

SHARC BUOY State Machine Design

Author Jamie Jacobson  
Student Number: JCBJAM007  
Date: 22 May 2020

### The following document outlines a design for a high-level state machine to be implemented on the STM32L476RG microcontroller

1. Introduction

In a multi-sensing system, it is important to manage the interactions and data flow between the various aspects of the system to ensure the device operates in a predictable, manage manor. To achieve this, a state machine can be implemented to provide a high-level form of control over the system. This can be achieved by decomposing the overall function of the buoy into a series of finite states. These states are connected through a series of transitions which can be described using Boolean techniques. Through this, the buoy retains a modular structure both in firmware and in hardware which can allow for additionally sensors and functions to be implemented seamlessly. This document outlines the State Machine design process and shows the steps taken to designing the machine.

# Design Methodology

The state machine design will be broken down into two main parts:

1. The main Loop
2. Asynchronous Behavior

For each part, the key functionality will be discussed and from there, states will be identified. For Asynchronous behavior, exit and entry conditions will be outlined, and a brief discussion will be given on the practical application. Finally, the state machine will be implemented on the STM32l476RG micro controller using Atollic TrueStudio™ Version 9.0.0 with CubeMX and HAL Library Driver for hardware interfacing/access.

1. Main Loop

# State Detection

The goal of the buoy is to sample environmental, gps and power data at a fixed rate. This rate will be used to describe the period between sampling the devices. Each Sample will be condensed into a byte packet and stored in flash memory at a sector. After every 4 samples, the device will load the packets from memory into a buffer and transmit the data. When the device exits this state, it will reset the sample count and repeat until the buoy is turned off or dies.

The primary loop can therefore be broken down into 4 main states:

1. **Reset State:** The device initializes the counter and verifies the sensors.
2. **Sample State**: During this state, the device actively receives data from the sensors and stores them into a packet which is then saved to Memory.
3. **Sleep State**: The device enters this state between samples and active states. Here, the device will remain in this state for a time . After which, the buoy will wake up
4. **Transmit State:** The device will load the data from memory and transfer to the Iridium Modem Buffer. Upon successful transmission, it will enter the Reset state

Each state will control which routines are performed during the function. A typical run is given by the diagram below:

The inputs to the state machine are:

1. C: a 2-bit integer signifying the number of samples performed (0 =< N <4)
2. T: Variable that matters when the system is asleep. Signifies whether the system has slept for the required Tsample

The system has no explicit outputs however, the state machine is used to control which routines will be executed during the execution phase of the program. Therefore, the outputs can be considered as the Routine Rx as shown below:

Rsample -> Sensor sample routine, this can involve all the sensors or just a select number. For simplicities sake, this period implies all sensors will be sampled from

Rsleep-> Device is in a sleep state and will wake up when the periodic wake up unit counts to Twake

RTransmit -> Satellite Transmission Routine

Given the following information, we derive our ASM chart and PS/NS diagram

# Present State Next State Diagram

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Inputs** | | | **Present State** | | **Next State** | | **Outputs** | | |
| **C1** | **C0** | **T** | **Q1** | **Q0** | **D1** | **D0** | **Rsleep** | **Rtx** | **Rsample** |
| x | x | x | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | x | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | x | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | x | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | x | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| x | x | x | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| x | x | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| x | x | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

## 

# Present State Next State Diagram

Figure 1: Main Loop ASM Chart

Note: Twake emits A High pulse when the system has Slept for a length of time T as defined by the user. In addition, the sample counter gets reset after every transmission state and when the buoy enters a reset state. The number of samples before transmission is chosen to be 4 to optimize packet size for the transmission buffer. Since the Iridium Buffer is 340 Bytes long and the Transmission rate is per 50 bytes, the goal is to transmit as much data that would fit into the buffer as possible. Too frequent transmissions incur a high data cost but result in data integrity. Too few transmissions can result in lost sample points if a transmission is not received.

