The Vending Machine - Code Documentation

By
Umang Deshpande and Akshay Hegde

June 2017

1 Program Code

The git repository for the complete code can be found here. It contains the complete project folder used in CCS.

- The console support libraries are present in the Console folder.
- The font and graphic libraries are present in the Images folder.
- VM_RTOS.c is the main file. This contains the Statechart and RTOS implementation of the vending machine abstraction.

2 Main Code

Firstly, following headers are included:

```
/* XDC module Headers */
#include <xdc/std.h>
#include <xdc/runtime/System.h>


//needed for any Log_info() call
#include <xdc/runtime/Log.h>
//header file for statically defined objects/handles
#include <xdc/cfg/global.h>
//needed for any Log_info() call
#include <xdc/runtime/Log.h>


/* BIOS module Headers */
#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Clock.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
```

```
/* Standard CHeader files*/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>

/* Custom Game Console Header */
#include "Console/console.h"
#include "Console/glcd.h"

/* Timer and GPIO function Headers */
#include "driverlib/timer.h"
#include "inc/hw_memmap.h"

/*For ROM Functions*/
#define TARGET_IS_BLIZZARD_RB1
#include "driverlib/rom.h"

/*
 * This Header files for all custom imagery
   including smiley, soda can, coin graphic.
 */
#include "Images/images.h"
```

- XDC Headers handle XDCTools as part of RTOS, which aid in debugging.
- BIOS module headers allow use of Tasks and Semaphore definitions.
- `stdint.h` Headers allow use of `uint32` variables, stdbool allows use of `stdbool.h` variables, `string.h` allows manipulation of string in string variables, `time` allows for use of the time functions for block randomization.
- The Console headers provide developer level functions to the console, so that initialization can be performed using a single function `_init_()`. And GLCD functions for direct display of fonts and graphic are obtained from `glcd.h` header.
- For the timer and timer interrupt functions used in Timer ISR(Task Scheduler), `timer.h` and `hw_memmap.h` are used.
- `rom.h` Allows the use of ROM functions for the given board.
- `images.h` Contains various different images in the form of hex arrays to be displayed on screen.

Following this, define all global variables are defined as follows:

```
/*
 * Global Variables
 */

uint32_t latency, pinName, baseName, flag;
volatile uint32_t tickCount=0;
signed int sum;
char digits[4] = "00c";
char text[];
unsigned char p, ch, reg_n, diet_n, holder, character;
unsigned char temp[8];
```

- `pinName`,`baseName` and `flag` are used for Switch selection. `baseName` is the port to which the switch belongs to. `pinName` is the actual pin to which the switch is connected.
- `tickCount` is used for Task Scheduling in `Timer_ISR`.
- `sum` holds the total money entered into the vending machine
- `digits[4]` is used to hold the sum entered in string format to display total money entered on GLCD.
- `text[]`,`ch`, `temp[8]` and `character` hold text to be displayed on GLCD.
- `reg_n` and `diet_n` holds total number of regular and diet sodas requested by the user.

All the different states for the various state machines used are enlisted as follows:

```
/*
 * Enumeration of States for State Machine
 */
enum vm_modes{

    // States for Vending Machine

    INITM,
    COIN,
    SELECT,
    DELIVERY
};
// Initialization
enum vm_modes vm_mode = INITM;
```

`vm_mode` enumerated variable is thus initially in `INITM` state.

```
  enum inputs{

    // Different input transitions enumeration
```

```
    INITI,
    DIME,
    NICKEL,
    QUARTER,
    NONEI
};
// Initialization
enum inputs input = INITI;
```

`input` enumerated variable is thus initially in `INITI` state.

```
enum outputs{

    //Output internal states enumeration

    INITO,
    REGULAR,
    DIET,
    CHANGE,
    NONEO
};

//Initialization
enum outputs output = INITO;
```

`output` enumerated variable is thus initially in `INITO` state.
Following this, various functions are defined as follows:

- `coinScreenInput()`. This function detects which coin has been entered through switch press detection. Handles inputs in COIN state. Calculate sum based on input transition. Also transition to next Vending Machine state.

```
void coinScreenInput()
{
    // Update the total Sum to be displayed
    digits[1] = (sum%10) + '0';
    digits[0] = (sum/10) + '0';
    if(detectKeyPress(0) == 1)
    {
        // UP Switch Pressed
        // -> Entered Dime
        input = DIME;
        sum += 5;
        micros(50);
        glcd_clearDisplay();
    }
    else if(detectKeyPress(2) == 1)
    {
```

```
        // DOWN Switch Pressed
        // -> Entered Nickel
        input = NICKEL;
        sum += 10;
        micros(50);
        glcd_clearDisplay();
    }
    else if(detectKeyPress(4) == 1)
    {
        // HAT Switch Pressed(On Thumbstick)
        // -> Entered Quarter
        input = QUARTER;
        sum += 25;
        micros(50);
        glcd_clearDisplay();
    }
    else if(detectKeyPress(1) == 1)
    {
        // RIGHT Switch Pressed
        // -> Go to Selection Screen
        vm_mode = SELECT;
    }
}
```

