Queen Mary
University of London

EMBEDDED SYSTEMS

Project Lab A

“Reversing Alarm”

Lecturer: William Marsh

Surname	Cruz Felix	**First name**	Frank
Student number	150684871		
Project Letter	A	**Project title**	Reversing Alarm
GIT Hub Link	https://github.research.its.qmul.ac.uk/ECS642-714-EmbeddedSystems/bt15585ProjectA.git		

December 12th, 2018

1 Information about your System Requirements

The aim of this project is to implement a simple reversing alarm in which an IR emitter and IR detector are used to detect light reflected from an Object and an audible tone is generated when detecting an object. The system has achieved the following requirements:

Given Requirement	**Status**	**Detailed Behaviour**
A button should be used to arm the system and then disarm it. In a car, this is done when the reverse gear is engaged.	Fully Implemented	The software developed allows users to interact with an external button as this is performing same as when the reverse gear is engaged. The requirement was achieved by configuring a pin in port D (PTD6) as input to detect the event when the button is pressed. This event is handled by a thread (t_button).
When it is engaged, the system should give a brief audible confirmation that it is working.	Fully Implemented	When the button is pressed, an audible confirmation is played. This requirement was achieved by adding and setting a speaker. The pins used in the speaker are port A pin 2 (used for the audio signal- PTA2), port A pin 4 (used to control the volume - PTA4), a pin for the voltage supply (supplies 5v positive - P5V_USB) and a pin for the GND (GND). A function called reverseAlarmOnOff produces the audio tone.
Object is detected	Fully Implemented	The detection of the object is made by implementing a sensor, which uses an IR emitter (LED) and Detector (Phototransistor). Both types of IR devices point in the same direction to determine if an object is reflecting IR energy. This sensor uses the following pins: for voltage supplier (Vcc 3.3 volt), for ground (GND 0 v), for an analog input (to MCU – PTB0), for digital output (from MCU – PTD7).
An intermittent audible tone is generated and becomes increasingly noticeable as the distance decreases.	Fully Implemented	If an object is detected, an intermittent audible tone is generated and becomes increasingly noticeable as the object gets closer. As the object comes closer, voltage decreases and in turn the program plays a tone corresponding to this.
Frequency, the on/off times of the intermittent tone and volume were changed.	Fully Implemented	The frequency, the time_on/time_off periods and volume are all changed with respect to the measured voltage. Voltage ranges have been created in order to set the variable volt [0,11], which is used as the index value for the "delays" array. The time on/off period decreases by using the built function osDelay(), depending on the distance between the sensor and the Object. The volume of the intermittent tone increases as the distance decreases.

2 Design System

2.1 Overview

The system uses the RTOS (Real-Time Operating Systems), which allows multiple threads to execute at the same time. This makes use of the ISR (Interrupt Service Routine) or Interrupt handler.

The software developed is a “Reverse alarm”, in which the intermittent audio tone becomes increasingly noticeable as the distance between the sensor and object is closer.

2.1 Peripherals and Pins

Speaker:

Name	KL25Z Pin Label	Purpose
PWM	PTA4	Volume control using PWM
Audio	PTA2	Audio Signal
+ve	P5V_USB	5v positive supply
GND	GND – common ground – j2 pin 14	0V

Infrared:

Name	Colour	KL25Z Pin Label	Purpose
+ve	Red	Vcc 3.3 volt – J9, 4	3.3 positive supply
Analog Input	White	PTB0	Measure IR Brightness (ADC readings)
GND	Black	GND - common ground j2 pin 14	0v
Digital output	Blue	PTD7	Switch on IR emitter (Enable/Disable)

Button

Name	KL25Z PIN label	Purpose
GND	GND – common ground j2 pin 14	0v
GPIO pin	PTD6	Configuring pin as input

2.2 RTOS

This system uses RTOS approach with the use of threads, event flags and delay functions. When the button is pressed to either enable or disable the system, an event is created.

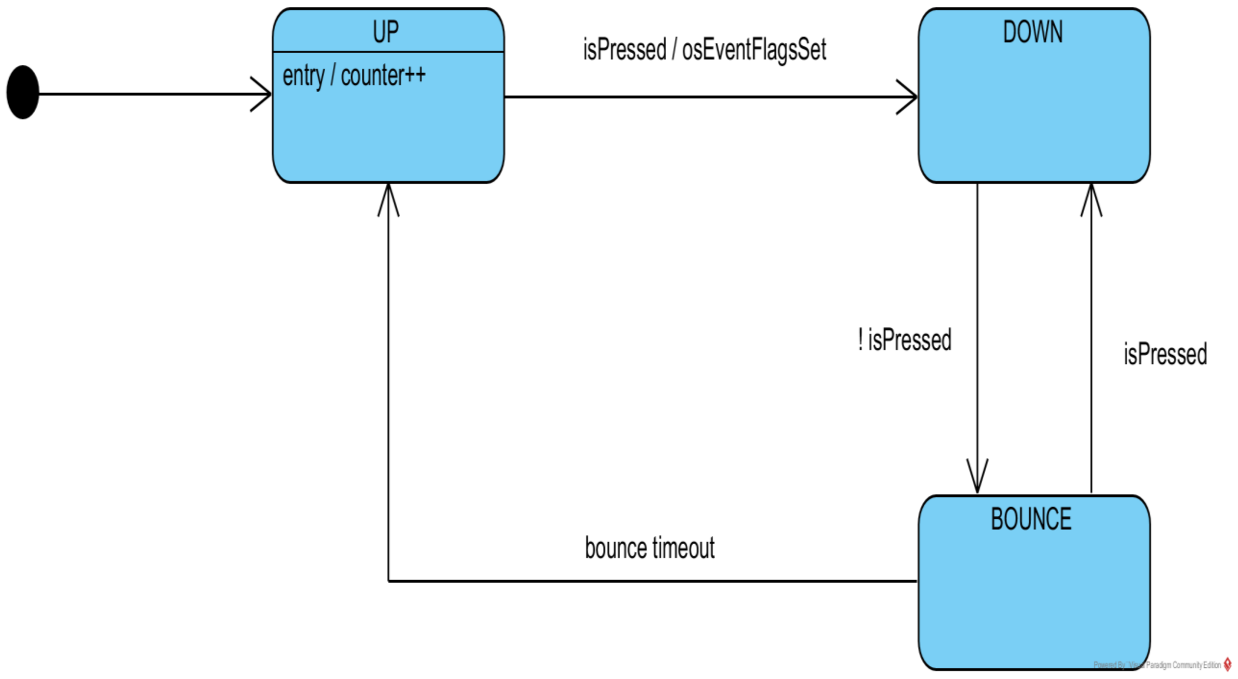
Once the button has been pressed, the other thread (tone thread) waiting will carry out the process. The tone thread plays the initial tone, the beeps function is then called. The voltage is repeatedly measured from the ADC and the tone is played according to the values read. The osDelay function is used to change the time on/off.

