 to the board in r0.

board_symbol_copy:
Helper function for choose_board. Takes the base addresses of 2 boards in r0 and r1. Copies each symbol character from the board in r1 to the board in r0.

num_digits:
Determines the number of digits in an integer and returns that number in r0. The integer is passed into the routine in r0.

str2int:
Converts a string to an integer and returns the integer in r0. The pointer to the string is passed into the routine in r0.

int2str:
Converts an integer to a string. The integer is passed into the routine in r0. The pointer to the string is passed in r1. The number of digits is passed in r2.

ascii_to_int:
Converts an ascii integer character to the corresponding integer. The ascii integer character is passed into the routine in r0.

int_to_ascii:
Converts a single digit integer to the corresponding ascii integer character. The single digit integer is passed into the routine in r0.

remove_all_colored_symbols:
Clears the color board and symbol board in memory of all symbols of a specific color, excluding the ‘O’ symbol. The specific color is passed into the routine in r0 as an ascii character.

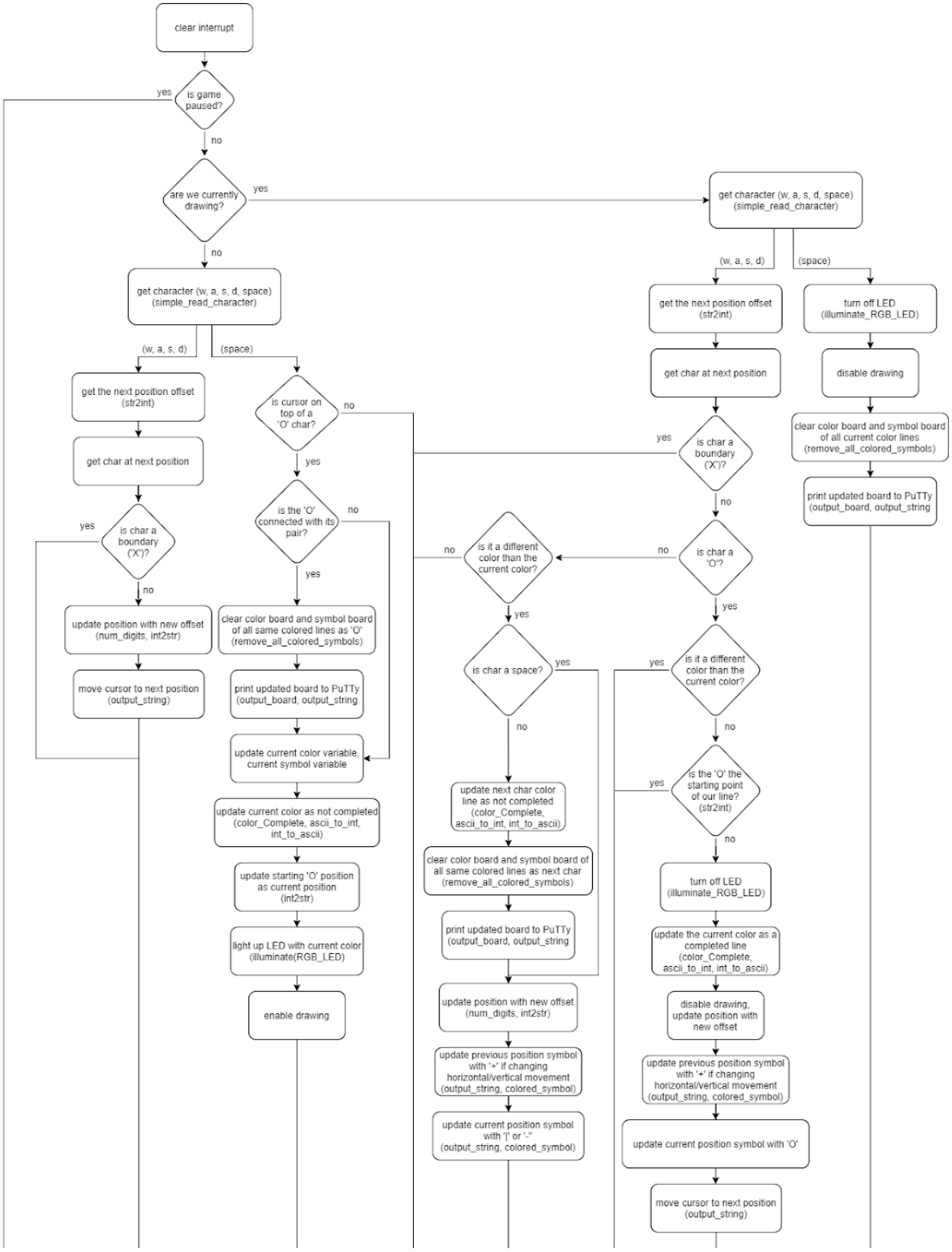
illuminate_RGB_LED:
Illuminates the color to be displayed. The color is passed into the routine as an int in r0 (blue=1, red=2, green=3, cyan=4, white=5, magenta=6, yellow=7). The GPIO data register’s pin 1 (2nd bit), pin 2 (3rd bit), and pin 3 (4th bit) are then set to the appropriate color.