- `coinScreenDisplay()`.This function displays GUI to user in the COIN state. Handles GLCD. Changes graphic and text displayed based on input transition.

```
void coinScreenDisplay()
{
    switch(input){
    case INITI:
        // Display Home Screen with COIN graphic
        // and text. Switch to No Coin Inserted State.
        glcd_write(coin);
        displayText("  5c>Press UP    10c>Press DOWN
        25c>Press HAT  Next>Press RIGHT", 4);
        input = NONEI;
        break;
    case DIME:
        // DIME input detected. Total Money
        // Entered Displayed. Further provision
        // to enter more coins.
        displayText(digits, 0);
        displayText("Entered
        5c>Press UP     10c>Press DOWN  25c>Press HAT
        Next>Press RIGHT", 1);
        break;
    case NICKEL:
        // NICKEL input detected. Total Money
```

```
        // Entered Displayed. Further provision
        // to enter more coins.
        displayText(digits, 0);
        displayText("Entered
        5c>Press UP     10c>Press DOWN  25c>Press HAT
        Next>Press RIGHT", 1);
        break;
    case QUARTER:
        // QUARTER input detected. Total Money
        // Entered Displayed. Further provision to enter
        // more coins.
        displayText(digits, 0);
        displayText("Entered
        5c>Press UP     10c>Press DOWN  25c>Press HAT
        Next>Press RIGHT", 1);
        break;
    case NONEI:
        // No Input detected.
        break;
    }
}
```

- `selectScreenInput()`. This function selects soda based on switch press. Can select multiple soda at once. Dispensed based on money entered. Handles Input in SELECT state. Also transition to next Vending Machine state.

```
void selectScreenInput()
{
    if(detectKeyPress(3) == 1)
    {
        // LEFT Switch Pressed
        // -> Selected Regular Soda, cost 35c.
        sum -= 35;
        reg_n++;
        output = REGULAR;
        glcd_clearDisplay();
    }
    else if(detectKeyPress(1) == 1)
    {
        // RIGHT Switch Pressed
        // -> Selected Diet Soda, cost 35c.
        sum -= 35;
        output = DIET;
        diet_n++;
        glcd_clearDisplay();
    }
    else if(detectKeyPress(0) == 1)
    {
        // UP Switch Pressed
```

```
        // -> Abort Transaction. No Soda Selected.
        output = CHANGE;
        glcd_clearDisplay();
    }
    else if(detectKeyPress(2) == 1)
    {
        // DOWN Switch Pressed
        // -> Continue to Delivery State.
        vm_mode = DELIVERY;
    }
}
```

- `selectScreenDisplay()`. This function displays GUI in SELECT state. Change screens based on Soda Selection.

```
void selectScreenDisplay()
{
    switch(output)
    {
    case INITO:
        // Display Initial Selection Screen, with
        // Soda Can Graphic.
        glcd_clearDisplay();
        display40x32(0, 3, soda_can);
        displayText("Reg>Press LEFT
        Diet>Press RIGHTCancel>Press UP", 5);
        output = NONEO;
        break;
    case REGULAR:
        // Regular Soda Selected Screen.
        displayText("Selected Regular
        Reg>Press LEFT  Diet>Press RIGHTContinue>Press
        DOWN", 2);
        break;
    case DIET:
        // Diet Soda Selected Screen.
        displayText("Selected Diet
        Reg>Press LEFT  Diet>Press RIGHTContinue>Press
        DOWN", 2);
        break;
    case CHANGE:
        // Abort transaction, no soda selected Screen.
        displayText("Abort Transaction
        Continue>Press  DOWN", 2);
        break;
    case NONEO:
        // Undefined state.
        break;
    }
}
```

- `deliveryOutput()`. This function displays GUI in DELIVERY state. Additionally, also in charge of LED blink delivery indication. Certain screens are displayed based on dispensed soda and change, in addition to LED blinks.

```
void deliveryOutput()
{
    switch(output){
    case REGULAR:
        // Regular Soda Output
        if(sum >= 0)
        {
            // Provided enough coins entered,
            // Display Soda Delivery LED blink
            // and GLCD Screen with Smiley Graphic.
            glcd_clearDisplay();
            display40x32(1, 3, smiley);
            displayText("  Please Enjoy
            Your Drink!", 5);
            while(reg_n > 0)
            {
                // Blink LED as many times as
                // number of Regular Sodas Selected.
                ledON(1);
                millis(1000);
                ledOFF(1);
                millis(1000);
                reg_n--;
            }
        }
        else
        {
            // If coins are not sufficient for
            // the transaction, display refusal screen.
            sum =+ ((diet_n+reg_n)*35);
            glcd_clearDisplay();
            displayText(" Sorry, unable to dispense.
            Insufficient Cash", 2);
        }
        // Switch to CHANGE output state.
        output = CHANGE;
        break;
    case DIET:
        // Diet Soda Output
        if(sum >= 0)
        {
            // Provided enough coins entered,
            // Display Soda Delivery LED blink
            // and GLCD Screen with Smiley Graphic.
            glcd_clearDisplay();
```

```
            display40x32(1, 3, smiley);
            displayText("  Please Enjoy
            Your Drink!", 5);
            while(diet_n > 0)
            {
                // Blink LED as many times as
                // number of diet sodas selected.
                ledON(2);
                millis(1000);
                ledOFF(2);
                millis(1000);
                diet_n--;
            }
        }
        else
        {
            // If coins are not sufficient for the
            // transaction, display refusal screen.
            sum += ((diet_n+reg_n)*35);
            glcd_clearDisplay();
            displayText(" Sorry, unable  to dispense.
            In-sufficient Cash", 2);
            millis(2000);
        }
        output = CHANGE;
        break;
    case CHANGE:
        // Deliver change output screen display
        glcd_clearDisplay();
        glcd_write(coin);
        displayText("    Collect         Change", 6);
        millis(2000);
        // Continue to blink LED until all change
        // is returned. Change is given as 5c each
        //per LED blink.
        while(sum > 0)
        {
            ledON(3);
            millis(500);
            ledOFF(3);
            millis(500);
            sum -= 5;
        }
        // Display Thank you screen.
        glcd_clearDisplay();
        displayText(" Thank you for  using SodVen
         Vending Machine!", 2);
        millis(2000);
        glcd_clearDisplay();
        // Switch Back to Initial State.
```

```
            vm_mode = INITM;
            break;
        }
}
```

- `updateOutput()`. This implements the overall vending machine statechart for the outputs, within it and calls various other functions in different states. Switching State Machine Implementaton in charge of outputs(GLCD and LEDs). Switches between various screens in different Vending Machine states. Is implemented as a task in RTOS, initialized with OutputSem Semaphore.

```
void updateOutput(void)
{
    while(1)
    {
        Semaphore_pend(OutputSem, BIOS_WAIT_FOREVER);

        // Output Control State Machine.
        switch(vm_mode){
        case INITM:
            // Resets various variables.
            sum = 0;
            reg_n = 0;
            diet_n = 0;
            vm_mode = COIN;
            input = INITI;
            output = INITO;
        case COIN:
            coinScreenDisplay();
            break;
        case SELECT:
            selectScreenDisplay();
            break;
        case DELIVERY:
            deliveryOutput();
            break;
        }
    }
}
```

- `readInput()`. This function also implements the overall vending machine statechart but for inputs and calls various other functions in different states. Switching State Machine Implementaton in charge of inputs(Switch presses). Has functional descriptions for different states. Is implemented as a task in RTOS, initialized with SwitchSem Semaphore.

```
void readInput(void)
{
    while(1)
```

```
    {
        Semaphore_pend(SwitchSem, BIOS_WAIT_FOREVER);

        // Input Control State Machine.
        switch(vm_mode){
        case INITM:
            break;
        case COIN:
            coinScreenInput();
            break;
        case SELECT:
            selectScreenInput();
            break;
        case DELIVERY:
            // No user input in delivery state
            break;
        }
    }
}
```

`main()` function serves to initialize and start the BIOS. The BIOS handles program execution. Now, the `main()` function is defined as follows:

```
int main(void)
{
    /* Set display refresh latency */
    latency = 40;

    /* Initialize all peripherals, GPIOs,
    Interrupts and GLCD. */
    _init_();
    glcd_init();
    glcd_clearDisplay();

    /* Turn off all LEDs*/
    ledOFF(1);
    ledOFF(2);
    ledOFF(3);
    ledOFF(4);

    /* Start BIOS */
    BIOS_start();

    return (0);
}
```

Finally, the `Timer_ISR()` is defined, which implements the task scheduling as shown:

```
void Timer_ISR(void)
{
    // Clear Timer Interrput
    ROM_TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
    // Increment tickCount used for Scheduling
    tickCount++;
    // Task Scheduler
    switch(tickCount){
    case 15:
        // Post Switch Semaphore
        Semaphore_post(SwitchSem);
        break;
    case 30:
        // Post Output Semaphore
        Semaphore_post(OutputSem);
        // Reset tickCount
        tickCount = 0;
        break;
    }
}
```

This is the complete code for the Vending Machine Controller, with a state machine implementation.