2.3 ISRs (Interrupt Service Routine)

The PIT.c contains the code for the API functions and the ISR. There is a single ISR for both channels, which looks at and resets the flag that shows a counter has run down to zero.

2.4 Task 1: Button Thread (t_button)

The first task detects button press events. This thread waits for the button to be pressed so that an event flag is set. The button is initially up (BUTTONUP). When the button is pressed, it generates an OS event flag event, the state of the button is set to DOWN and the counter is incremented. The system will go to the DOWN state and then to the BOUNCE one. Aslong as the key is still pressed, it will remain in the BOUNCE state passing over the DOWN one. The status of the system (enabled/disabled) is kept in the variable counter. If this variable is an even number then, the system is disabled, otherwise, the system is enabled. This functions with the use of a modulus operator (%) that returns the remainder. The state diagram for this task is as follows:



2.4 Task 2: Tone played (t_tone)

After the button is pressed, the thread carries out the process. The sensor is switched on and an initial audible tone is played. Then, the voltage is repeatedly measured in order to detect if there is an object in one of the ranges created. If the object gets closer to the infrared sensor, the audible tone becomes increasingly noticeable. The startBeeps function calls the following functions in this order to emit the beeps:

setPWMDuty(volume[volt]) --> sets and changes the volume according to the value stored in the volume array.

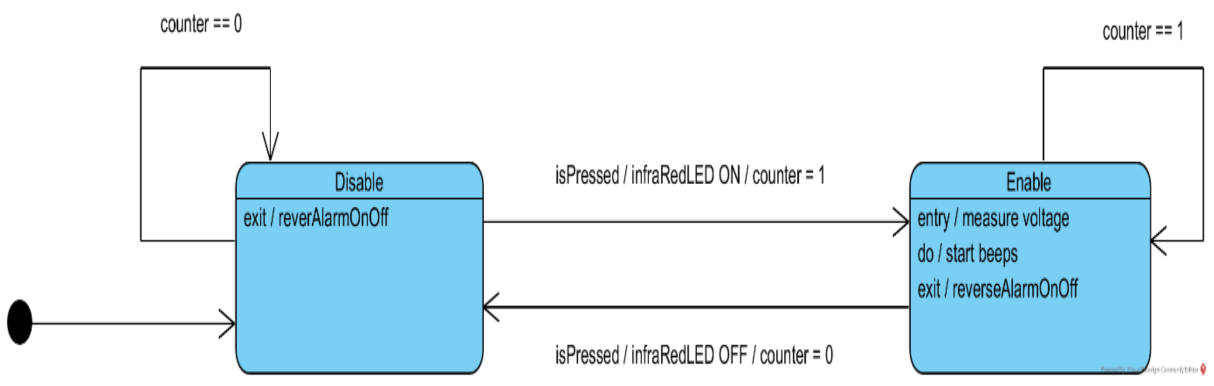
setTimer(0, array[0]) --> set the timer for a new countdown value and this is set without stopping the timer, in this case, it is used the first element of the array named array.

startTimer(0) --> start the timer on channel zero

voltageRange() --> It matches the voltage read with one the ranges create, it sets an integer value named volt.

stopTimer(0)--> stops the timer on channel zero

osDelay(delays[volt]) --> changes the on/off times



2.5 Voltage Range (voltageRange())

The voltageRange function is repeatedly called as this is the function matches the twelve different voltage ranges and sets the value for the integer volt. As it can be seen, the twelve different ranges have been divided as follows.

Range	Volt value
voltage > 3.2702	0
voltage > 3.1549 && voltage < 3.27	1
voltage>3.1407 && voltage<3.1548	2
voltage >3.1304 && voltage < 3.1406	3
voltage>3.1102 && voltage<3.1303	4
voltage >3.0904 && voltage < 3.1101	5

voltage>3.0887 && voltage<3.0903	6
voltage >3.0687 && voltage < 3.0886	7
voltage>2.9087 && voltage<3.0686	8
voltage >2.7140 && voltage < 2.9086	9
voltage>2.1899 && voltage<2.7139	10
voltage >0.20 && voltage < 2.1898	11