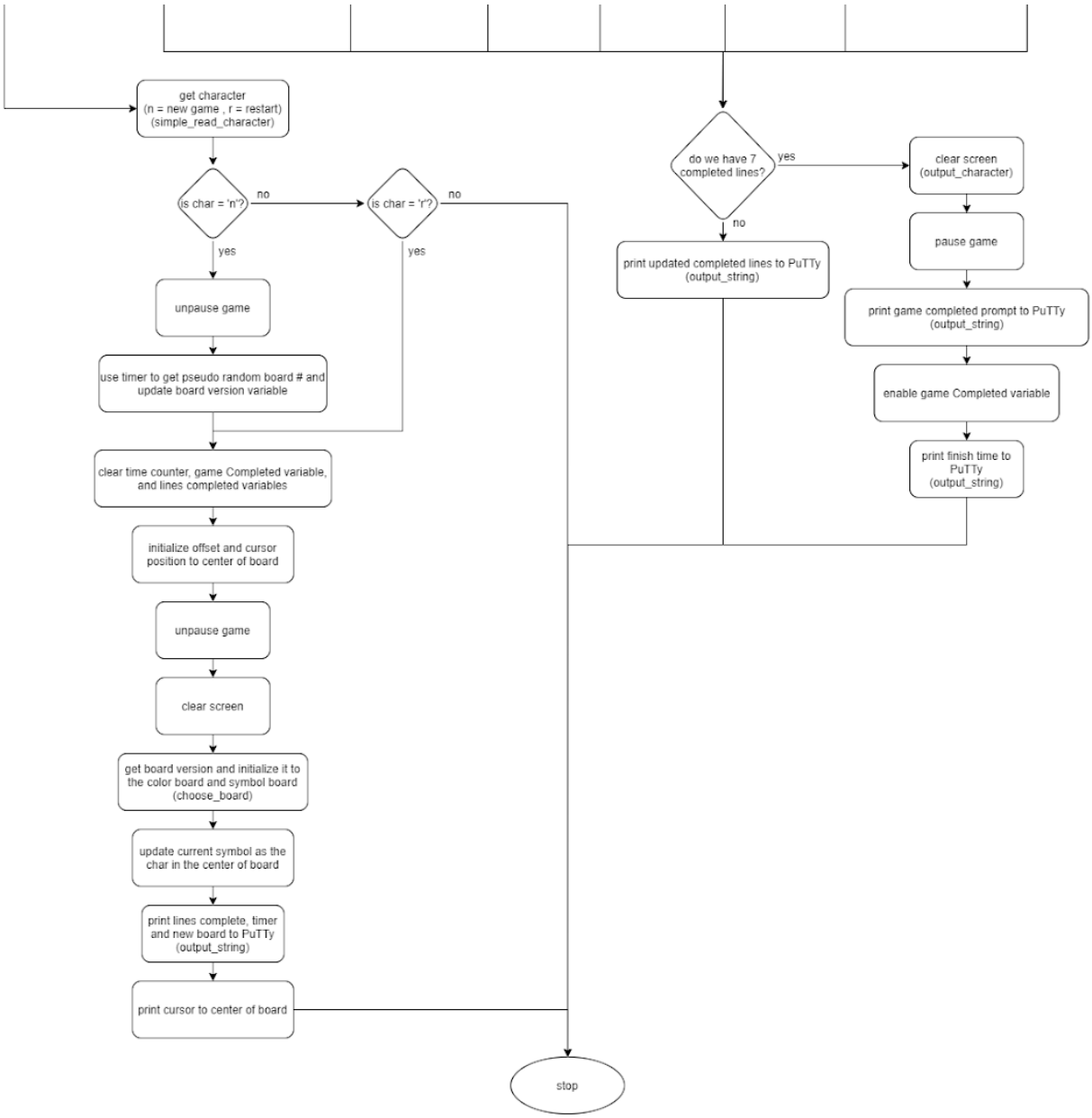
colored_symbol:
Takes in a color char int (blue=1, red=2, green=3, cyan=4, white=5, magenta=6, yellow=7) in r0 and a symbol char (‘X’, ‘O’, ‘|’, ‘-’, ‘+’) in r1. It then prints the correct colored symbol to PuTTy using ANSI escape sequences.

color_Complete:
Takes in a color char int (blue=1, red=2, green=3, cyan=4, white=5, magenta=6, yellow=7) in r0. Returns the ptr to the appropriate <color>Complete variable in r0. The <color>Complete variable indicates whether a line has been complete for that <color>.

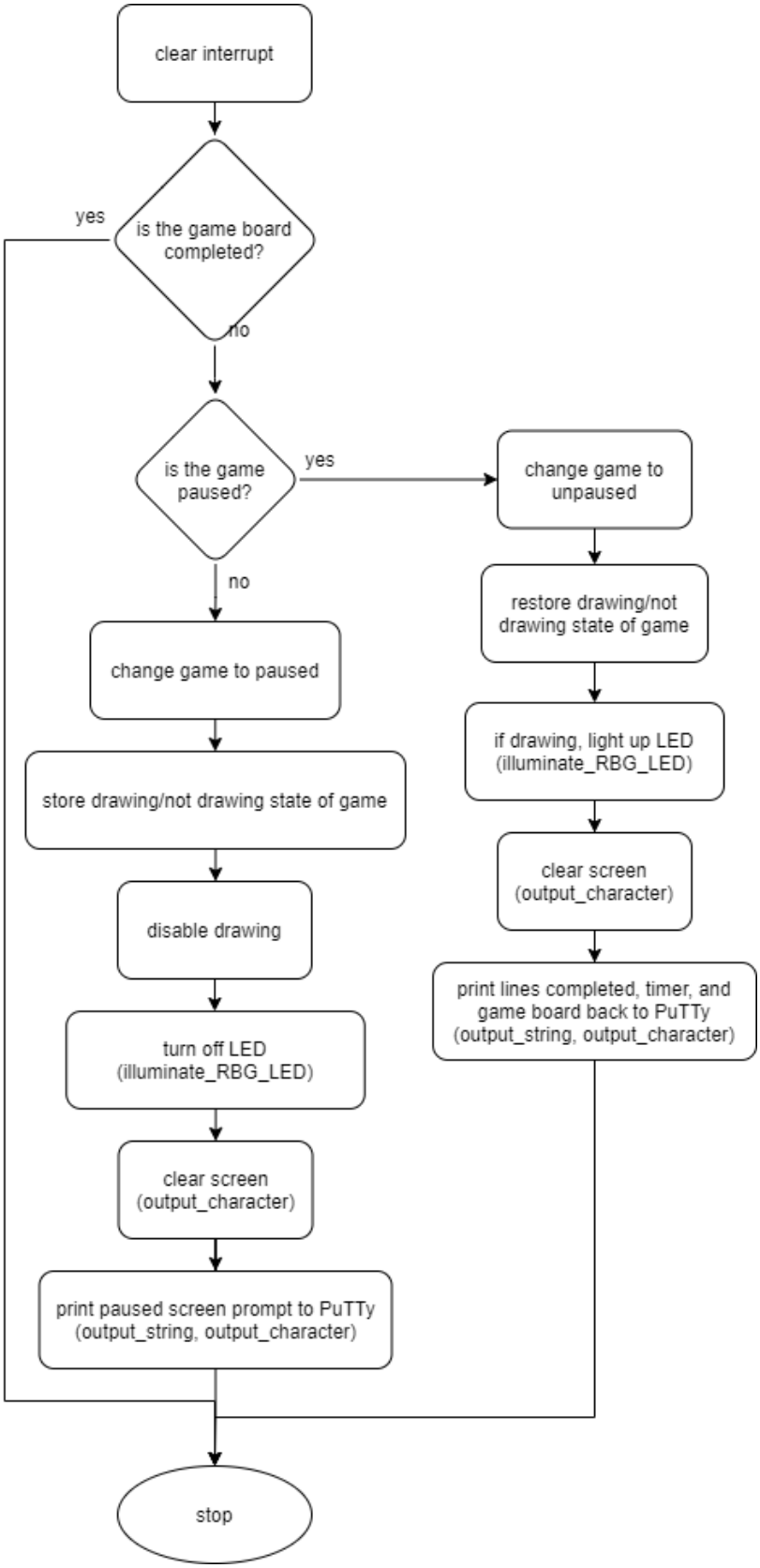
Section 4: Any additional Flowcharts

UART0_Handler:

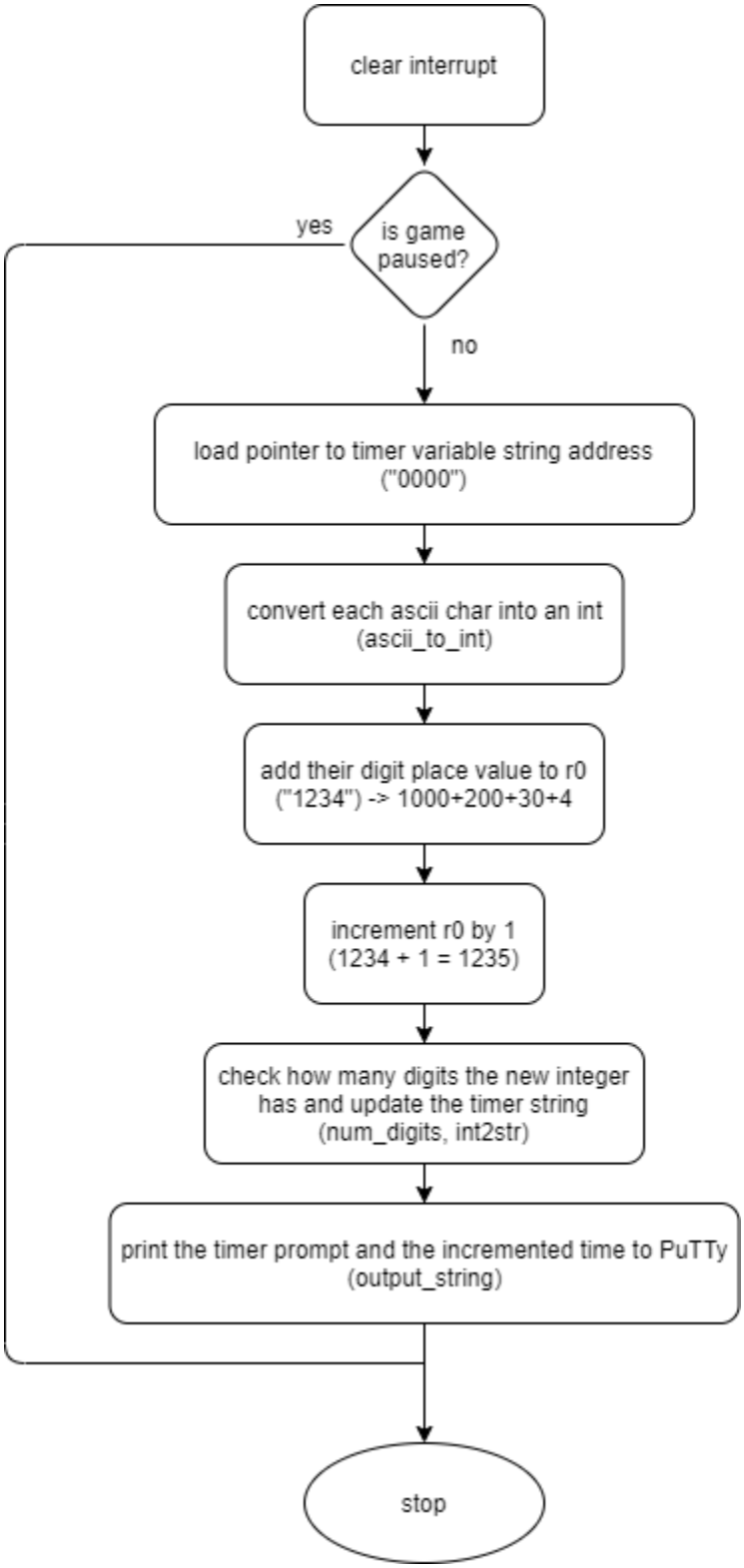



get character
(n = new game , r = restart)
(simple_read_character)
is char = 'n'?
no
is char = 'r'?
no
yes
yes
unpause game
use timer to get pseudo random board # and update board version variable
clear time counter, game Completed variable, and lines completed variables
initialize offset and cursor position to center of board
unpause game
clear screen
get board version and initialize it to the color board and symbol board (choose_board)
update current symbol as the char in the center of board
print lines complete, timer and new board to PuTTy (output_string)
print cursor to center of board
do we have 7 completed lines?
yes
no
clear screen (output_character)
pause game
print game completed prompt to PuTTy (output_string)
enable game Completed variable
print finish time to PuTTy (output_string)
print updated completed lines to PuTTy (output_string)
stop

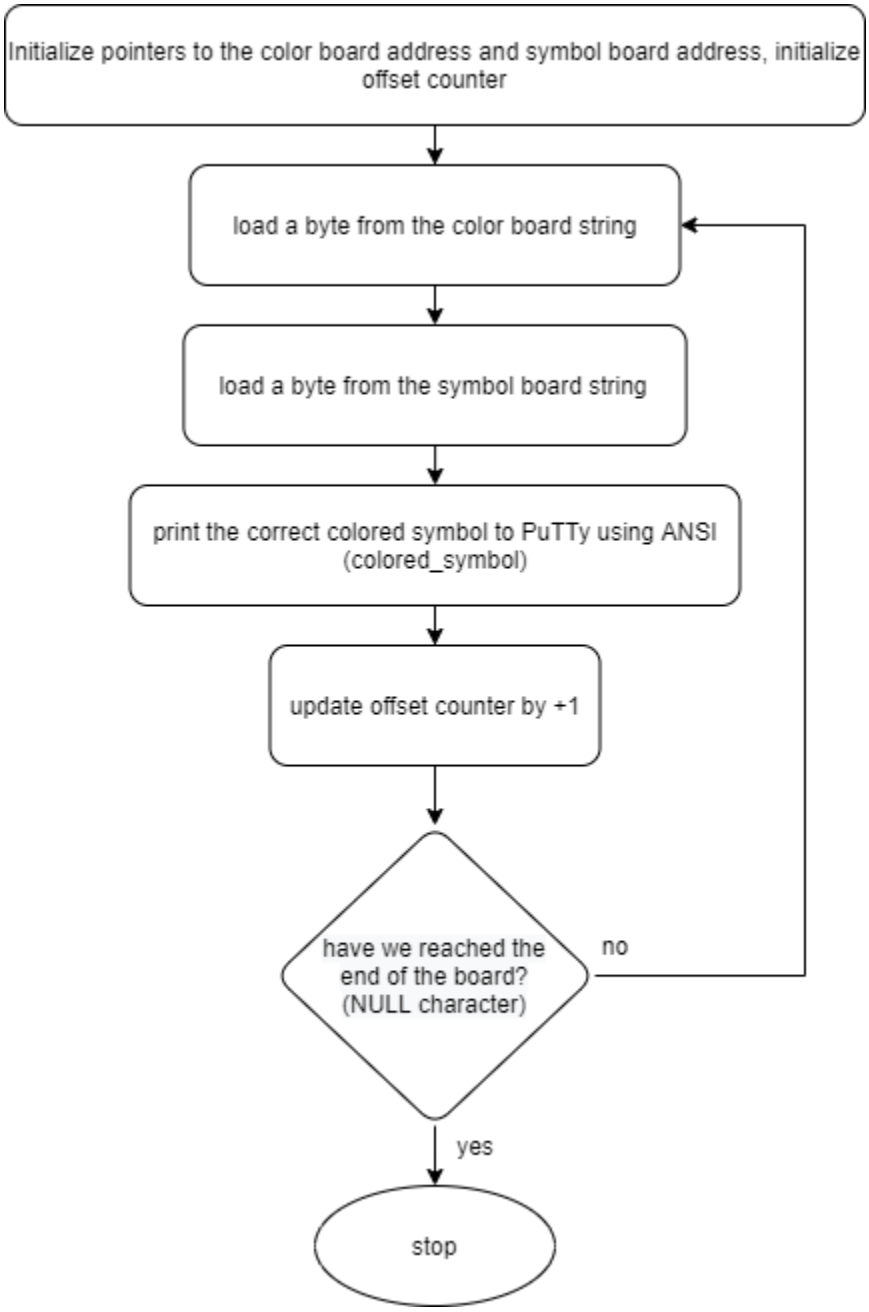
Switch_Handler:



