Lab 7

CSE 379

Enoch Anderson: enochand

Arianna Escobar-Reyes: aescobar

Lab Section: R3

04/20/2023

Table of Contents

Section 1: Division of Work

Enoch Anderson: Built interrupt handlers, update_game, move_cursor, print ball, print paddle, generate/remove/print bricks, detect wall/ground/brick collision, update scroe/lives/level, and some documentation.

Arianna Escobar-Reyes: Built random number generator, highl level game() function, restart_game_from_beggining, gpio/uart/clock initializations, lost live/restart lives, and majority of doucumentation.

Section 2: Program Overview

How to run the program:

1. Connect Putty to the appropriate COM# the Tivia C board is plugged into.
2. Create a new code composer studio project with the same configuration we've been using for the semester.
3. Copy and paste the code from the lab7main.c file into the main.c file code composer studio generated.
4. Place the lab7.s and lab7library.s files in the same directory as the main.c file generated by code composer studio.
5. Replace the tm4c123gh6pm_startup_ccs.c file with the file provided by this course.
6. Build and run the game.
7. In Putty you should see the prompt describing how to play the game and how to start.
    a. Choose to press SW2 to generate 1 row of bricks, SW3 to generate 2 rows of bricks, SW4 to generate 3 rows of bricks, or SW5 to generate 4 rows of bricks. While holding one of these buttons press any key on the keyboard to start the game.
8. At any point press SW1 to pause/unpause the game.

9. If you clear all 4 levels you beat the game, if you lose all 4 lives you get game over. Either way press ‘y’ if you would like to play again and any other key if you want to quit the game.

Section 3: Subroutine Descriptions

---

game:

Description: Starts the game when key is pressed

Input: Key pressed

Output: Game gets started

---

restart_game_from_beginning:

Description: The game gets restarted

Input: Key pressed

Output: At game over, if y is pressed then the game gets restarted, else quit the game.

---

increase_level_or_won:

Description: Called when all the bricks are destroyed. Either want to increase level or jump to end because they won

Input: None

Output: Increases level or jump to end because they won

---

generate_bricks:

Description: This function generates the desired number of bricks and stores them in the bricks memory value

Input: r0 = Desired number of bricks to generate, CANNOT BE BIGGER THAN 28 RIGHT NOW

Output: Bricks are generated and stored in bricks memory address

---

calc_next_brick_x_y:

Description: This function generates and returns the x and y coord of where the next brick should go

Input: Data at brickMinX, brickMaxX, brickMinY, curBrickX, curBrickY

Output: Data at brickMinX, brickMaxX, brickMinY, curBrickX, curBrickY

---

rand_int_from_0_to_4:

Description: Generates a random number between 0 and 4 (inclusive). Must be run after the timer is initialized. To do this we do: (Current value of GPTM Timer A for Timer 0 % 5)

Input: Current value of GPTM Timer A for Timer 0

Output: r0 = A number between 0 and 4 (inclusive)

---

get_timer_value:

Description: Gets the current value of GPTM Timer A for Timer 0

Input: The current value of GPTM Timer A for Timer 0

Output: r0 = The current value of GPTM Timer A for Timer 0

---

print_bricks:

Description: This function prints all the bricks stored in the bricks array

Input: Data at bricks array

Output: r0 = Bricks are printed to terminal

---

print_one_brick:

Description: This function prints 1 brick that's stored in r0

Input: r0 = Brick to print

Output: Brick printed to terminal

---

change_brick_color:

Description: This function injects the color passed in from r0 into the brick_string

Input: r0 = Brick to print

Output: Color is injected into brick string

---

reset_ball:

Description: This function clears ball from screen, resets ballCurX, ballCurY, dirX, dirY, and then prints ball to screen. It also restores the ball color to white.

Input: None

Output: Clears ball from screen, resets ballCurX, ballCurY, dirX, dirY, and then prints ball to screen.

