CSE-379 Lab 6 Documentation

Patrick Wilkinson & Andy Dong
prwilkin
adong4

Section 1: Division of Work

Patrick Wilkinson:
Did the subroutines decode, encode, checker, mover, printer, printer_assist, end_game, game. Modified the code in UART0_handler.
Andy Dong:
Did the subroutine uart_init, uart_interrupt_init, gpio_interrupt_init, switch_Handler, Timer_interrupt_init, and Timer_Handler of the lab.

Section 2: Program Overview

The routine lab6 is the higher level function to enable the program. uart_init, uart_init_interupt, gpio_init_interupt, timer_init_interupt, and clear page are called in that order. r5 is initialized with 1 to set the counter for switch presses. Then an infinite loop begins. Each time the Timer_Handler happens, the grid refreshes and the star moves once in the grid.

Section 3: Subroutine Descriptions

uart_init(from lab5):

It initializes the user UART for use in arm assembly language. The argument for this subroutine is that r0 is set as the address needed to enable elements of the user UART and r1 is used to store the integer that is needed for that address. The subroutine provide clock to UART0, enable clock to PortA, disable UART0 Control, set UART0_IBRD_R for 115,200 bud, set UART0_FBRD_R for 115,200 bud, use system clock, use 8-bit word length, 1 stop bit, no parity, enable UART0 Control, make PA0 and PA1 as Digital Ports, change PA0,PA1 to use an Alternate Function, and configure PA0 and PA1 for UART.

uart_interrupt_init(from lab5):

It initializes the user interrupt UART for use in arm assembly language. The argument for this subroutine is that r0 is set as the address needed to enable elements of the user interrupt UART and r1 is used to store the integer that is needed for that address. The subroutine provides setting the receive interrupt mask bit in the UART Interrupt Mask Register and configuring the processor to allow the UART to Interrupt Processor.

gpio_interrupt_init(from lab5):

It initializes the momentary push button on the Tiva Board (SW1) and it also initializes the interrupt for the processor to let Port F to use the interrupt and allowing the interrupt handler to work for Port F. The argument for this subroutine is that r0 will be holding the base address of Port F that will also be added with offset to set clock, pin direction, enabling pin, edge sensitive, falling edge as the interrupt, enable the interrupt, and allowing Port F to interrupt in the processor to be used R1 will be used to load the data from r0 with offset and change the bit based on what needs to be 1 or 0. There is no return value, but it will initialize everything and will allow us to use the button properly.

timer_interrupt_init:

It initializes the Timer and the interrupt for the processor to let Timer0 to use interrupt and allows the interrupt handler to work for Timer0. The argument for this subroutine is that r0 is set as the base address of Timer0 to enable elements of the Timer0 and r1 is used to store the bits that need to be set or clear to allow the elements that are needed for the Timer to work properly. The elements that needed to be set up are disabling the timer, setting the Timer for 32-Bit Mode, putting Timer in Periodic Mode, setting up the Interval Period, setting Timer to interrupt Processor, configure processor to allow timer to interrupt Processor, and enabling the Timer. There is no return value, but it will initialize everything and will allow us to use the Timer properly.

UART_Handler:

The uart_handler calls simple_read_charcater after clearing the interrupt. r1 then loads the pointer to direction data. 4 comparisons occur to determine the key pressed. These comparisons consist of comparing W, A, S, & D against their ascii value which is in r0 from simple_read_character. If the character doesn't match it branch to the next character and if it does match r0 then holds 0 for w, 1 for d, 2 for s, and 3 for a. After that r0 is stored to memory at the location stored in r1.

Switch_Handler:

It shortens the time interval of the Timer to cause the Timer_Handler to be interrupted faster and clear the interrupt of the switch handler so it can be interrupted again. The argument for this subroutine is that r5 will be used as a counter that will be used to shorten the time interval by dividing 16Mhz with the value in r5. r0 will be holding the base address of Port F to clear the interrupt of the switch_handler and base address of timer to disable the timer, shorten the time interval, and enable the timer again. r1 will be used to load the data from r0 with offset and change the bit based on what needs to be 1 or 0.