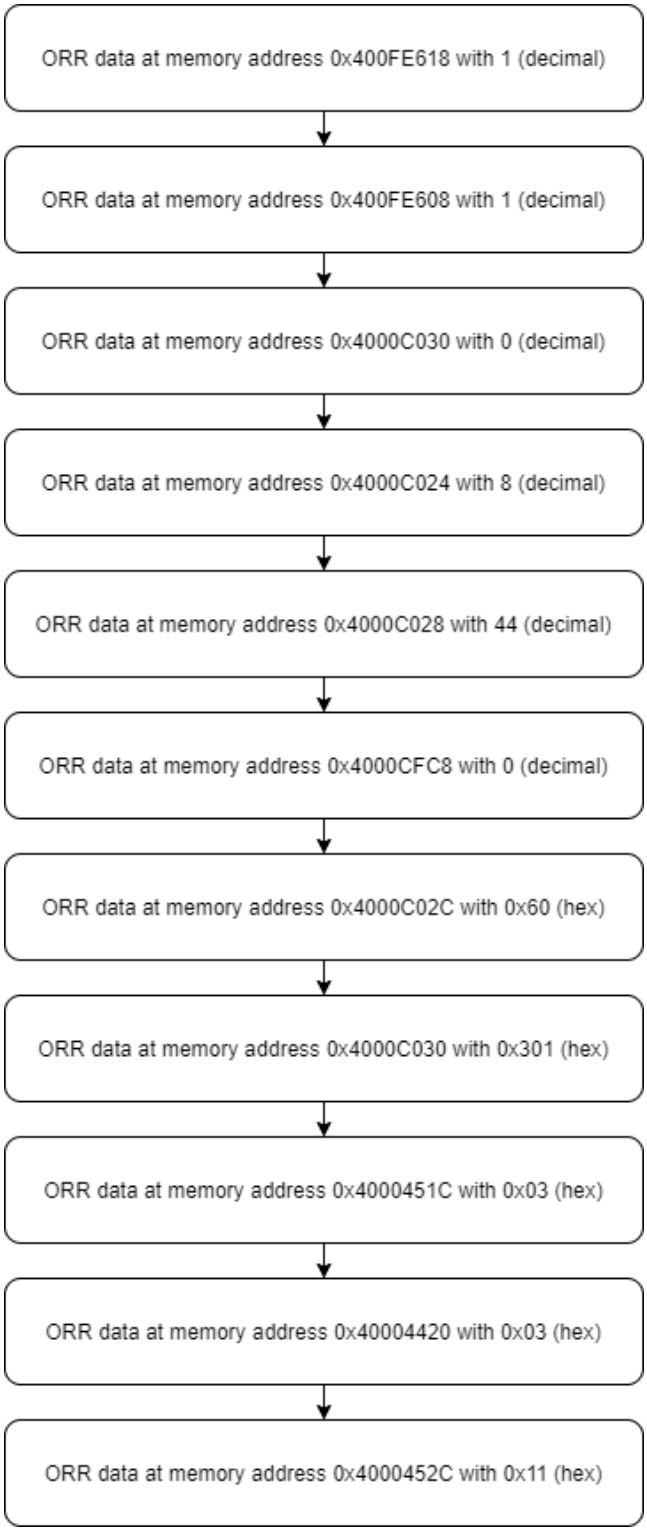
Timer_Handler:



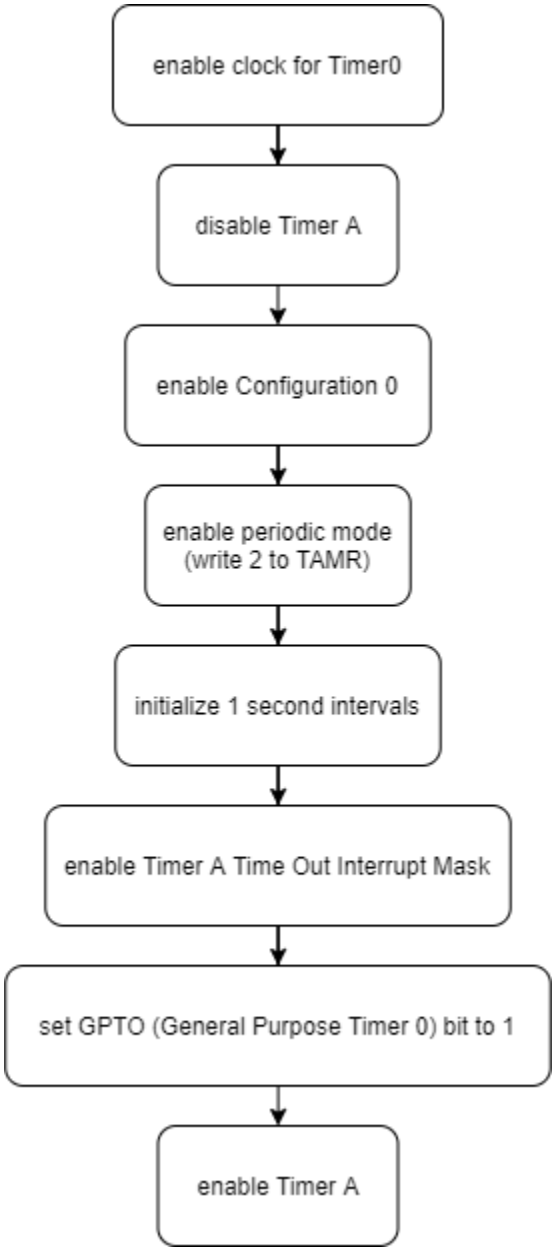
output_board:



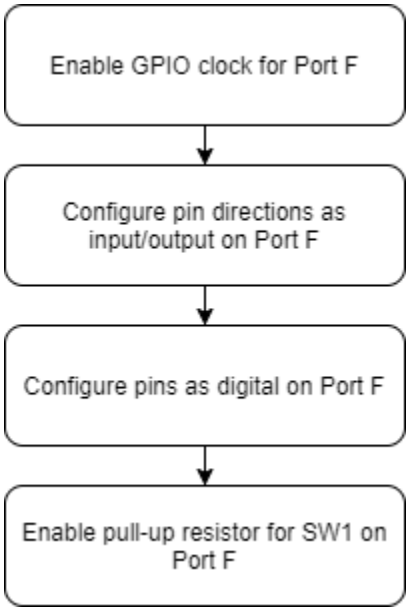
uart_init:


ORR data at memory address 0x400FE618 with 1 (decimal)
ORR data at memory address 0x400FE608 with 1 (decimal)
ORR data at memory address 0x4000C030 with 0 (decimal)
ORR data at memory address 0x4000C024 with 8 (decimal)
ORR data at memory address 0x4000C028 with 44 (decimal)
ORR data at memory address 0x4000CFC8 with 0 (decimal)
ORR data at memory address 0x4000C02C with 0x60 (hex)
ORR data at memory address 0x4000C030 with 0x301 (hex)
ORR data at memory address 0x4000451C with 0x03 (hex)
ORR data at memory address 0x40004420 with 0x03 (hex)
ORR data at memory address 0x4000452C with 0x11 (hex)

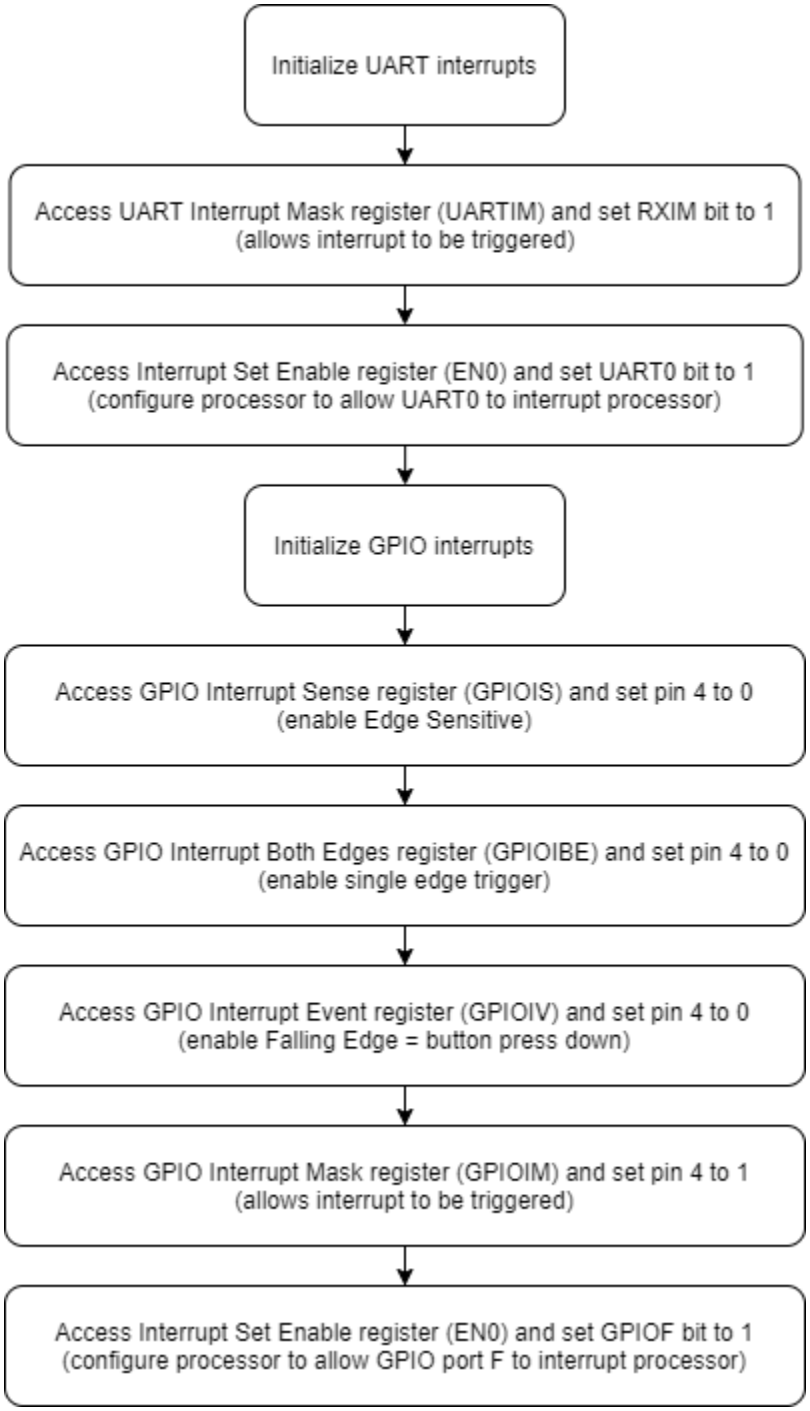
timer_init:


enable clock for Timer0
disable Timer A
enable Configuration 0
enable periodic mode (write 2 to TAMR)
initialize 1 second intervals
enable Timer A Time Out Interrupt Mask
set GPTO (General Purpose Timer 0) bit to 1
enable Timer A

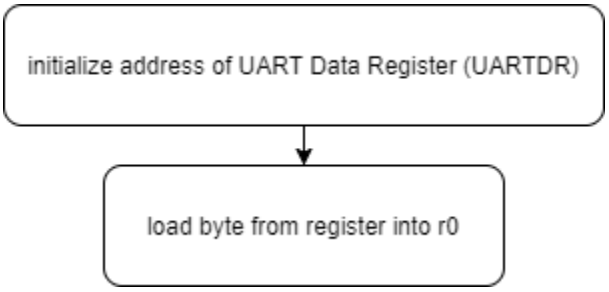
gpio_init:


Enable GPIO clock for Port F
Configure pin directions as input/output on Port F
Configure pins as digital on Port F
Enable pull-up resistor for SW1 on Port F

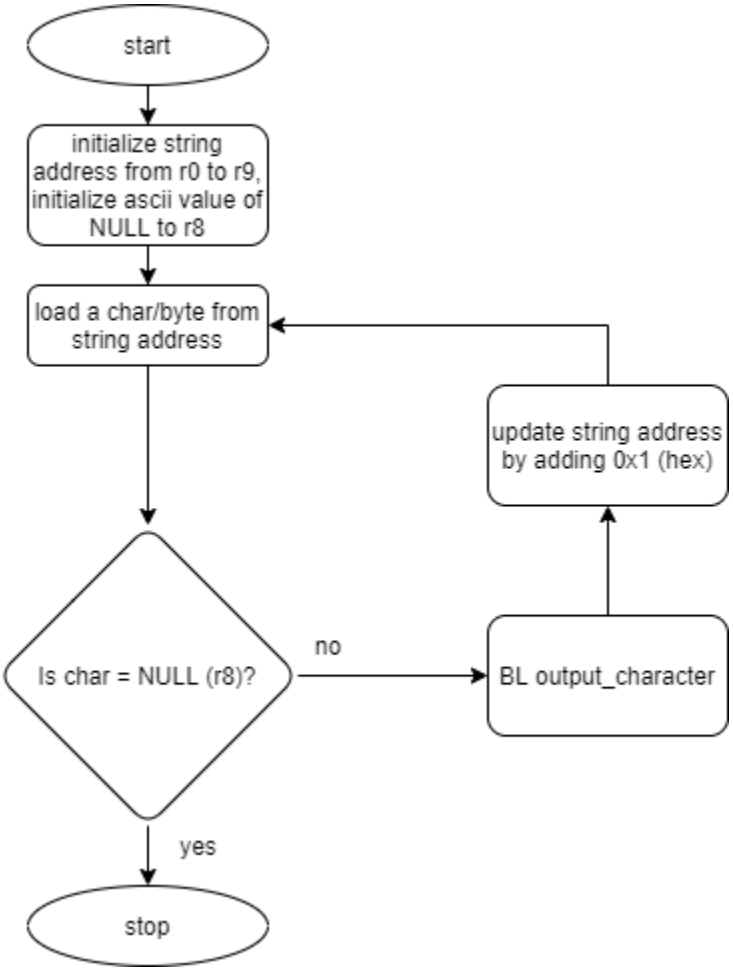
interrupt_init:



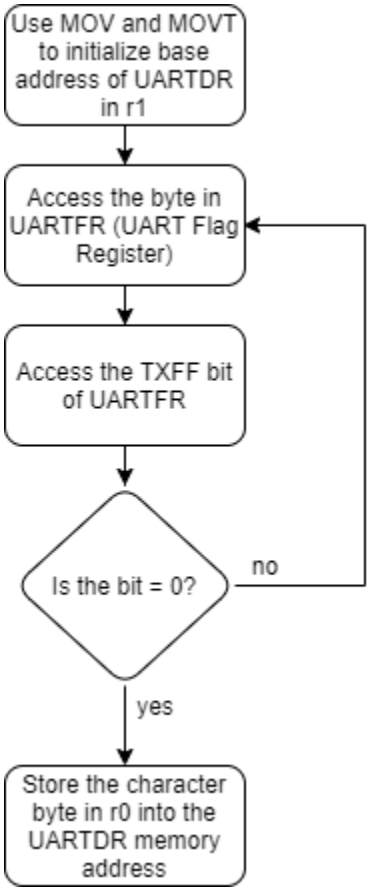
simple_read_character:



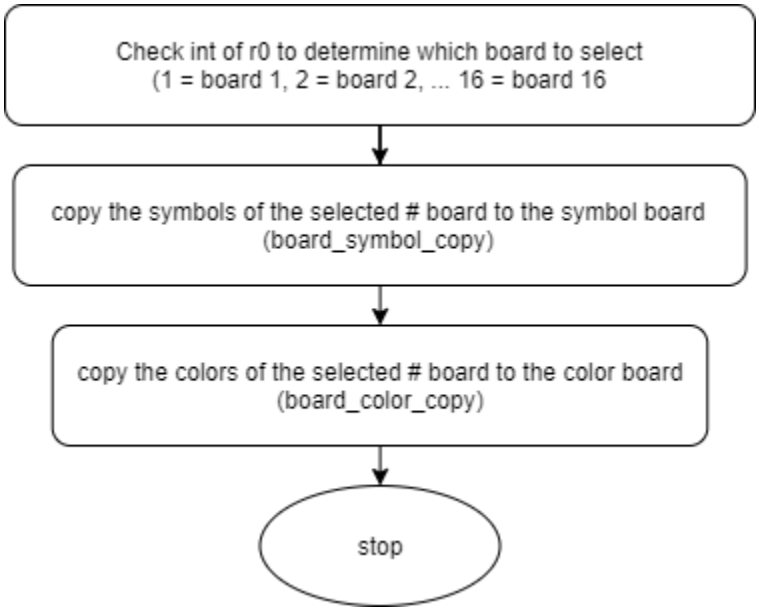
output_string:



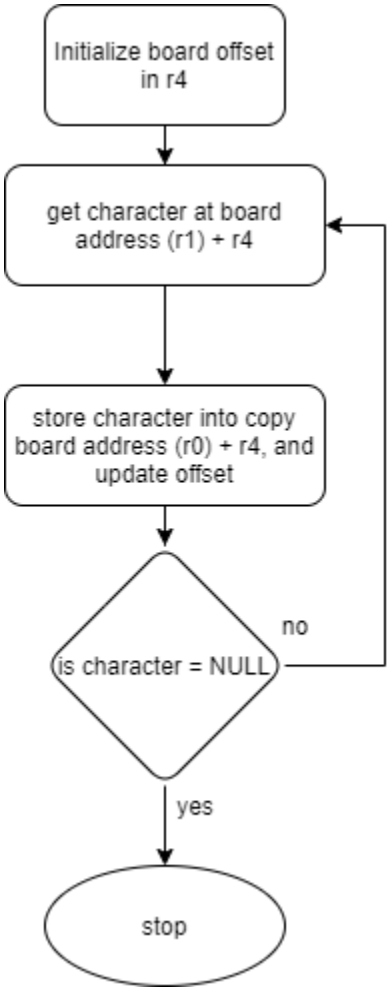
output_character:



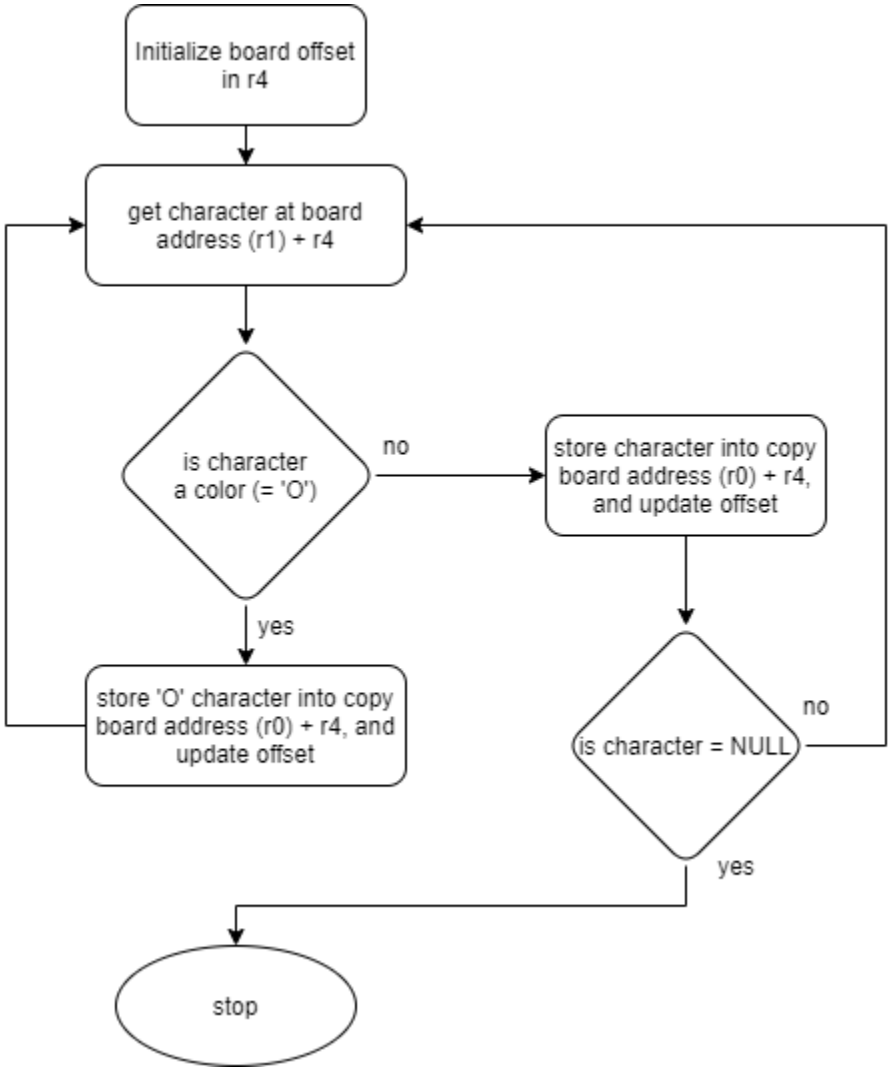
choose_board:



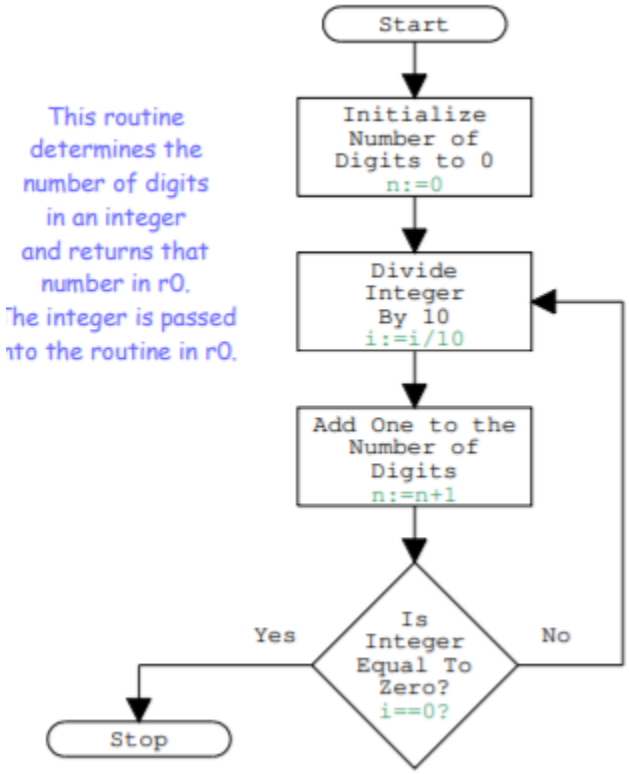
board_color_copy:



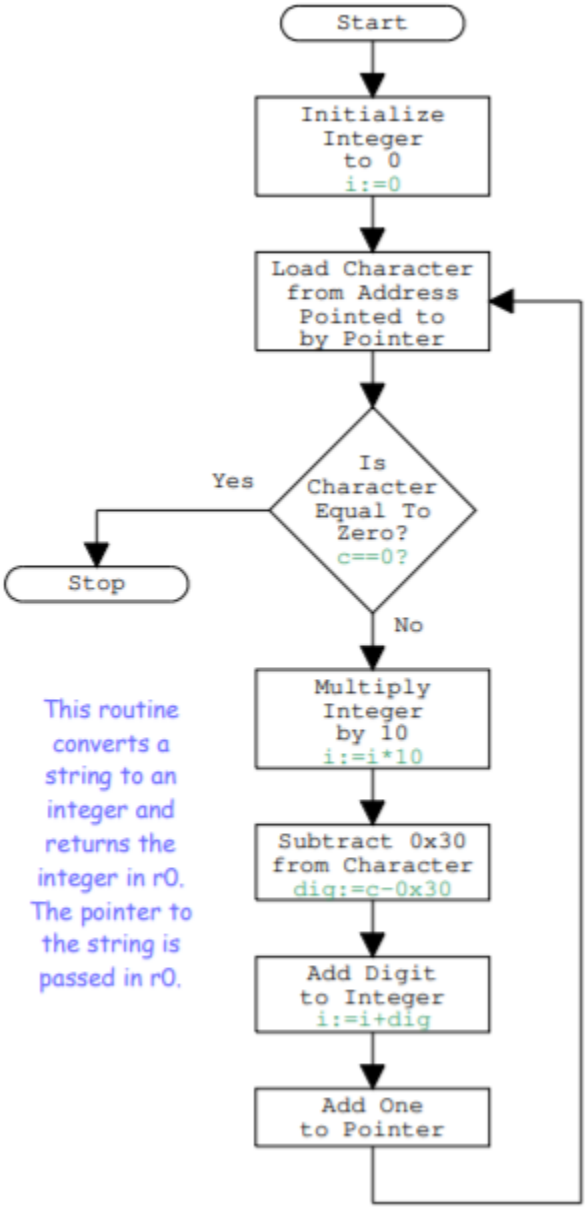
board_symbol_copy:



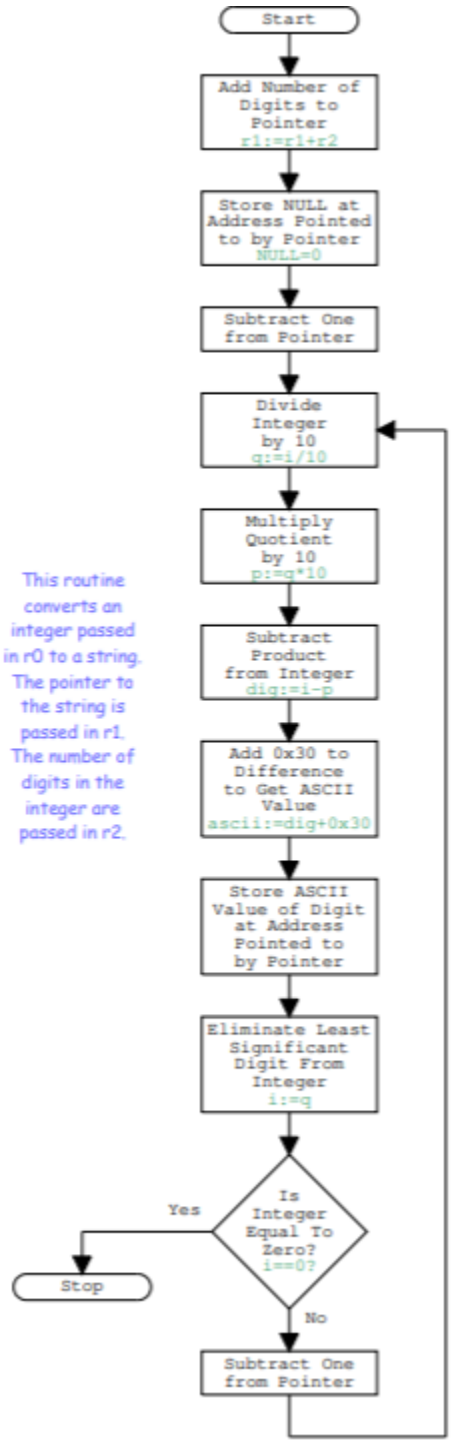
num_digits:


This routine determines the number of digits in an integer and returns that number in r0.
The integer is passed into the routine in r0.
Start
Initialize Number of Digits to 0
n:=0
Divide Integer By 10
i:=i/10
Add One to the Number of Digits
n:=n+1
Is Integer Equal To Zero?
i==0?
Yes
No
Stop

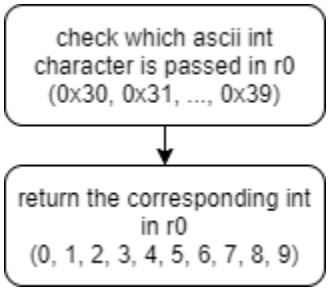
str2int:


Start
Initialize Integer to 0
i:=0
Load Character from Address Pointed to by Pointer
Is Character Equal To Zero?
c==0?
Yes
Stop
No
Multiply Integer by 10
i:=i*10
Subtract 0x30 from Character
dig:=c-0x30
Add Digit to Integer
i:=i+dig
Add One to Pointer
This routine converts a string to an integer and returns the integer in r0. The pointer to the string is passed in r0.