---

reset_ball_color:

Description: This resets the ball color to be white

Input: None

Output: Ball color updated in ball string

---

change_ball_color:

Description:This changes the ball color to be the foreground color passed in through r0

Input: r0 = Foreground color of ANSI escape sequence

Output: Ball color updated in ball string

---

restart_score:

Description: This function resets the score back to 0, and also prints the updated score to terminal

Input: None

Output: Data at score and score_string updated

---

hide_cursor:

Description: Calling this function prints the ANSI escape sequence to hide the cursor

Input: Data at hide_cursor_string

Output: ANSI sequence to hide cursor is printed to the screen

---

print_lives:

Description: This function prints the current lives inside lives to the screen

Input: Data at memory addresses lives, lives_string, livesX, livesY

Output: Current lives is printed to the terminal

---

print_score:

Description: This function prints the current score inside score to the screen

Input: Data at memory addresses score, score_string, scoreX, scoreY

Output: Current score is printed to the terminal

---

print_level_number:

Description: This function prints the current level inside level_number to screen

Input: Data at memory addresses level_number, level_number_string, level_numberX, level_numberY

Output: Current level number is printed to the terminal

---

print_paddle:

Description: This function prints the paddle to the screen

Input: None, but padX and padY are used know where to print the paddle

Output: Paddle is printed to screen

---

move_cursor:

Description: This function moves the putty cursor to the position stored in r0 and r1. xpos = r0 ypos = r1. BOTH NUMBERS MUST BE BETWEEN 00 AND 99

Input:
r0 = desired x position of cursor ;
r1 = desired y position of cursor

Output: PuTTy cursor is moved to desired location

---

restart_lives:

Description: We get all 4 lives back (LEDS 3-0 are lit)

Input:
r0 = desired x position of cursor ;
r1 = desired y position of cursor

Output: PuTTy cursor is moved to desired location

---

lost_life:

Description: When you lose a life, shut off the correct LED on the daughter board and update the number of lives

Input: None

Output: Displays the updated number of lives and shuts off the correct LED

Switch_Handler:

Description: This function is called when SW1 is pressed to pause and unpause the game

Input: Data at memory addresses pause_game, paused_string, unpaused_string, pausedX, pausedY

Output: Game is paused or unpaused

Timer_Handler:

Description: After reaching the end of the time interval, the game board gets updated

Input: None

Output: The game board gets updated

UART0_Handler:

Description: Distinguishes if ‘a’ or ‘d’ is pressed, based on the key it’ll print the paddle to the correct area

Input: Key ‘a’ or ‘d’

Output: The paddle gets printed to the correct area

---

reset_terminal:

Description: Resets the terminal to all default values, but also clears terminal

Input: None

Output: Terminal is cleared

---

check_paddle_collision:

Description: Checks if the ball is colliding with the paddle, and updates dirX and dirY accordingly if so

Input: Memory values at ballCurX, ballCurY

Output: r0 set to 1 if there was a collision 0 otherwise. This is so you know to reprint the paddle for edge case ball is inside the paddle

---

check_paddle_y_range:

Description: Checks if the ball is in the y range for a collision

Input: None

Output: r0 set to 1 if there was a collision, 0 otherwise. This is so you know to reprint the paddle for edge case ball is inside the paddle

---

check_paddle_x_range:

Description: Only run if ball in y range, checks if ball is in x range for collision. Returns 1 in r0 if there is a collision

Input: None

Output: r0 set to 1 if there was a collision 0 otherwise. This is so you know to reprint the paddle for edge case ball is inside the paddle push {lr}

---

update_game:

Description: Displays the new frame for the game. Moves the ball, checks for brick collision, checks for wall collision, etc. If a brick collision happens, clear the brick.

Input: None

Output: Displays the new frame for the game. Moves the ball, checks for brick collision, checks for wall collision, etc. If a brick collision happens, clear the brick.

---

reprint_paddle:

Description: Reprints the paddle to the screen exactly where it is

Input: None

Output: Reprints the paddle to the screen exactly where it is

---

move_ball:

Description: Calculates the new position of the ball (based off ballCurX ballCurY dirX and dirY in memory) and moves it there

Input: Values in memory at ballCurX ballCurY dirX and dirY

Output: Ball is moved in PuTTy terminal, also ballCurX and ballCurY are updated in memory

---

update_ballCurX_ballCurY:

Description: Uses dirX and dirY to update the ballCurX and ballCurY values

Input: dirX and dirY values in memory

Output: ballCurX and ballCurY are updated in memory

---

check_wall_collisions:

Description: Checks if the ball is colliding with a wall, and updates dirX, dirY if so

Input: Data at memory locations ballCurX, ballCurY

Output: Data at dirX and dirY are updated if the ball does collide with a wall

---

inside_top_boarder:

Description: Sets the value at need_top_boarder_reprint to 1, so other functions know to reprint the top border

Input: None

Output: Sets the value at need_top_boarder_reprint to 1, so other functions know to reprint the top border

---

reprint_top_boarder:

Description: Reprints the top border when ball intersects with one of it's characters

Input: None

Output: Reprints the top border when ball intersects with one of it's characters

---

flip_dirX:

Description: Flips the value stored at dirX

Input: Data at dirX

Output: Data at dirX is flipped from positive to negative, or negative to positive

---

flip_dirY:

Description: Flips the value stored at dirY

Input: Data at dirY

Output: Data at dirY is flipped from positive to negative, or negative to positive

---

check_bottom_collision:

Description: Checks if the ball collides with the bottom of the board. If so it decreases the life count by 1

Input: Data at bottom_boarderY, ballCurY

Output: Lives decreased by 1 if ball collides with bottom

---

bottom_collision_happened:

Description: Only gets called when a bottom collision does occur. Updates lives, restarts ballCurX, ballCurY, dirX, dirY, and also pauses the game

Input: None

Output:

- Lives, ballCurX, ballCurY, dirX, dirY, padX, padY all restored to default
- Game is also paused

---

clear_paddle_from_screen:

Description: Clears the paddle from the screen, doesn't update padX or padY

Input: Data at padX and padY

Output: Paddle cleared from screen

---

check_inside_brick_collisions:

Description: Checks if ballCurX and ballCurY are inside a brick, updates dirX, dirY, clears brick, and removes the brick from bricks if so.

Input: Data at ballCurX, ballCurY, bricks

Output:

- If collision happens, dirX, dirY, brick cleared from terminal, and brick removed from bricks
- r0 = 1 if a collision happens, 0 otherwise
- r1 = 1 if a collision happens, and ball is right next to wall

---

check_single_inside_brick_collision:

Description: Checks if the ball is inside a brick pointed to by an r0 pointer. Updates dirX, dirY, clears brick, and removes the brick from bricks if so. Also changes ball color to the color of the brick it hit

Input: r0 = pointer to brick in memory

Output: If collision happens, dirX, dirY, brick cleared from terminal, and brick removed from bricks. r0 = 1 if a collision happened

---

inside_brick_collision_happened:

Description: Called when inside brick collision happens

Input: r0 = ptr to brick

Output: Ball color updated, brick removed from screen, brick removed from bricks. Tivia RGB LED switches color, score incremented push {lr, r4}

---

remove_brick:

Description: Removes the brick pointed to by r0 from the bricks array.

Input: r0 = pointer to a brick

Output: Brick pointed to is removed from bricks array in memory

---

erase_one_brick:

Description: Erases 1 brick that's stored in r0 from the terminal

Input: r0 = brick to erase (this is not a pointer to the brick it is the brick data)

Output: Brick erased from terminal

---

check_outside_brick_collisions:

Description: Checks if the ball is colliding above or below a brick in bricks. Updates dirX, dirY, clears brick, and removes the brick from bricks if so. Also changes ball color to the color of the brick it hit

Input: Data at bricks and ballCuX and ballCurY

Output: If collision happens, dirX, dirY, brick cleared from terminal, and brick removed from bricks

---

check_single_brick_above_below_collisoin:

Description: Checks if the ball colliding above or below a brick pointed to by r0 pointer. Updates dirX, dirY, clears brick, and removes the brick from bricks if so.

Input: r0 = pointer to brick in memory

Output: If collision happens, dirX, dirY, brick cleared from terminal, and brick removed from bricks. r0 = 1 if a collision happened, 0 otherwise

---

check_x_direction:

Description: Gets called if the ball is above the brick

Input: r0 = pointer to brick. r1 = brick data

Output: r0 gets set to 1 if ball collision happened

---

int2string_noNullTerm:

Description: stores a string of characters at memory location r1, without placing a null terminator at end.

Input: r0 = int to store r1 = memory location to store

Output: string of characters stored at r1

---

outut_string:

Description: outputs string pointed to by r0 to putty terminal.

Input: r0 = pointer to null terminated string.

Output: String printed in terminal.