1. Asynchronous Behavior

# Asynchronous Functionality Design

Asynchronous behavior describes all functionality that occurs outside of the main loop. This can come from Interrupts/ External events which causes the system to exit the main loop regardless of state and execute the code. This can occur from the following sources:

1. Interrupts on the wake-up pins for Iridium message reception (Ring alerts) and IMU event detection (collisions / freefall)
2. Events such as:
   1. Low power detection
   2. Brown Out Detection
   3. Software Resets
   4. Watch Dog Resets

These events take precedence over the main loop function. The table below shows the entry/exit conditions. Functionality as well as return state after exit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Type** | **Entry Condition** | **Function** | **Exit Condition** | **Return State** |
| Ring Alert | Interrupt | Buoy In any state other than reset with GPIO mapped to EXTI, with wake up from sleep mode. The WUP Pin receives a Digital High from Ring Indicator Pin on Iridium | The user has transmitted a packet to the buoy. Download the packet and execute/store the data based on the packet structure | Device has downloaded user data which has been used to update the system and store data. | If entry source was a wake up, device will return to sleep. Otherwise device will return to the main loop. |
| Event Detection | Interrupt | Buoy In any state other than reset with GPIO wake up pin mapped to EXTI, with wake up from sleep mode. The WUP Pin receives a Digital High from Interrupt pin | Device reads the interrupt source from the IMU, initializes I2C peripheral and begins sampling IMU data. Interrupt source determines the sampling rate, period and mode | Device will exit when the IMU has finished sampling and the data has been stored into memory | If entry source was a wake up, device will return to sleep. Otherwise device will return to the main loop. |
| Brown Out Detection | Event | Buoy is in run mode or in Standby mode with Brown out detection voltage enabled. Vbrownout has been configured in option bytes. Event occurs when the voltage supplied to the microcontroller is less than Vbrownout causing the device to be held under reset. When the Voltage rises above the threshold, the device will enter the handler | Device resets the relevant flags and checks for data corruption. IF no data is corrupted. Device will reload the last state and attempt to run it again. Otherwise the device performs a software reset | Vsupply > Vbrownout, device successfully executes code in handler | Returns to main loop |
| Low Power Detection | Event | Device is in run or sleep, Power Voltage thresholds set in PWR and interrupt enabled. Event occurs when Vsupply < Vpower generating an event interrupt. | Device will read INA sensor and transmit final packet to base. All peripherals switched off; Device placed into shut down mode | No Exit | No Return State |
| Software Reset | Event | The NRST internal line is pulled low for a few seconds. This is triggered in any state by triggering a software reset in the NVIC | Reset the buoy to an initial state. Clear any pending flags. Reset data in back up registers | Successful reset of voltage domains | Return to Reset state and start of main loop |

Thus, we arrive at the following diagram for the asynchronous behavior

# A picture containing room Description automatically generatedAsynchronous Behavior Diagram

1. Implementation

The State Machine was developed using STM32 HAL Libraries and Atollic TrueStudio™ V9.0.0. The development was split into 3 phases as shown below,

Figure 2: see Document Power Mode and Clock Selection

The project was set up using CUBEMX for creating peripheral initialization and handling functions. Final code for the project can be found in the folder BUOY\_Frame\_L4. All the tools, definitions and functions developed for the Buoy frame have been organised into the library files Sharc\_Frame.h and Sharc\_Frame.c. This allows for the frame to ported over multiple projects allowing for a new firmware version to be developed from scratch instantly. The following section describes key features and functionality.

# Program Structure

The project code files are organised into the following folders:

1. Drivers
2. SRC
3. Start Up

The Drivers folder contains the HAL and CMSIS libraries for the device. The SRC Folder contains the main.c file which acts as the entry point for the program to run. The start up file contains assembly code that specifies the vector table, Hard fault/ Reset Handler Entry Points as well as the entry point for the main code. When the file startup\_stm32l476xx.s is run, the program enters into the main() function and begins running from there. The SRC folder contains the .h/.c pair Sharc\_Frame files which are implemented in the main.c

The main() code consists of a set up phase and a loop phase. During the set-up, the functions HAL\_Init(); and SystemClock\_Config(); are used to reset the peripherals and the systick timer and set the System clock to the correct source and speed. These two functions run in the set-up phase of the code and are called whenever the program re-enters the main function. The next step in the set up phase is to configure the unused GPIO pins to analog floating mode. This greatly reduces the current consumption by the micro controller. The peripherals required for debugging the code are placed here. Before deployment, the code will be removed. This phase is referred to in the program as the System Init and Clock Configuration. It is the first phase to be run.