Timer_Handler:

Everytime the Timer_Handler is interrupted, it first clears the interrupt for the Timer_Handler and branches to another subroutine to move the star inside the 20x20 box and check the coordinate of the star to know where the star is at. The argument for this subroutine is that r0 will be holding the base address of Timer0 and r1 will be used to load the data from r0 with offset and change the bit to clear the Timer0 interrupt. Game is called to perform its functions.

simple_read_character(from lab5):

It reads a character provided by the UART from Putty and returns the character to r0. r2 loads the memory address of UART0. Then store the memory address of r2 to r0.

output_character(from lab5):

It transmits a character from the UART to Putty. The argument for this subroutine is that r2 is holding the memory address of UART0 and loads r2 with the offset of UART Flag Register to r1. Then we use bit masking to isolate the TxFF bit and compare if r1 is equal to 0. If it is not, then it loops back and repeats the code again until r1 does not equal to 1. Then store the memory address of r2 to r0.

output_string(from lab5):

Takes r0 as a base address to the location of a string stored in memory location. The character is loaded from the memory location and passed to output_character in r0 if the character is not a NULL. If character is NULL, output_character terminates. When returned from output_character, the address is incremented by 1 in a byte size.

Int2string (from lab3):

It takes the int stored of r0 and converts it into string, then storing the string into r1. The argument for this subroutine is that r4, r5, r9, and r10 are used to be used as immediate numbers, so we are allowed to use the multiply instruction in the subroutine. The subroutine gets the int from r0 and checks if the int is null. Then if it is, then divide 10 on r0 and store it in r1. If not, then convert the ascii digit to int and then multiply by 10 so the next digit can be placed into the int if the digit is greater than 10.

decode:

An address to a half word size section of memory is passed in r2. The half word is loaded into r0 from location in r2. r0 is anded with #0xFF and stored in r1, and then r0 is shifted right by 8 bits. This is done to decode the coordinates and then store the 1 byte x coordinate in r0 and the 1 byte y coordinate.

encode:

An address to a half word size section of memory is passed in r2. r0 is shifted left by 8 bits , and then r0 is or'ed with r1 storing in r0. The half word is stored from r0 at location in r2. This is done to encode the coordinates and then store the 1 byte x coordinate in r0 and the 1 byte y coordinate in r1 into memory.

checker:

A pointer for current coordinates is loaded into r2 and then decode is called. After receiving a 1 byte x coordinate in r0 and the 1 byte y coordinate in r1 from decode, r2 is then used to load the memory location of the direction data and then the direction data itself. After this, comparisons occur following this structure, check the coordinates to see if you're along a wall, if no branch to the next coordinate wall check. If yes then check is direction has you going into that wall, if no then branch the next coordinate wall check, and if yes branch to end_game. The order is, left wall (x coordinate 0x00) and direction left (3), then right wall (x coordinate 0x13) and direction right (1), then top wal (y coordinate 0x00) and direction up (0), and finally bottom wall (y coordinate 0x13) and direction (2). The subroutine then ends.

mover:

A pointer for current coordinates is loaded into r2 and then decode is called. After receiving a 1 byte x coordinate in r0 and the 1 byte y coordinate in r1 from decode, r2 is then used to load the memory location of the direction data and then the direction data itself. After this, comparisons occur following this structure, check the direction, if not that direction then advance to the next direction. If it is direction then adjust the appropriate coordinate according to direction. Then branch to the end and load a pointer for next coordinates into r2 and call encode to store the new coordinates. The order is, direction up (0) and y coordinate is subtracted by 1, direction right (1) and x coordinate is added by 1, direction down (2) and y coordinate is added by 1, then direction left (3) and x coordinate subtracted by 1.

printer:

clear_page is called and then the top wall pointer is stored in r0 and then output_string is called to display the top wall and then newline is called. Next each row's pointer is loaded into r1 and then passed to printer assist. This occurs 20 times, once for each row. After all 20 have been printed the bottom wall pointer is stored in r0 and then output_string is called to display the bottom wall.

