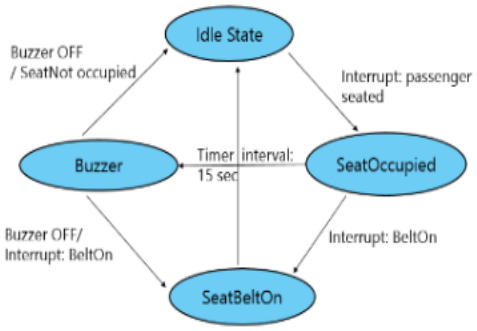
Introduction

FSMs are the main design approach in designing embedded systems. The finite state machine is made up of multiple states. At any point of time, the system is in one state and an event/interrupt certain actions changes the state of the system. The FSMs can be modeled in embedded systems using switch condition or if-else-if constructs. Consider an example model of Seat Belt Controller module in automobile. The module can be modeled using FSM concepts. The state transition diagram of model is shown below:



As per FSM model system can be in one state at any point of time. The four states are:

1. Idle state
2. Seat Occupied
3. Seat Belt on
4. Buzzer

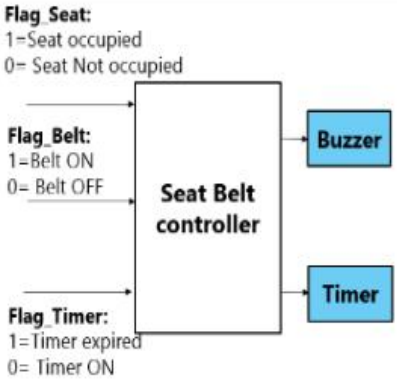
The above diagram shows the transition from one state to another state. The initially system will be in idle state and waiting for interrupt.

System Design Specifications

System Inputs interface:

Switch S1: Interrupt on this S1 set flag_seat flag (Seat occupied)

Switch S2: Interrupt on this S2 set flag_belt flag (Seat belt on)



System outputs Interface:

Buzzer Pin (P1.2): Generate PWM pulse of constant duty cycle (say 50%) on this pin

LED1: Notify driver about seat belt status.

FSM States description:

1. Idle state
    a. The module is waiting for interrupt from switch S1
    b. The system also monitors the seatbelt on flag, If Flag is set it notifies through LED1 state HIGH.
2. Seat Occupied
    a. When interrupt comes from switch s1
    b. In Seat Occupied state, System waits for Switch S2 interrupt as well as timer out signal. If interrupt from Switch S2 will not come till 15 seconds, the system goes to Buzzer state.
3. Buzzer state
    a. In buzzer state, the system generates PWM pulses on controller pin, where buzzer is connected to notify passenger about the Seat belt reminder.
    b. On arrival of Switch S2 interrupt, the system will return to Seat Belt on state.
4. Seat Belt on State

a. There is only one transition possible. The system returns to idle state after checking seat belt on flag and indicating LED1 state High.

Expected outcomes

O1: Able to write basic code for various CPU peripherals (Timers, GPIO and PWM)

O2: Understand various configuration registers and peripheral operations

O3: Able to Analyze application features / requirements

O4: Able to relate feature implementation with CPU peripherals

O5: Design algorithm and pseudo code

O6: Design Embedded C code based upon algorithm

O7: Able to design using modular approach

Task/Activities

Activity-I: Modular architecture and C code design

- Choose relevant software architecture and design modular C code for hardware platform.
- Ensure Usage of macros, enums, bit-fields, structures, function pointers to design code.
- Ensure correct usage of datatypes, Qualifiers like volatile, const, extern, static.

Activity-II: Circuit simulation

Design Test circuit to validate the functionality

Activity-III: Code quality and related language standards

Develop the code as per coding guidelines and related language standards.

Guidelines & Assessment Schema

Total time duration - open book (18 hours)

Assignment submission packages

- Module Implementation files [.c files] and corresponding header files [.h files]
- Main program [.c file]

- .HEX file
- Simulation circuit [.simu file]

Resources and tools required

Tools required	Description	Download link
Code Block	IDE used to develop C/C++ based embedded applications	http://www.codeblocks.org/downloads/binaries
WinAVR	Open source gcc based tool chain for AVR micro-controllers	https://sourceforge.net/projects/winavr/
Simulide	Used for Circuit designing, uploading hex file and simulating the application	https://sourceforge.net/projects/simulide/

Activity based assessment rubrics and weightage

Activity based outcome mapping with assessment weightage								
Activity	Weightage	Outcome mapping chart						
		O1	O2	O3	O4	O5	O6	O7
Activity-I	60%	×	×		×		×	×
Activity-II	20%	×			×			
Activity-III	20%		×					

Assessment rubrics for various activities	Rating Scale (0-10)
Rubrics for Activity-I	
Modular Programming and related guidelines	
Proper usage of variable naming conventions	
Relevant use of various storage classes	
Proper usage of datatypes, structs, enum, pointers, typedefs	
Overall Software Design and Interrupt usage	
Rubrics for Activity-II	

Overall functionality	
Ensure Proper test cases as per input - output requirements	
Rubrics for Activity-III	
Code optimizations	
Coding guidelines	

Additional note

- Ensure the code works for the stated requirements.
- You are not supposed to switch case conditions.
- Use Timers to create PWM as well as for creating delay.
- Ensure basic test cases are fulfilled.
    - Ex: System must notify the passenger by making LED1 pin HIGH indicating seat belt on state.
- Comply the code with **code style guidelines.**
- Develop an embedded C code for the above scenarios.
- Proper use of interrupt needs to be emphasized.