---

read_character:

Description: reads character pressed from uart0 data register

Input: None

Output: r0 = character pressed

---

uart_interrupt_init:

Description: initialized the uart to interrupt processor

Input: None

Output: None

---

uart_init:

Description: initializes uart0

Input: None

Output: None

---

initialize_buttons:

Description: initializes sw2-5

Input: None

Output: None

---

illuminate_RGB_LED:

Description: Sets RGB led to be color passed into r0

Input: r0 = color

Output: RGB LED color changes

---

gpio_interrupt_init:

Description: initializes SW2-5, LED 0-3, RGB LED

Input: None

Output: None

---

clock_interrupt_init:

Description: initializes timer0, timerA

Input: None

Output: None

---

read_from_push_buttns_easy:

Description: returns 1 if sw2 pressed, 2 if sw3 is pressed, 3 if sw4 is pressed, 4 if sw5 is pressed.

Input: sw2-5 button pressed

Output: r0 = 1, 2, 3, or 4

Section 4: Flowcharts

Attached on the next page


game()
initialize everything, but don't enable interrupts
print welcoming prompts Must say how to play the game and press any key to start. Also say hold down button on daughter board to choose number of rows for game
Accept user keypress to start game
Branch to restart)game_from_beginning (only return from this when game is won or died)
did player win? Lives >0 if they won
game over play again?
You won play again?
Yes
No
print goodby and do final return
restart_game_from_beginning
clear terminal
reprint board
restore ball position
restore ball direction vectors
restore paddle position, clear and move paddle here
restore life count
restore score to 0
restore level to 1
print/display on daughter board lives score level number
Generate desired number of bricks
print bricks
restore timer speed and enable timer
enter Infinite game loop
Check if all bricks are destroyed
Check if lives == 0
Is level >= 4?
Call increase level function must -disable timer -recalcualte bricks and print them -increase speed and start timer
Disable All interrupts
return
increase_level_or_won
All the bricks have been destroyed. Do you want to increase a level or jump to the end?
Jump to end
You won!
Increase a level
Level increases
rand_int_from_0_to_4
Get the Timer A current value from Timer 0
That value % 5 and store in r0
get_timer_value
Get the Timer A current value from Timer 0 and store in r0
change_brick_color
r0 = Some color
Injects color in r0 into brick_string to be printed to terminal
print_bricks
Input is the data at bricks array
Print all the bricks stored in the bricks array
r0 = Bricks are printed to terminal
print_one_brick
r0 = Brick to print
Prints 1 brick that's stored in r0 to terminal
change_ball_color
Ball color gets changed to the foreground passed into r0
generate_bricks
r0 = The number of bricks
have we genearated enough bricks yet?
done, return
call make_one_brick
store brick in bricks
reset_ball
Clears the ball from the screen
Resets the ball X and Y position AND ball X and Y direction
Prints ball to screen in the color white
reset_ball_color
Set the ball color to white
Ball color gets updated in the ball string
restart_score
Sets the score back to 0
Prints the updated score to the terminal
hide_cursor
Input is the data at hide_cursor_string
Prints the ANSI escape sequence to hide the cursor
print_score
Prints the current score to the screen
print_level_number
Prints the current level inside level_number to the screen
move_cursor
r0 = desired x position of cursor r1 = desired y position of cursor
PuTTy cursor is moved to the location based on r0 and r1
print_lives
Prints the current live count to the screen
print_paddle
Prints the paddle to the screen
restart_lives
All 4 LEDS on the daughter board get lit and the live count is set to 4
lost_life
Decrement 1 from live count and shut down the correct LED light based on the new live count
reset_terminal
Resets the terminal to all the default values, but also clears terminal
check_paddle_y_range
Checks if the ball is in the y range for collision
r0 set to 1 if there was collision, 0 otherwise
check_paddle_x_range
Checks if the ball is in the x range for collision
r0 set to 1 if there was collision, 0 otherwise
reprint_paddle
Reprints the paddle to the screen exactly where it is
update_game
move ball based on dirx and dirY
reprint top boarder in case ball was inside it. Reprint left and right boarder in case ball was inside it
check if ball is colliding with paddle, update dirX and dirY if so.
check if ball is colliding with a brick, if it is -remove brick from screen -remove brick from bricks array -update dirX and dirY vectors
check if ball is colliding with wall, update dirX and dirY if so.
Timer_Handler
clear timer interrupt bit
call update_game
bx lr
Switch_Handler
clear interrupt clear register
load in data from paused_game in memory (0 if game not currently paused 1 if currently paused)
was paused_game == 0?
call unpause_game function
call pause_game function
bx lr
make_one_brick
get random number 0-4 for brick color
calcualte x and y position of brick
store data in 1 register as follows. byte0 = brick color byte1 = x position byte2 = y position byte4 = extra
return brick
calc_next_brick_x_y
load in curBrickX and curBrickY
Save numbers, because they are the values returned
is curBrickX + 3 < max X value?
curBrickX = minX value cruBrickY += 1
curBrickX = curBrickX + 3
store new curBrickX and curBrickY values
return values that were saved earlier in r0 and r1. r0 = X r1 = Y
UART0_Handler
clear interrupt clear register
read character pressed
verify character pressed was 'a' or 'd'
clear paddle from screen
calcualte new paddle positio based on character pressed
print paddle to screen at new location
bx lr
Pause_game
store 1 in pause_game memory address
disable timer interrupts
print "paused" to terminal
store currentl color of RGB led in memory
make RGB LED blue
return
unpause_game
set pause_game in memory to 0
clear "paused" string from terminal
restore RGB LED to color it was previously
enable timer interrupts
return
output_string
r0 points to string in memory.
Store pointer to string in r4, so it will be preserved across function calls
load next character to output into r0
Check if r0 contains NULL character
increment r4 to point to next character
call output_char function
return
read_character
PUSH lr
We use MOV and MOVT to load the UART0 address #0x4000C000
We save #0x10 in r1 to get the 5th bit which is where we find RxFE
We load bits from the flag register (#0x4000C018)
We use r2 AND r1 to determine if the bit we're looking at is 0
r2 == 0?
Load the register byte from r1 at address #0x4000C000 into r0
POP lr
output_character
We use MOV and MOVT to load the UART0 address #0x4000C000
We save #0x20 in r1 to get the 6th bit which is where we find TxFF
We load bits from the flag register (#0x4000C018)
We use r2 AND r1 to determine if the bit we're looking at is 0
r2 == 0?
Store the register byte in r0 to memory address #0x4000C000
inside_top_boarder
Set the value at need_top_boarder_repent to 1
reprint_top_boarder
When the ball intersects with the top border, reprints the top border
flip_dirX
Flips the value stored at dirX. (If it's positive, it's negative now. Vice versa)
flip_dirY
Flips the value stored at dirY. (If it's positive, it's negative now. Vice versa)
check_bottom_collision
Did the ball collide with the bottom of the board?
Decrease life count by 1
bottom_collision_happened
Sets ballCurX, ballCurY, dirX, dirY to default values. Game also gets paused with the lives updated
check_wall_collisions
With the memory locations ballCurX and ballCurY, check if the ball collided with a wall
Data at dirX and dirY are updated if the ball collides with a wall
check_single_inside_brick_collision
If the ball is inside a brick pointed to by an r0 pointers, update the dirX and dirY values, clear the brick, and remove the brick from the brick array. Change the ball color to the brick it hit
check_inside_brick_collisions
If the ballCurX and ballCurY are inside a brick, update dirX and dirY, clear the brick, and remove the brick from bricks array
inside_brick_collision_happened
Remove the collided brick from the screen
Change ball color to that brick
Tiva RGB changes color to new ball color
Score gets incremented
move_ball
With the values in memory at ballCurX, ballCurY, dirX, and dirY, create the new position of the ball
Update the memory values mentioned based on the new position and move the ball in PuTTy
update_ballCurX_ballCurY
Use the dirX and dirY to update the ballCurX and ballCurY values
clear_paddle_from_screen
The paddle gets cleared from the screen. padX and padY doesn't get updated
remove_brick
Removes brick pointed to by r0 from the brick array
erase_one_brick
Erases 1 brick stored in r0 from terminal
check_outside_brick_collisions
Checks if the ball is colliding above or below a brick in bricks
Updates dirX and dirY
Clears brick
Removes the brick from bricks
Ball color changes to the color of the brick it hit
check_single_brick_above_below_collisoin
Checks if the ball colliding above or below a brick pointed to by r0
Updates dirX and dirY
Clears brick
Removes the brick from bricks
r0 = 1 if collision happened, 0 otherwise
check_x_direction
Gets called if the ball is the brick
r0 gets set to 1 if collision happened, else 0
int2string
lr, r4, r5, r6 are PUSHED
Copy the address of r0 into r4 to preserve the address
Copy the address of r1 into r6 to preserve the address
We want to remember our inputs and use temporary registers in our loop. That's why we're doing this :)
We call count_digits to get the number of digits in the input and store in r0
Doing this will avoid an IndexOutOfBounds error
To get the leftmost digit, we subtract 1 from r0 and save it in r0
Save the value of r0 into r5 since we'll overwrite r0 in our loop
r5 == -1?
Save the input integer in r0
Save the index for the digit we want to get in r1
Call get_int_at_position, which will get the digit at position r1 from the number in r0
After the function call, the digit is stored in r0. We save the ASCII value in r0 after adding 48 to r0
We store the new value of r0 into the r6's memory address
We add 1 to r6 and save the new address to r6. This will give us the next address to store our next digit
Subtract 1 from r5 and store it in r5. This will get the new index of the next digit
We create our NULL terminator by saving #0 into r0
This ensures that our output is in string format
Store the value of r0 into the r6's memory address
lr, r4, r5, r6 are POPPED
The string format of the input integer is in the memory addresses we want
check_paddle_collision
Paddle is made up of 5 characters, I call them LL LM MM RM RR
Is ball right above or equal to the paddle's Y position?
Check if ball is in LL position.
dirX = -1 dirY = -1
Check if ball is in LM position.
dirX = -1 dirY = -2
Check if ball is in MM position.
dirX = 0 dirY = -1
Check if ball is in RM position.
dirX = 1 dirY = -2
Check if ball is in RR position.
dirX = 1 dirY = -1
return
uart_interrupt_init
Set the Receive Interrupt Mask (RXIM) bit to 1 in the UART Interrupt Mask Register (UARTIM)
Set bit 5 to 1 in the Interrupt 0-31 Set Enable Register (EN0)
simple_read_character
Read data at data reg for UART0
Return
read_from_push_btns
Store memory address of port D in r1
Store data from data register of port D in r2, uisng r1 and constant GPIODATA_OFFSET
Because we want a 1 when the button is pressed. Bits 0-3 will be set with a 1 corresponding with what button is pressed
MOV r0, r2 return
uart_init
initialize_Buttons
Set clock for port D
Set direction register for pins 0-3 to be input
Set pins 0-3 to be digital
Set pull up resistors for pins 0-3
illuminate_RGB_LED
Input: r0 as one of the constant colors
Constant colors are #RED, #GREEN, #BLUE, #YELLOW, #WHITE, #PURPLE
Copy the Port F address (#0x40025000) into r1
Load the register byte from #0x400253FC to r2
Use BFI to overwrite pins 1-3 with the bits in r0. The result is stored in r2
We set constant colors mentioned above to the bits we want in pin 1-3
Store the register byte of r2 into address #0x400253FC
The color of your choice is displayed on the Tiva board
gpio_interrupt_init
Initialize clock for Port F
Initialize direction register for Port F
Initialize digital register for Port F
Initialize pull up register for Port F
Initialize GPIO Interrupt Sense Register (GPIOSIS)
Enable edge sensitive by setting Pin 4 to 0
Initialize GPIO Interrupt Both Edges Register (GPIOIBE)
Allow GPIOEV to determine edge by setting pin 4 to 0
Initialize GPIO Interrupt Event Register (GPIOIV)
Set Pin 4 to high (rising edge) by setting pin 4 to 1
Initialize GPIO Interrupt Mask Register (GPIOIM)
Enable interrupt by setting pin 4 to 1
Initialize Interrupt 0-31 Set Enable Register (EN0)
Enable bit 30 by setting it to 1
clock_interrupt_init:
Connect the clock to timer by setting bit 0 of address #0x400FE604 to 1
Disable timer by setting bit 0 of address #0x4003000C to 0
Put timer in 32-bit mode by setting bits 0-2 of address #0x40030000 to 0
Put timer in periodic mode by setting bits 0-1 of address #0x40030004 to 2
Setup interval period by updating all 32 bits in address #0x40030028 to 16,000,000 cycles
Enable timer to interrupt processor bu updating bit 0 of address #0x40030018 to 1
Enable timer by setting bit 0 of address #0x4003000C to 1