printer_assist:

Takes r1 as a pointer to the row coordinates. It first prints the right wall then sets r4 as a counter equal to 0 and finally loads the next coordinates into the r2. This is the beginning of a loop that loops 20 times before exiting. A comparison is made to determine if r4 is equal to 20 and then r4 is added by 1. If r4 was equal to 20 then it will exit the loop, otherwise the coordinates at address r1 will be stored in r3 and the address in r1 will be incremented by 1. r3 is compared to r2 to determine if the current position being printed matches the next coordinates. If they match an asterisk is printed, otherwise a space is. The loop then repeats, and upon exiting the loop the left wall is printed alongside a newline.

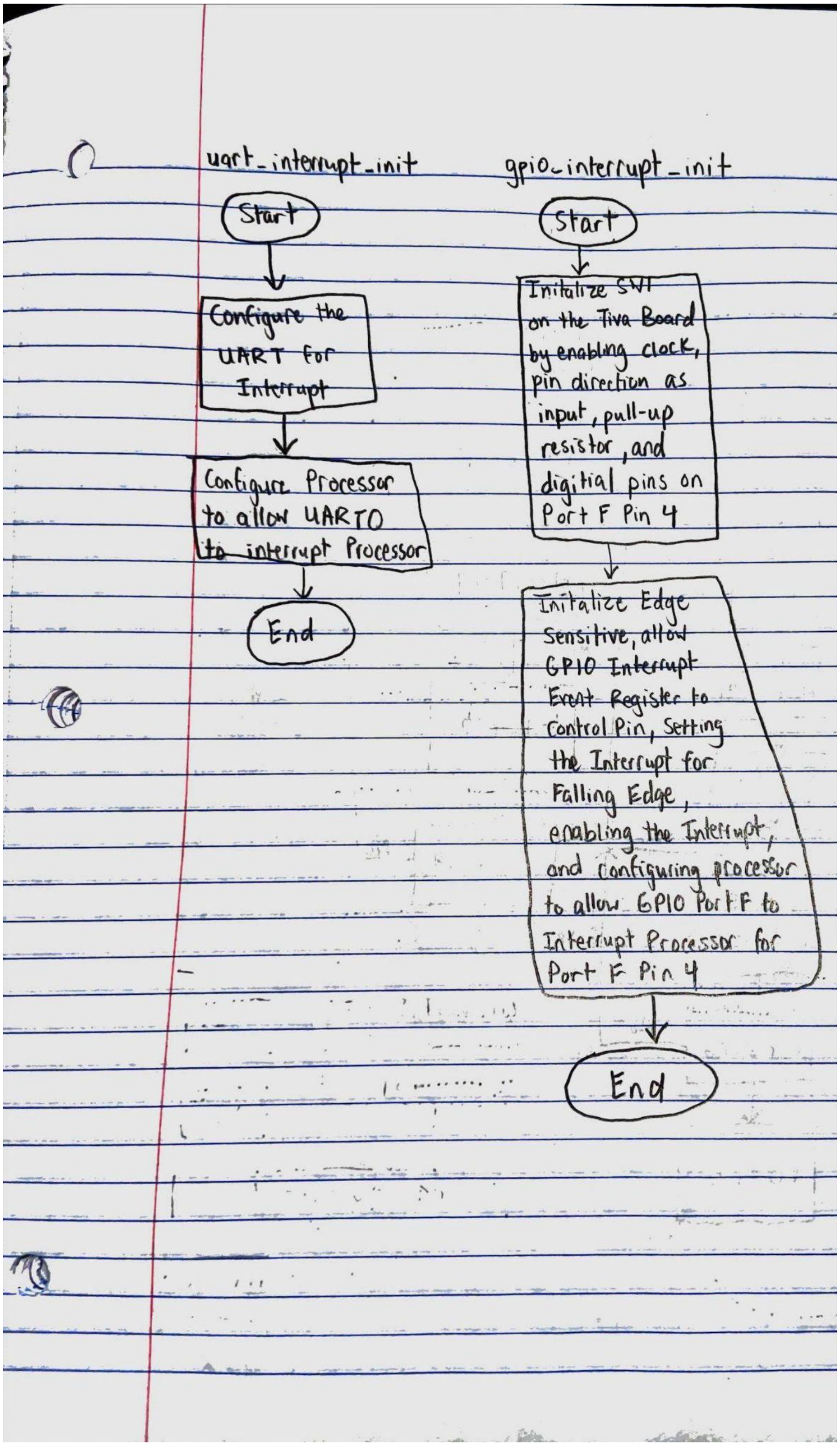
end_game:

A new line is printed under the game board and r0 loads a pointer to a message saying you hit a wall and is passed to output_string. A new line is printed and then another pointer to another message stating the total move is loaded in r0 and passed to output_string. r9 is moved to r0 and r1 loads a pointer to a small space in memory to pass to int2String so the total number of moves can be converted to a string. r0 then loads the same memory pointer to pass to the output string and allows the number to be printed then the entire program ends.

game:

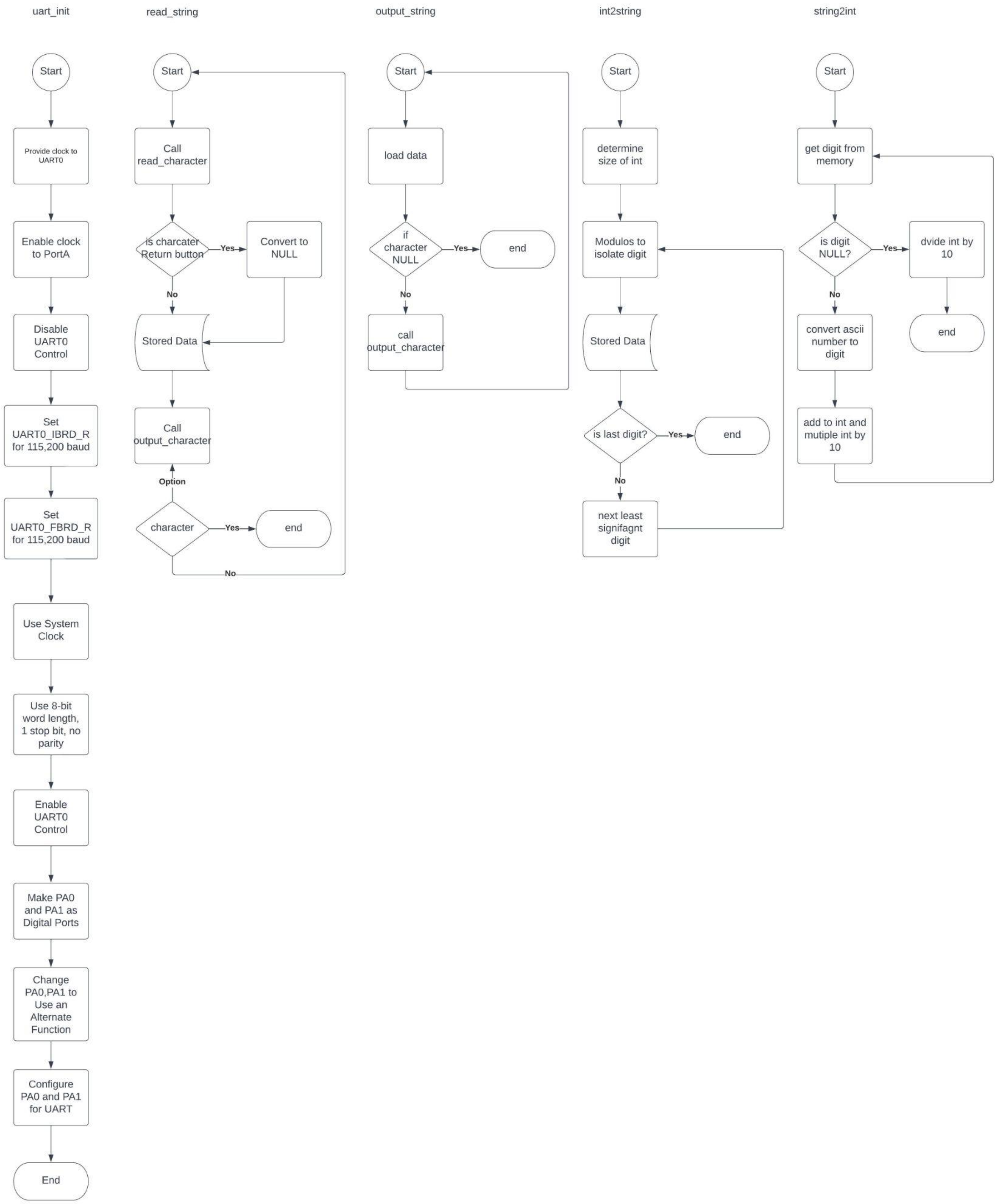
Calls checker to check for game end. Next mover is called to adjust the coordinates of the asterisk. Following that printer is called to print the gameboard and the asterisk. Finally the coordinates in coordinatesNext are stored in coordinateNow.

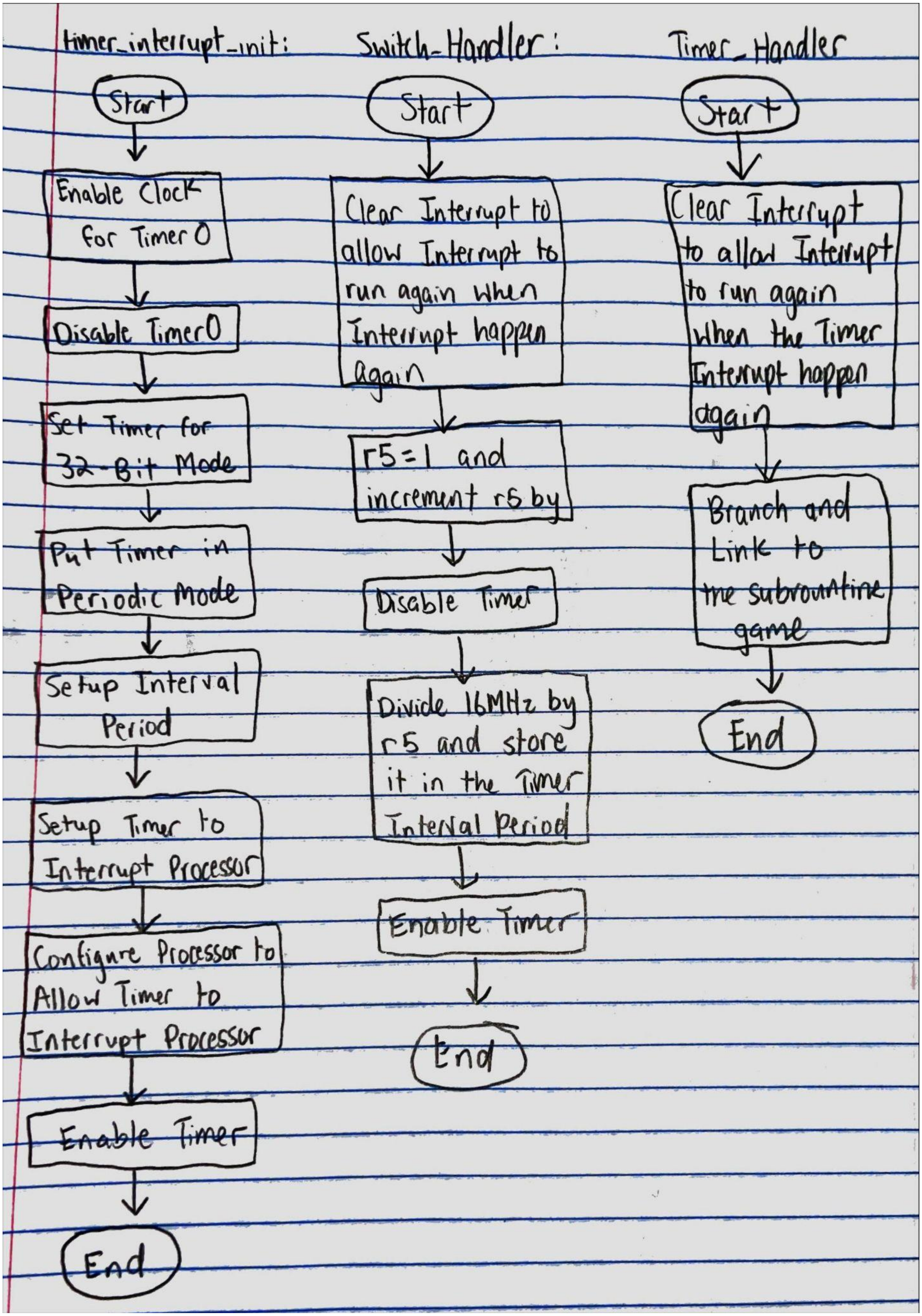
Section 4: Subroutine Flowcharts


uart_interrupt_init
Start
Configure the UART for Interrupt
Configure Processor to allow UART0 to interrupt Processor
End
gpio_interrupt_init
Start
Initalize SW1 on the Tiva Board by enabling clock, pin direction as input, pull-up resistor, and digitial pins on Port F Pin 4
Initalize Edge Sensitive, allow GPIO Interrupt Event Register to Control Pin, Setting the Interrupt for Falling Edge, enabling the Interrupt, and configuring processor to allow GPIO Port F to Interrupt Processor for Port F Pin 4
End

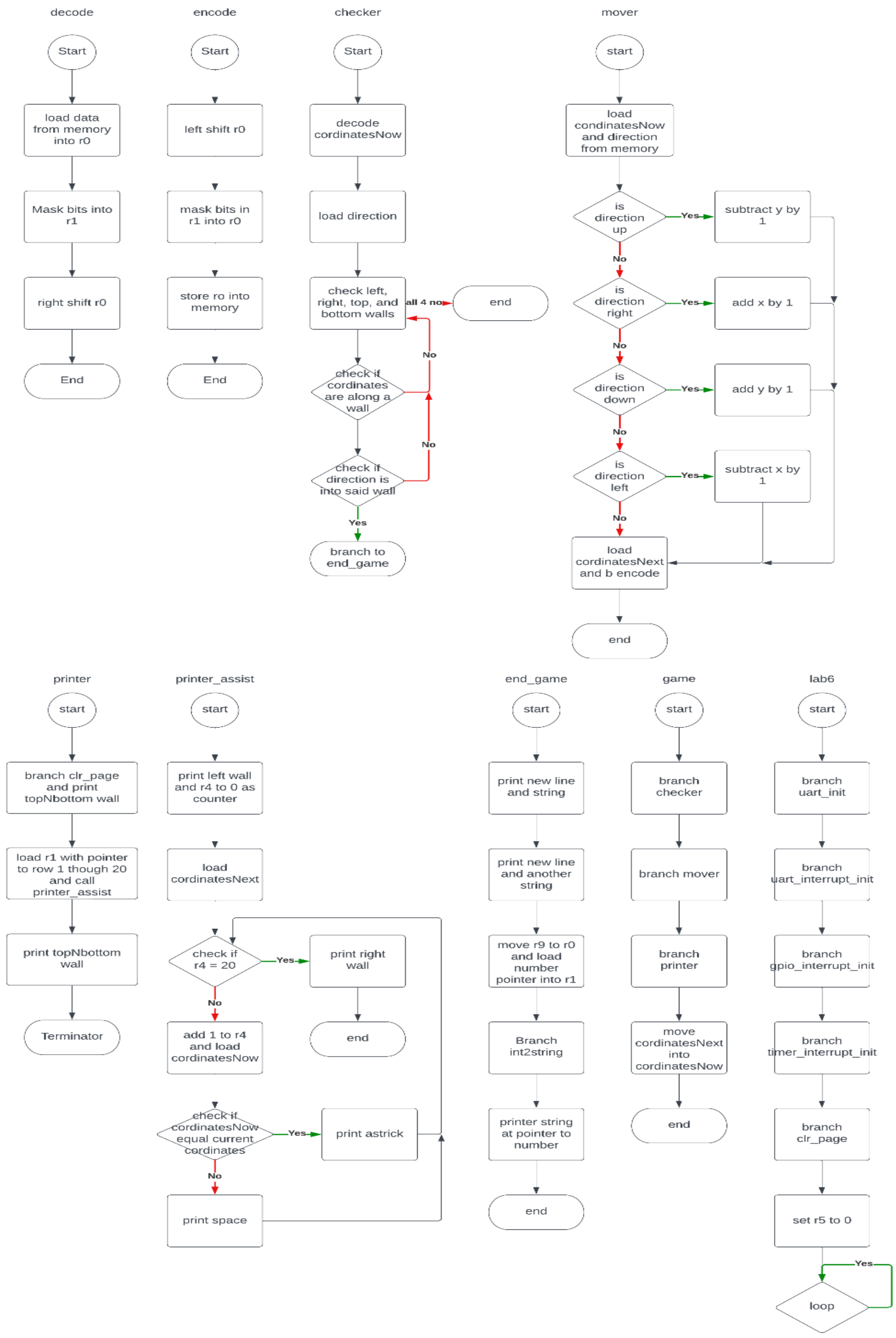
CSE-379 Lab 6 Documentation

From Lab5




timer_interrupt_init:
Start
Enable Clock for Timer 0
Disable Timer0
Set Timer for 32-Bit Mode
Put Timer in Periodic Mode
Setup Interval Period
Setup Timer to Interrupt Processor
Configure Processor to Allow Timer to Interrupt Processor
Enable Timer
End
Switch_Handler:
Start
Clear Interrupt to allow Interrupt to run again when Interrupt happen again
r5=1 and increment r5 by
Disable Timer
Divide 16MHz by r5 and store it in the Timer Interval Period
Enable Timer
End
Timer_Handler
Start
Clear Interrupt to allow Interrupt to run again when the Timer Interrupt happen again
Branch and Link to the subrountine game
End

CSE-379 Lab 6 Documentation


decode
Start
load data from memory into r0
Mask bits into r1
right shift r0
End
encode
Start
left shift r0
mask bits in r1 into r0
store ro into memory
End
checker
Start
decode cordinatesNow
load direction
check left, right, top, and bottom walls
all 4 no
end
check if cordinates are along a wall
No
check if direction is into said wall
No
Yes
branch to end_game
mover
start
load condinatesNow and direction from memory
is direction up
Yes
subtract y by 1
No
is direction right
Yes
add x by 1
No
is direction down
Yes
add y by 1
No
is direction left
Yes
subtract x by 1
No
load cordinatesNext and b encode
end
printer
start
branch clr_page and print topNbottom wall
load r1 with pointer to row 1 though 20 and call printer_assist
print topNbottom wall
Terminator
printer_assist
start
print left wall and r4 to 0 as counter
load cordinatesNext
check if r4 = 20
Yes
print right wall
end
No
add 1 to r4 and load cordinatesNow
check if cordinatesNow equal current cordinates
Yes
print astrick
No
print space
end_game
start
print new line and string
print new line and another string
move r9 to r0 and load number pointer into r1
Branch int2string
printer string at pointer to number
end
game
start
branch checker
branch mover
branch printer
move cordinatesNext into cordinatesNow
end
lab6
start
branch uart_init
branch uart_interrupt_init
branch gpio_interrupt_init
branch timer_interrupt_init
branch clr_page
set r5 to 0
Yes
loop