The next phase in the Set-up is the Power and Reset State Check. If any power event occurs, a software reset is generated, and the program will restart from the main() function. When this happens, a flag is set in the RCC CSR. This can occur in the form of a brown out, Pin reset or Low Power event. This phase will check for the occurrence of any event and handle them before the program enters the main loop. Finally, if successful the program will enter the main loop and start the state machine.

# State Implementations

States are implemented as enumerations in the programming. The structure Buoy\_State\_typedef; contains an enumerated list of buoy states. Each state label is represented by a unique unsigned 8-bit integer. The table below shows the current states, the assigned label and their value.

|  |  |  |
| --- | --- | --- |
| State Name | Label | Uint8\_t value |
| Reset | STATE\_RESET | 0b01 |
| Sample | STATE\_SAMPLE | 0b10 |
| Sleep | STATE\_SLEEP | 0b11 |
| Transmit | STATE\_Transmit | 0b100 |
| Asynchronous Event | STATE\_ASYNCNT | 0b101 |

The device also contains a hidden state. This occurs when the state is set to 0b0. This occurs when the system back domain has been reset forcing the storage location of the state to reset the data. This state is the power on state and is used to transition into the main loop by reinitialising the device. The entry point into the main loop is the reset stat

# State Variable

The State Variable holds the value of the current state of the buoy. This variable is stored in two locations: When the system is in run mode, the value is stored in the global variable Current\_State. When the device is in a deep sleep state, the variable is stored in the RTC Back up registers at byte 0 of Back-Up Register 0. Upon wake up, the value is loaded from the register and placed in the global variable.

# State Transitions

The main loop follows a sequential state transition as described in Figure 1. To achieve this, at the start of each loop, the program reads the value stored in the state variable. This determines what the previous state was. Based on this value, the new state is determined and stored in the state variable. This process is described using the following flow chart:

A screenshot of a cell phone

Description automatically generated

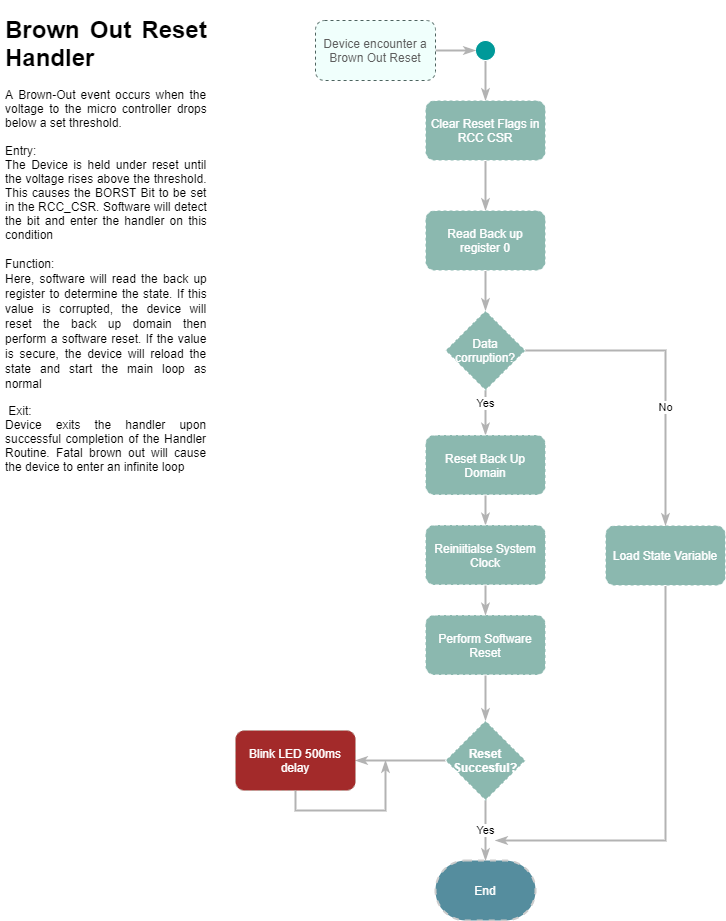
Figure 3: Flow chart for the state-check algorithm

Figure 3 above shows the algorithm for selecting and transitioning between states. This algorithm allows for states to be linked in any order and, most importantly, Separates the state selection from the state function. By separating these two concepts, a more modular framework is created. This allows for the addition of more states and transitions without modifying the routines that are currently in place. This allows for device functions to be turned on and off as desired without drastic changes to the firmware.

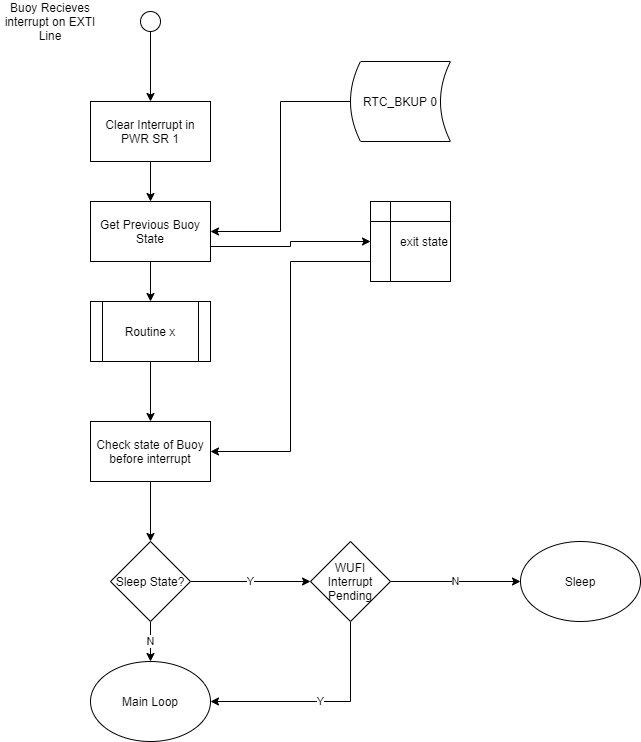
# Asynchronous Events

Asynchronous States are states that occur through interrupts / events. These states take a higher precedence over the main loop states and therefore are checked before the state check shown above. The order of precedence is shown below

Power Events generate a system reset and raise a flag in the PWR Status Register. When the flag is set, the program enters the handler and, if the event is non-fatal, returns to the main loop. The following flow chart shows an example of such a case for a Brown Out Event



Some sensors have interrupt pins and can be configured to trigger upon detection of a specific event. When this happens, the sensor will send a digital high on the interrupt pin. As mentioned in section 3, by connecting these pins to external wake up pins, the buoy is capable of event detection in deep sleep mode. If an event is detected while in deep sleep, the interrupt causes the buoy to wake up and resume from the beginning. Note: no interrupt handler is entered at this point. A flag is set the PWR\_SR at the position of the wake-up pin it detected. The buoy will enter the asynchronous state depending on which flag is set and will execute the routine associated with it. When the buoy wakes up from an internal wake up timer, the pins are reconfigured into GPIO EXTI mode which allows the buoy to receive interrupts when active. Note: by keeping the buoys configured as wake up pins, the system will reset when an interrupt is detected.



1. Recommendations

Currently the device operates on a synchronized time basis. The value c is arbitrary and is chosen to ensure that sample periods are evenly spaced apart with a transmission period being a perfect multiple of the Sampling period. This dependency can be removed by triggering the Transmission state when the data buffer reaches a certain threshold. Therefore, the device can reduce the number of transmissions and increase the amount of data per each delivery. It may occur that the data packets may not fit nicely into the full 340 bytes. Therefore, additional data processing is required. Finally, some devices will not be operated in transmission mode but will rather log the data to a secondary, high capacity storage. For this purpose, it is necessary to consider a log state. This will replace the transmission state and result in data being stored in a clear format on this storage. This will require additional driver code to be written depending on the device and the extra consideration of peripheral space and power consumption

1. Conclusion

Therefore, a Buoy state machine has been designed with additional information on asynchronous behavior. Each state will be used to activated sensors/ routines based on previous states according to the PS/NS diagram and ASM chart as shown above. Finally, sources of asynchronous behavior are described in the table above and the actions/return states are shown additionally to ensure that the main buoy firmware is easy to follow, and the behavior of the code is predictable.