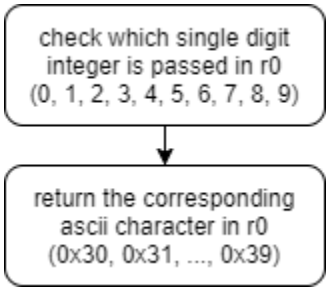
int2str:


Start
Add Number of Digits to Pointer
r1:=r1+r2
Store NULL at Address Pointed to by Pointer
NULL=0
Subtract One from Pointer
Divide Integer by 10
q:=i/10
Multiply Quotient by 10
p:=q*10
Subtract Product from Integer
dig:=i-p
Add 0x30 to Difference to Get ASCII Value
ascii:=dig+0x30
Store ASCII Value of Digit at Address Pointed to by Pointer
Eliminate Least Significant Digit From Integer
i:=q
Is Integer Equal To Zero?
i==0?
Yes
Stop
No
Subtract One from Pointer
This routine converts an integer passed in r0 to a string. The pointer to the string is passed in r1. The number of digits in the integer are passed in r2.

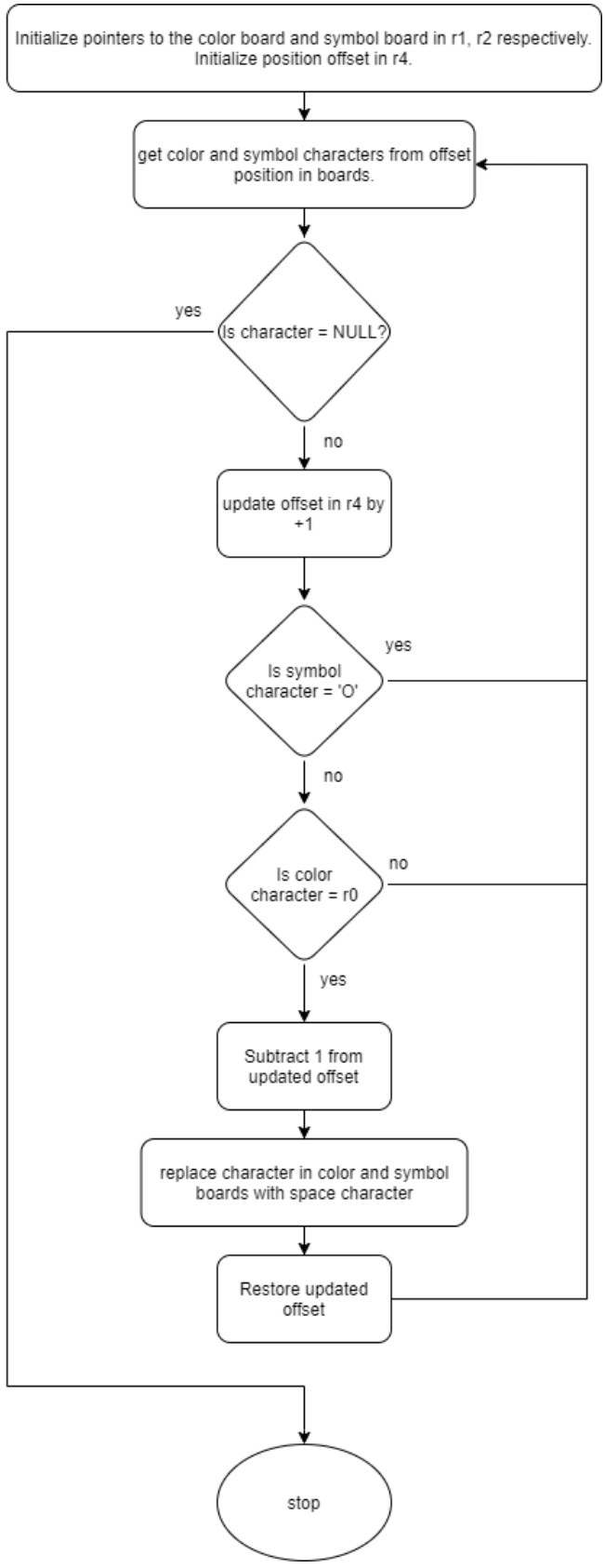
ascii_to_int:


check which ascii int character is passed in r0 (0x30, 0x31, ..., 0x39)
return the corresponding int in r0 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

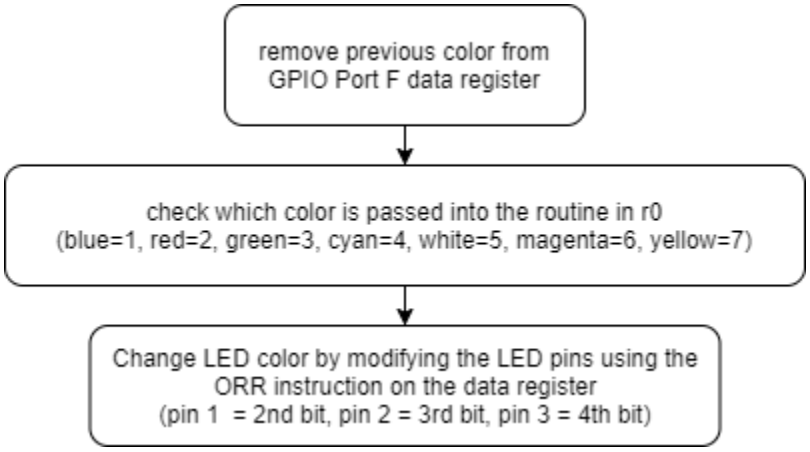
int_to_ascii:


check which single digit integer is passed in r0 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
return the corresponding ascii character in r0 (0x30, 0x31, ..., 0x39)

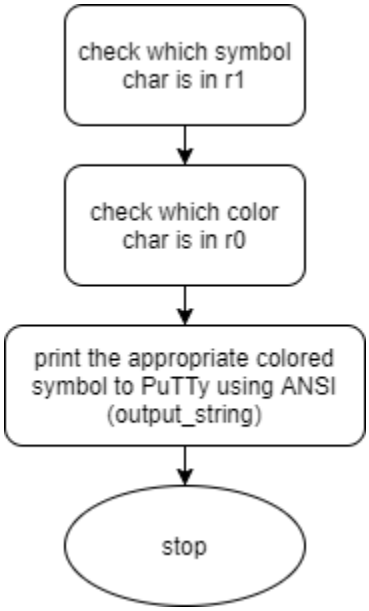
remove_all_colored_symbols:



illuminate_RGB_LED:



colored_symbol:



color_Complete: