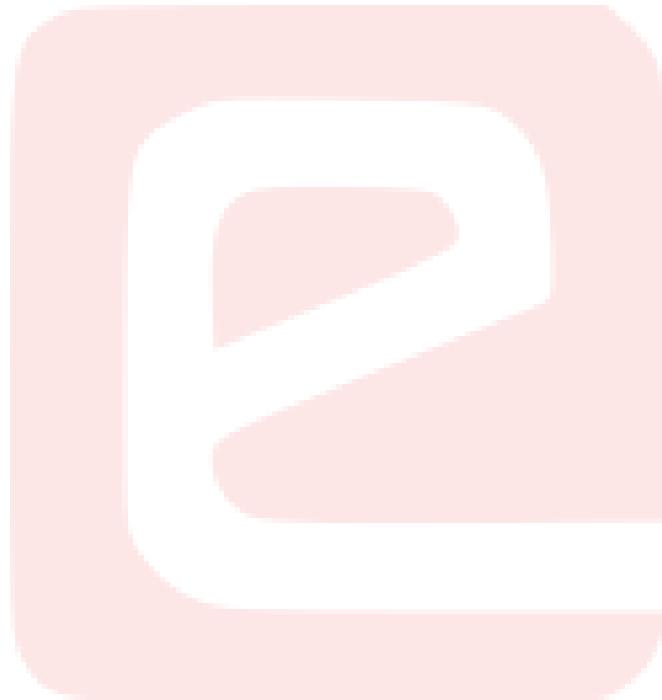


eYSIP - 2017

# GAME DEVELOPMENT WITH TI-RTOS



Akshay Hegde  
Umang Deshpande  
Sanam Shakya  
Vishwanathan Iyer

Duration of Internship: 22/05/2017 – 07/07/2017

2017, e-Yantra Publication

# Game Development with TI-RTOS

## Abstract

Real Time Systems is the reactive embedded systems where system has to perform various tasks within timeline. There are various software architecture for programming Real Time Systems. So this project deals with various concepts of Real Time Systems, Statechart design principles and developing various exercises based on TI-RTOS and Switch Case Statecharts. This project aims at developing software modules which will form the basis for any game that can be developed on the console.

## Completion Status

The requisite software modules for game development have been created successfully. Also game design approaches are fully covered. The following modules have been developed:

- Flexible GLCD library for game objects display, robust Font library in two different sizes for display of text on GLCD and Tones library for playing any music and beeps using onboard buzzer,
- Working integration of statecharts with TI-RTOS for easier task scheduling.
- Statechart implementation exercises for an insight into the design approach, including *The Vending Machine Controller*(with RTOS) and *Timed Bomb Controller*(without RTOS) have been developed.
- Completely implemented *Breakout* Game designed using Statechart Approach, implemented using Switch Case construct and TI-RTOS.(Some minor bugs exist, but should not hinder understanding of game design)



## 1.1. HARDWARE PARTS

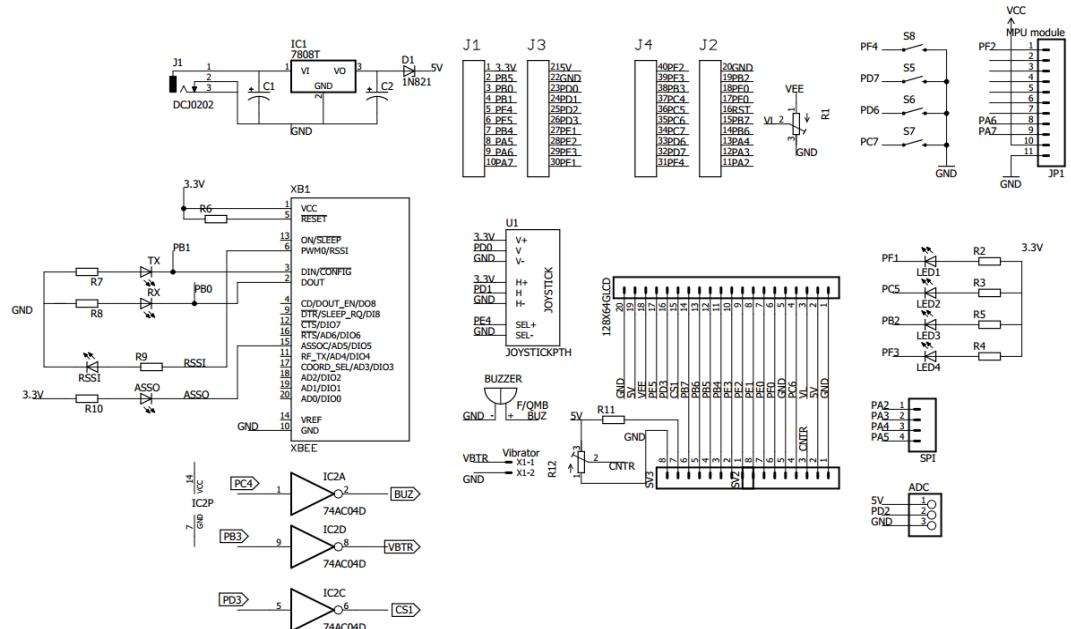
### 1.1 Hardware parts

- List of hardware
  - Graphical LCD
  - Buzzer
  - Joystick
  - Switches
  - LEDs
  - Vibration Motor
  - TIVA C Series TM4C123G6HPM
  - IC 7404
- Detail of each hardware:
  - TIVA C Series TM4C123G6HPM
    - \* High Performance TM4C123GH6PM MCU
    - \* 80MHz 32-bit ARM Cortex-M4-based microcontrollers CPU
    - \* 256KB Flash, 32KB SRAM, 2KB EEPROM
    - \* Two Controller Area Network (CAN) modules
    - \* USB 2.0 Host/Device/OTG + PHY
    - \* Dual 12-bit 2MSPS ADCs, motion control PWMs
    - \* 8 UART, 6 I2C, 4 SPI
    - \* On-board In-Circuit Debug Interface (ICDI)
  - Graphical LCD
    - \* 128x64 Display.
    - \* 20 pins.
    - \* It has 128 coloums and 8 pages.Each page is 8 bit wide.
    - \* It is divided into 2 halfs, each half is controlled by a controller KS0108.
    - \* Writing is done page wise.
  - Buzzer
    - \* Rated voltage 3V.2-5V MAX Limits.
    - \* Rated Current 30mA
    - \* Sound Level at 10cm 85dB



## 1.1. HARDWARE PARTS

- \* Frequency 2300+/-300Hz
- Joystick
  - \* 2 Axis Joystick.
  - \* ADC Output.
  - \* 2 Potentiometer.
  - \*
- Vibration Motor
  - \* An eccentric rotating mass vibration motor (ERM).
  - \* Uses a small unbalanced mass on a DC motor, when it rotates it creates a force that translates to vibrations.
  - \* A linear resonant actuator (LRA) contains a small internal mass attached to a spring, which creates a force when driven.
- Switches
- LEDs
- IC 7404
- Connection diagram Game Console





## 1.2. SOFTWARE USED

- Links
  - [GLCD Datasheet](#)
  - [IC 7404 Datasheet](#)

## 1.2 Software used

### 1.2.1 Code Composer Studio

*Code Composer Studio (CCStudio or CCS)* is an integrated development environment (IDE) to develop applications for Texas Instruments (TI) embedded processors. It has complete Windows, Linux and Mac<sup>1</sup> support for the entire Texas Instruments portfolio, including the board used for the project, i.e., Tiva C TM4C123GH6PM Launchpad.

#### Version Used

Code Composer Studio Version:7.1.0.00016 is used throughout the project. Installation of RTOS, and training material are accessible directly through the Resource Explorer in this version.

#### Downloading CCS

Download the Windows installer from

1. [CCS\(Windows\) Web Installer](#) or
2. [CCS\(Windows\) Offline Installer](#).

Download the macOS installer from

1. [CCS\(macOS\) Web Installer](#) or
2. [CCS\(macOS\) Offline Installer](#).

Linux versions can be installed from [CCS All Versions Download Page](#).

The web installer will require Internet access until it completes. If the web installer version is unavailable or you cant get it to work, download, unzip and run the offline version. The offline download will be much larger than the installed size of CCS since it includes all the possible supported hardware.

<sup>1</sup>Please note that only microcontroller and connectivity devices are supported on Mac. Processors devices are not supported.

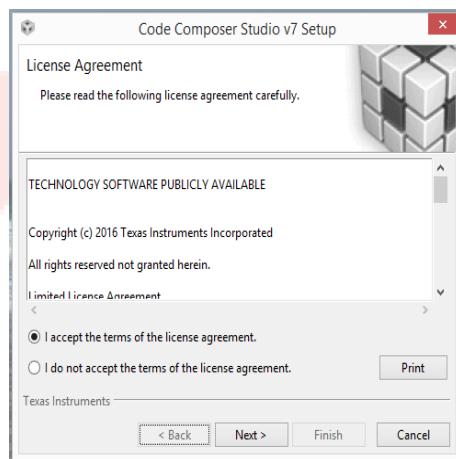


## 1.2. SOFTWARE USED

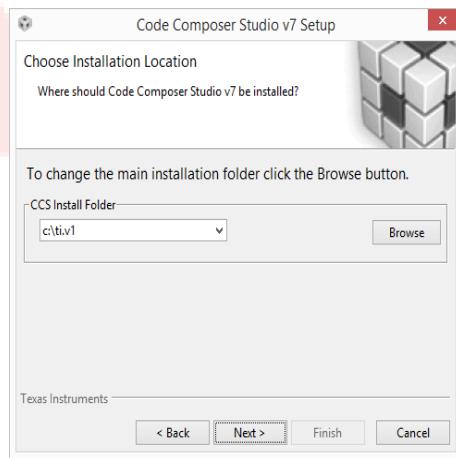
### Installing CCS(Windows)

Double click on the downloaded installer, after the installer has started, follow the steps mentioned below:

1. Accept the Software License Agreement and click Next.



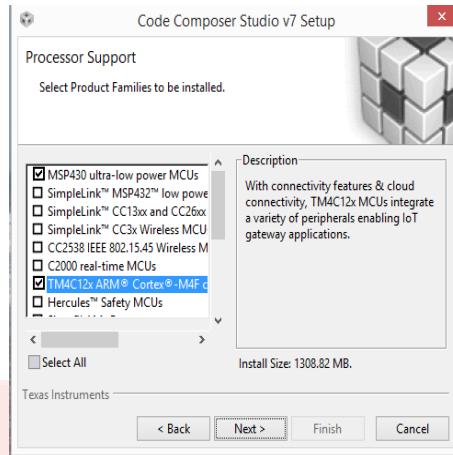
2. Select the destination folder and click next.



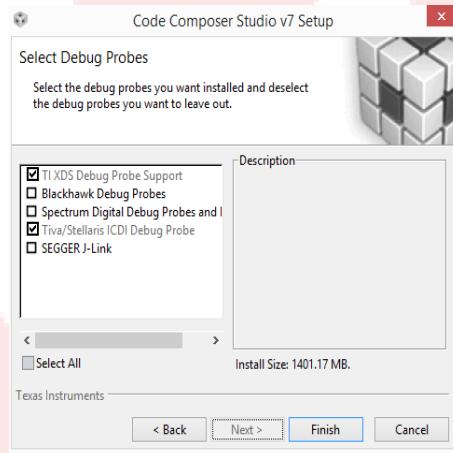
3. Select the processors that your CCS installation will support. You must select "TM4C12X Arm Cortex M4". You can select other architectures, but the installation time and size will increase.



## 1.2. SOFTWARE USED



4. Select debug probes and click finish



5. The installer process should take 15 - 30 minutes, depending on the speed of your connection. The offline installation should take 10 to 15 minutes. When the installation is complete, uncheck the Launch Code Composer Studio v7 checkbox and then click Finish. There are several additional tools that require installation during the CCS install process. Click Yes or OK to proceed when these appear.
6. Download and install the latest full version of TivaWare from: [Tivaware Install Link](#). The installation is straightforward, and the driver library from Tivaware is used for development.



## 1.2. SOFTWARE USED

### Installing CCS(macOS)

Unzip the downloaded zip package, and double click on ccs\_setup\_xxx.app. The installer opens, and refer to steps 1-5 of Windows Installation, it is identical.

For Tivaware installation, download the .exe from [Tivaware Install Link](#), unzip using Unarchiver and copy the folder to the ti install location at /Applications/ti. Thus, Tivaware is set to be used on macOS.

### Installing TI-RTOS

TI-RTOS can be installed directly from CCS App Center(recommended)

1. In the Search Box, search for RTOS.
2. Press Install.
3. Restart CCS.

Additionally, RTOS can be directly downloaded and installed from the [TI-RTOS Download Page](#), for the desired Operating System.

You can find additional information at these websites:

Launchpad Home page:

<http://www.ti.com/launchpad>

Tiva C Series TM4C123G LaunchPad:

<http://www.ti.com/tool/ek-tm4c123gxl>

TM4C123GH6PM Resources:

<http://www.ti.com/product/tm4c123gh6pm>

LaunchPad Wiki:

<http://www.ti.com/launchpadwiki>

### 1.2.2 Mikroelektronika GLCD Font Creator

GLCD Font Creator enables the creation of personalized fonts, symbols and icons for LCDs and GLCDs. Create fonts and symbols from scratch, or by importing existing fonts on your system. It lets you modify and adjust them for your needs, apply effects and finally export them as source code for use in mikroC, mikroBasic or mikroPascal compilers.

#### Downloading and Installing Mikroelektronika GLCD Font Creator

Download the Windows Installer from their [official website](#). The Install Wizard is pretty straightforward.



### 1.3. SOFTWARE AND CODE

For macOS, download the same .exe. Then use Wine to open it (Tutorials on installing and using Wine on macOS can be found [here](#)).

#### 1.2.3 Hobbytronics BMP-LCD Converter

This software converts a 2 colour BMP file into the hex character array that can be used in a program to display the graphic on the LCD screen. It can be downloaded from [here](#). It requires no installation.

## 1.3 Software and Code

Three software systems are implemented during the course of the project. They are:

1. Timed Bomb Controller
2. The Vending Machine Controller
3. The Breakout Game

### 1.3.1 Timed Bomb Controller

### 1.3.2 Problem Statement

The time bomb has a control panel with an GLCD that shows the current value of the timeout and three buttons: UP, DOWN, and ARM. The user begins with setting up the time bomb using the UP and DOWN buttons to adjust the timeout in one-second steps. Once the desired timeout is selected, the user can arm the bomb by pressing the ARM button. When armed, the bomb starts decrementing the timeout every second and explodes when the timeout reaches zero. An additional safety feature is the option to defuse an armed bomb by entering a secret code. The secret defuse code is a certain combination of the UP,DOWN and LEFT buttons. Of course, the defuse code must be correctly entered before the bomb times out.



### 1.3. SOFTWARE AND CODE

#### 1.3.3 StateChart Solution

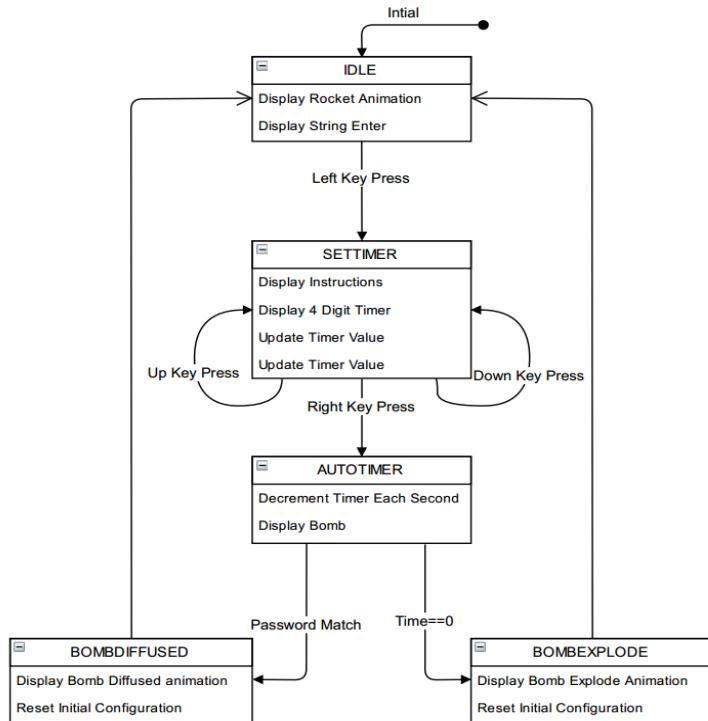


Fig a: TimerState Statechart

From the problem statement, the states of Timer Bomb over which its behaviour changes considerably are taken to be individual states of Timer Bomb. In this case, the Bomb Timer problem can be segregated into:

- **Idle State:**

Perform basic initialization of 4 digit Timer, reset password and display animation.

```
case idle:
    if(flag1 == 0)
    {
        // Display once
        cal_time=120;
        glcd_frame1_write(); //Displays initial animation
        flag1=1;
    }
break;
```



### 1.3. SOFTWARE AND CODE

- **setTimer** State:

Here UP and DOWN switches are used to set Timer. 4 Digit Timer is updated according to switch presses. MAX and MIN limits are 2 Minutes and 10 seconds respectively. RIGHT switch is used to arm bomb as illustrated in Fig a.

- **autoTimer** State:

In this state 4 Digit Timer starts decrementing till value reaches zero. illustrated in Fig a.

```
case autoTimer:  
    // In autoTimer Mode, evaluate time and display the timer bits  
    glcd_digit_write(bit_pos0,0);  
    glcd_digit_write(bit_pos1,1);  
    glcd_digit_write(bit_pos2,2);  
    glcd_digit_write(bit_pos3,3);  
    clear_section_glcd(2,0,78);  
    glcd_bomb_write();  
    // Switch to Bomb Explosion on Timeout  
    if(cal_time==0)  
    {  
        flag5=1;  
        flag4=0;  
    }  
    // Decrement Time  
    if(cal_time>0)  
    {  
        cal_time--;  
    }  
    eval_time();  
    break;
```

- **bombDiffused** State:

No input from the user, but based on switch presses, the password is checked. Sequence is detected. If correct, Bomb is diffused. as illustrated in Fig a.

- **bombExplode** State:

No input from the user, but based on timer value. As Timer value reaches zero bomb Explodes. Animation for the same is displayed as illustrated in Fig a.



### 1.3. SOFTWARE AND CODE

#### 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.
- `timedbom.c` is the main file. This contains the Statechart and the Timer Bomb abstraction.

#### 1.3.4 The Vending Machine Controller

##### Problem Statement

The overall objective is to design a vending machine controller. The system has five digital inputs and three digital outputs. You can simulate the system with five switches and three LEDs. The inputs are *quarter*, *dime*, *nickel*, *regular soda* and *diet soda*. The quarter input will go high, then go low when a 25¢ coin is added to the machine. The dime and nickel inputs work in a similar manner for the 10¢ and 5¢ coins. The sodas cost 35¢ each. The user presses the soda button to select a regular soda and the diet button to select a diet soda. The *GiveSoda* output will release a regular soda if pulsed high, then low. Similarly, the *GiveDiet* output will release a diet soda if pulsed high, then low. The *Change* output will release a 5¢ coin if pulsed high, then low.

### 1.3. SOFTWARE AND CODE

#### Statechart Solution

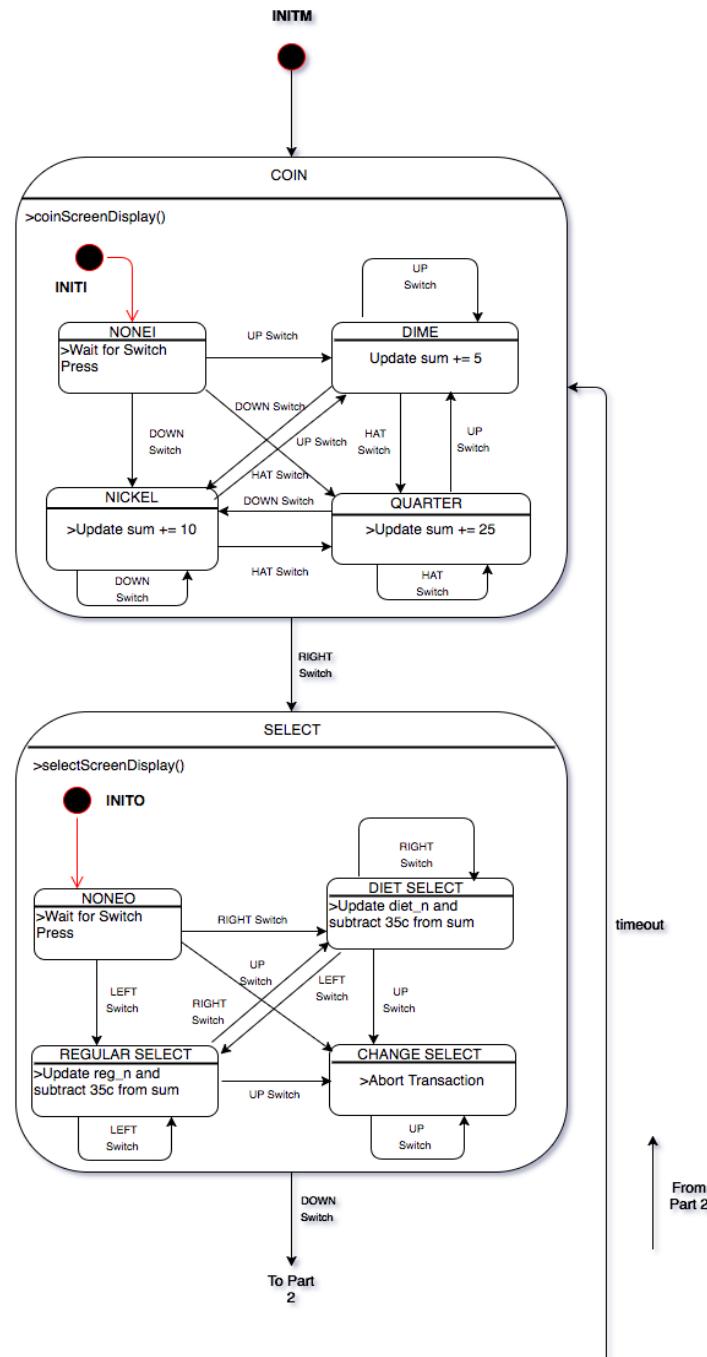


Fig a: Vending Machine Statechart Part 1



### 1.3. SOFTWARE AND CODE

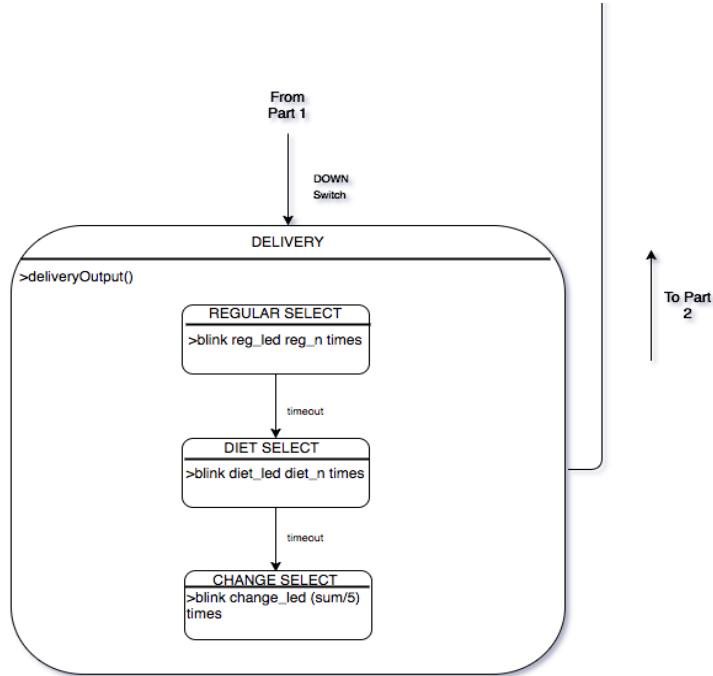


Fig b: Vending Machine Statechart Part 2

From the problem statement, the states of vending machine over which its behaviour changes considerably are taken to be individual states of the vending machine. In this case, the vending machine problem can be segregated into:

- **INITM State:**

Perform basic initialization of sum entered, number of regular and diet soda selected variables.

```
case INITM:  
    // Resets various variables.  
    sum = 0;  
    reg_n = 0;  
    diet_n = 0;  
    vm_mode = COIN;  
    input = INITI; // for COIN state  
    output = INITO; // for SELECT state
```

- **COIN State:**

Switches correspond to coin entry, and coin entry GUI is displayed. (Unique Input and Output Behaviour). Internal transitions corresponding to each coin entry is as illustrated in Fig a.



### 1.3. SOFTWARE AND CODE

- **SELECT** State:

Switches correspond to soda selection or cancel transaction, and Soda Selection GUI is displayed. It has internal state transitions corresponding to each selection as illustrated in Fig a.

- **DELIVERY** State:

No input from the user, but based on previous input, dispenses Regular Soda, Diet Soda and change through LED blinks and GLCD display. Internal transition between various states is as illustrated in Fig b.

## RTOS Implementation

The use of statechart greatly simplifies task scheduling in RTOS. Here, basically two tasks are run as follows:

- **readInput()** Task:

Handles switch press inputs. Has an internal state machine(implemented using the switch case construct) running as shown in the following code snippet:

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

- **displayOutput()** Task:

Handles GLCD display and LED blink outputs. Also has an internal state machine(implemented using the switch case construct) running as shown in the following code snippet:

```
switch (vm_mode){  
    case INITM:  
        // Resets various variables.  
        sum = 0;  
        reg_n = 0;
```



### 1.3. SOFTWARE AND CODE

```
    diet_n = 0;
    vm_mode = COIN;
    input = INITI;
    output = INITO;
    case COIN:
        coinScreenDisplay(); // Displays GUI for
        break;               // coin entry state
    case SELECT:
        selectScreenDisplay(); // Displays GUI
        break;               // for soda select state
    case DELIVERY:
        deliveryOutput(); // Handles Soda and
        break;               // coin dispensing
}
```

#### 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.

### 1.3.5 The Breakout Game

#### Game Design

The overall objective is to design the classic Breakout Game for the Tiva C Game Console, using the onboard peripherals. The breakout game has a rectangular paddle at the bottom of the screen off which a moving ball ricochets off. At the top of the screen, it consists of rows of bricks. The victory condition is to clear the row of bricks, by hitting it with the ball.

### 1.3. SOFTWARE AND CODE

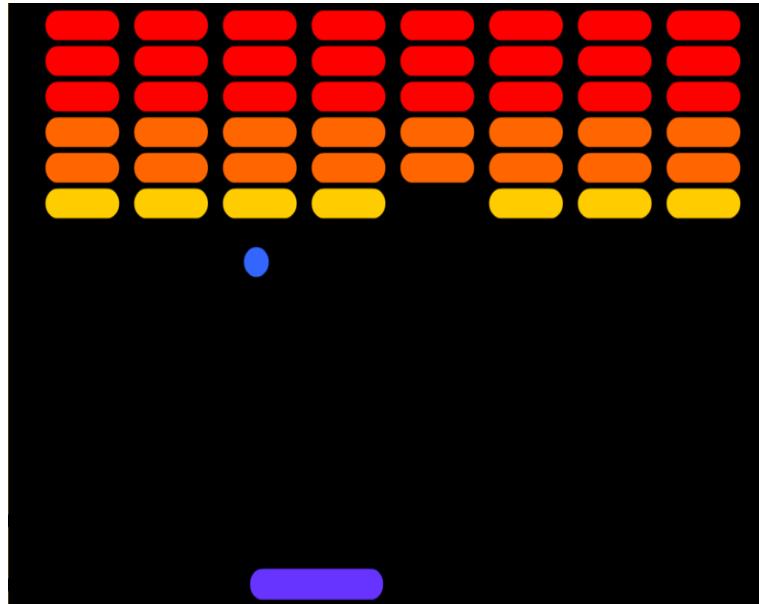


Fig a: The Breakout Game Concept

Our implementation of the game is to have 5 types of bricks:

- *HARD* brick - Takes 3 hits to disappear.
- *MEDIUM* brick - Takes 2 hits to disappear.
- *EASY* brick - Takes 1 hit to disappear.
- *MAGIC1* brick - Delivers a hit to all surrounding bricks.
- *MAGIC2* brick - Increases paddle size for 10 seconds.

There are 3 difficulty levels:

- *EASY* - 3 lives, higher proportion of EASY bricks.
- *MEDIUM* - 2 lives, higher proportion of MEDIUM bricks.
- *HARD* - 1 life, higher proportion of HARD bricks.

Implicit above is an implementation of a life system (allowed number of retries for the player). There are three speeds for the ball:

- *SLOW* - Ball moves slowly.
- *MEDIUM* - Ball moves at an average speed.
- *FAST* - Ball moves very fast.



### 1.3. SOFTWARE AND CODE

---

Additionally, the game has the following screens:

- Entry Cutscene
- Menu Screen
- Instructions Screen(Accessible from the menu)
- Settings Screen(Accessible from the menu)
- Gameplay Screen
- Victory Screen
- Game Over Screen

#### Statechart Solution

Since the game has several active components at the same time, it runs several parallel state machines each of which may or may not be independent of the other. The different state machines are as follows:

1. *Ball Direction* State Machine

### 1.3. SOFTWARE AND CODE

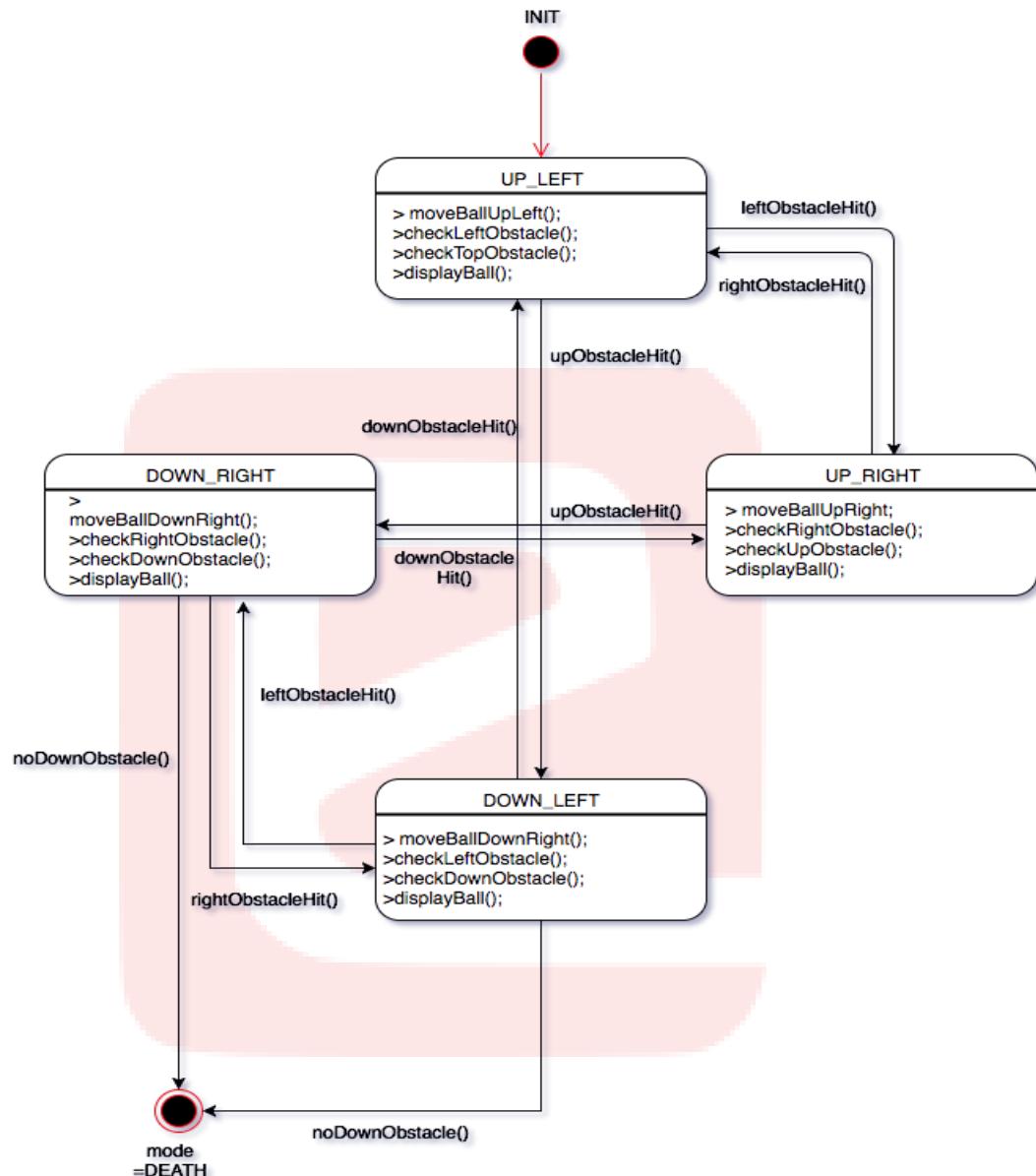


Fig b: Ball Direction State Machine

### 2. Brick Type State Machine

### 1.3. SOFTWARE AND CODE

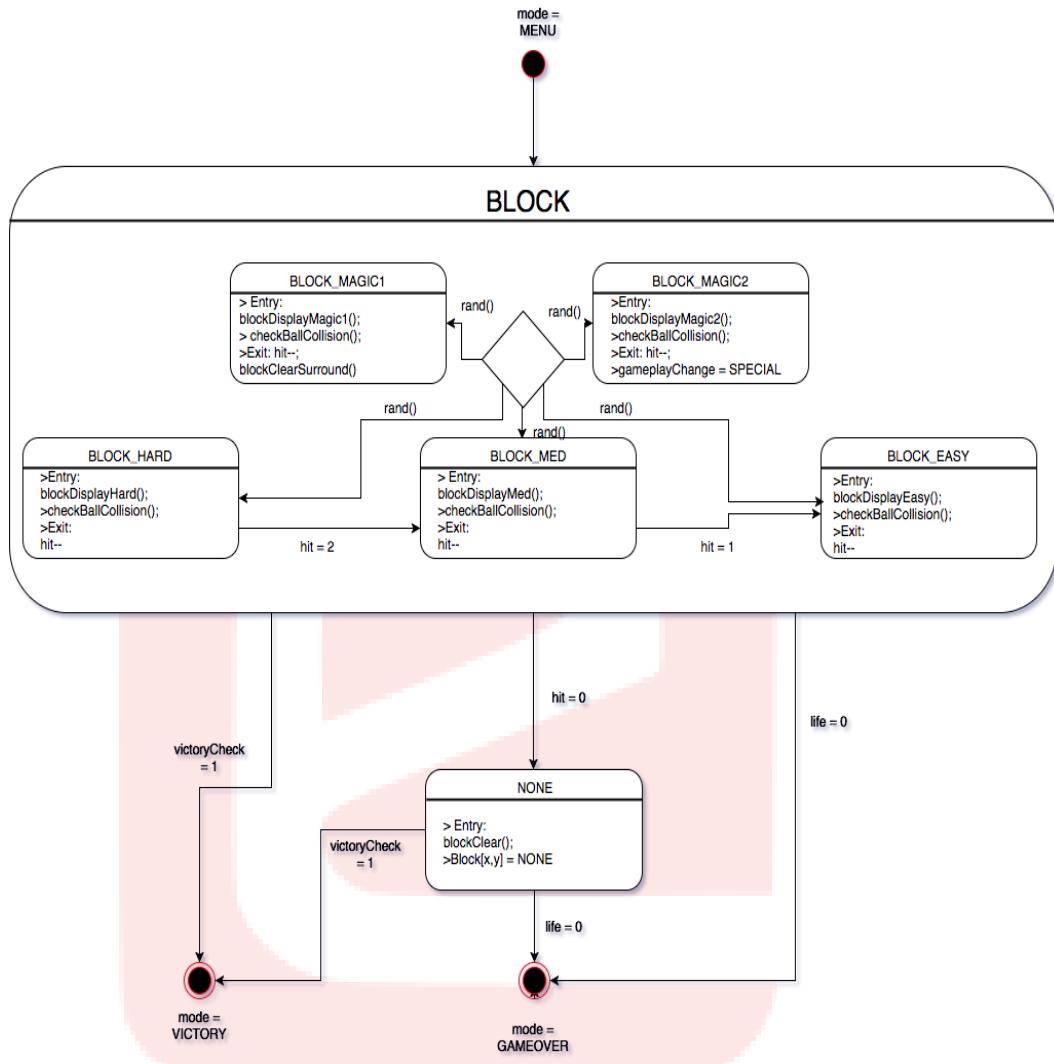


Fig b: Ball Direction State Machine

## 1.4. USE AND DEMO

### 1.4 Use and Demo

#### 1.4.1 Images

Timer Bomb

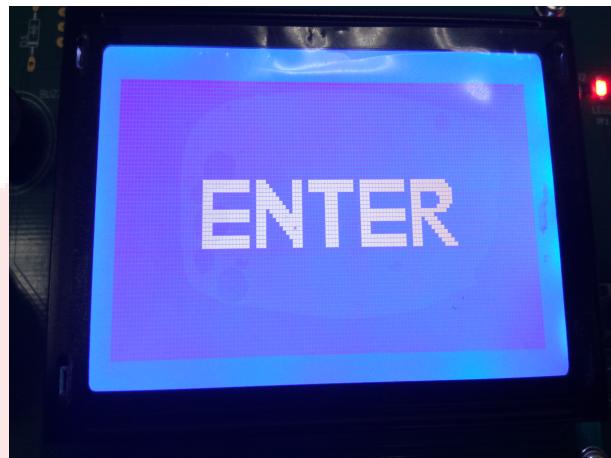


Fig 1: Idle State



Fig 1: setTimer State

#### 1.4. USE AND DEMO



Fig 1: autoTimer State

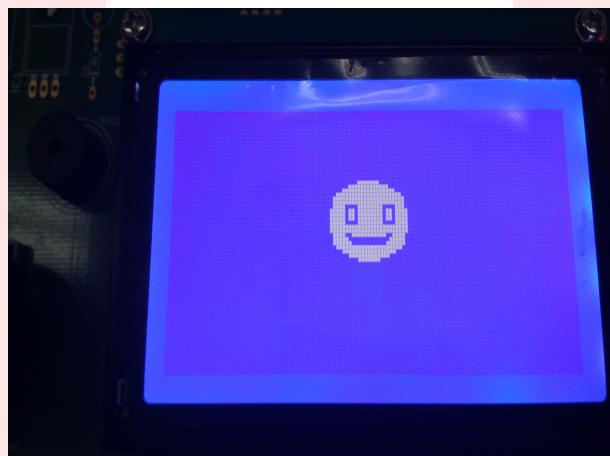


Fig 1: bombDiffused State



Fig 1: bombExplode State

#### 1.4. USE AND DEMO

Vending machine



Fig 1: COIN State

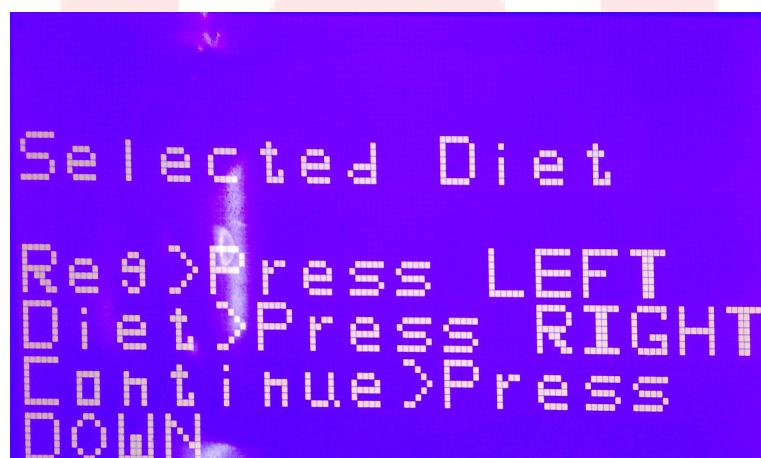


Fig 1: SODASELECT State





## 1.5. FUTURE WORK

Fig 1: DELIVERY State

User Instruction for demonstration

[Youtube Link](#) of demonstration video

## 1.5 Future Work

What can be done to take this work ahead in future as projects.

## 1.6 Bug report and Challenges

Any issues in code and hardware.

Any failure or challenges faced during project

# Bibliography

- [1] Ad Kamerman and Leo Monteban, *WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed band*, 1997.