Table of Contents

1. Introduction

The objectives of the project are as follows:

- Interpretation of the necessities of complex task and encapsulation into sub-task
- Fulfillment of co-operation of utility modules
- Understanding a given complex hardware and compatibility of its components
- Writing a multi-task software for a given complex set up
- Introduction of the serial communication on TM4C123G and utilization of the facility on SPI protocol

2. Project Definition

In this project individuals are expected to build a car park sensor system that cooperates with driving safety components for a better driving experience. This useful system is found in almost all modern cars with varying degrees of functionality. In some cases, it is sold as a separate upgrade module to bring park sensor functionality to cars that do not have it from factory.

The system has three major functions:

2.1 Ultrasonic Sensing

There can be multiple ultrasonic sensors located on the periphery to provide information from all sides of the car. The sensors will continuously measure the distance to the obstacle (sidewalk or the front bumper of the car behind) and the system will act based on the distance. If it is closer than a user specified threshold, the car will break, and the user will be visually informed about the situation.

2.2 Preventative Braking

The car needs to stop if an obstacle is too close. The brakes can be engaged immediately when the threshold is exceeded, or they can be engaged proportionally as the distance gets smaller, provided that the car fully stops when the limit is exceeded.

2.3 User Interface

The user needs to set the threshold of what is considered a safe distance from the obstacle. The user should also be able to remove the braking condition at will and continue driving the car. An information screen will show the user the threshold distance, the currently located obstacle, and the state of preventative brakes (whether engaged or not).

3. Hardware

The park sensor system uses the following components:

1. NOKIA 5110 LCD Screen
2. 1 Potentiometer
3. 2 Buttons Placed on the TM4C123G Board
4. HC-SR04 Ultrasonic Sensor
5. Stepper Motor

4. Algorithm

4.1 Initializations and Purposes

INIT_SW:	Onboard switch buttons which correspond to pins PF4 and PF0 are initialized as inputs. A GPIO Interrupt is defined to detect the push on these buttons.
PORTA_INIT:	PA5 – PA2 are defined to be SSI ports. Data transfer between screen and board is enabled via these pins.
PORTB4_INIT:	PB5 is defined to be output and PB4 is defined to be input and it uses alternate function on Timer1. These pins are used to measure the distance with ultrasonic sensor.
PORTC_INIT:	PC7 and PC6 are defined as outputs. Screen’s reset and DC pins are connected to these pins.
PORTD_INIT:	Pins PD0 – PD3 are defined as outputs. Step motor is driven with these ports.
DELAY100:	A subroutine to cause a delay of 100ms by creating a loop.
INIT_SSI0:	SSI0 module defined on port A is initialized. Clock rate, mode and bit amount is selected.
INIT_TIMER_1A:	Timer1A is initialized. It is selected to be 16-bit countdown edge time capture mode. Distance measurements are done using these captured values.
INIT_TIMER_0A:	Timer0A is initialized. It is selected to be 16-bit countdown mode with timeout interrupt. Step motor is driven with these interrupts. A pre-

scale value of 3 is selected to make successive motor step intervals longer than or equal to 5ms.

INIT_SCREEN: Screen is initialized by giving the necessary commands. For this operation followings are selected:

- Horizontal mode
- V_{OP}=0xB6
- Temperature Control = 0x05
- Voltage Bias = 0x13
- Basic Command Mode
- Normal Display Mode

Moreover, this subroutine also clears the screen by calling the subroutine CLEAR_SCREEN to make sure that no past displaying is left on screen. Then, FIRST_DISPLAY subroutine is called to display items that will not change during the whole operation such as “Meas: mm” or “Thre: mm”

INIT_PORTE3_ADC: Pin PE3 is used to process analog data and obtain information on potentiometer. ADC utility is also initialized under this subroutine.

4.2 Distance Measurement

As mentioned above, distance measurement is done with the help of Timer1A and pins PB5 and PB4. By using the ultrasonic sensor’s utilities an echo signal is created and pulse width is measured. Using this measurement distance is calculated. During the calculation process different subroutines are used.

MEAS_2 subroutine calculates the distance from obtained pulse width by using the following relation:

$$Distance\ (mm) = Pulse\ Width(nsec) * 17 * 10^{-5}$$

After making this calculation, 2 separate subroutines are called which are UPDATE_MEAS and UPDATE_BAR.

UPDATE_MEAS subroutine writes the calculated distance value on screen. The area where this value written is specified.

UPDATE_BAR subroutine takes the measured value and pre-defined threshold value. Then these values are divided by 100 and stored in a specific place in memory to make them accessible for other subroutines. After this point, UPDATE_SPEED subroutine is called to adjust the step motor’s speed based on the distance between threshold value and measured distance.

UPDATE_SPEED subroutine loads the measured value and threshold value from memory(the location specified in UPDATE_BAR) into registers. After comparing these values, rotation speed of the motor is increased or decreased proportionally. Rate of increase or decrease depends on 2 parameters which are measured distance and threshold value. The algorithm to determine the rotation speed is given below:

$$TH = \frac{Determined\ Threshold\ Value}{100}$$

$$ME = \frac{Measred\ Value}{100}$$

$$SS = Slowing\ Scale = 9 - TH$$

$$DD = Distance\ Difference = ME - TH$$

$$SF = Slowing\ Factor = SS - DD$$

$$LU = Limit\ Up(Shoertest\ Timeout\ Duration) = 0x4E20$$

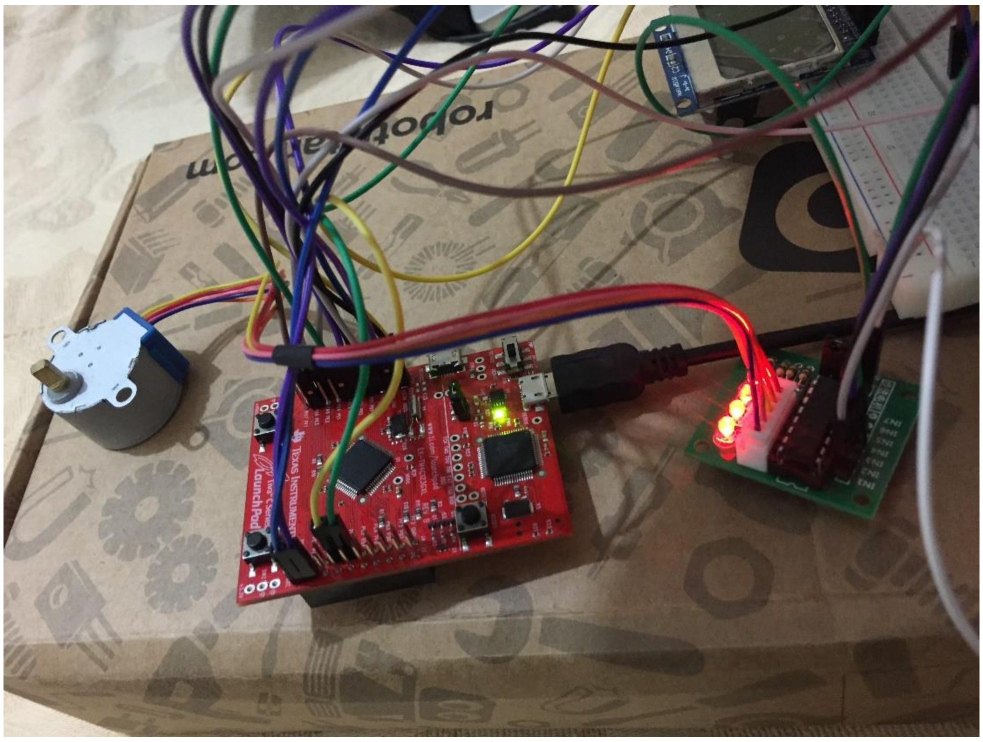
$$LI = Limit\ Interval(Longest\ Duraion - Shortest\ Duration) = 0xB1DF$$

$$ND = New\ Duration = LU + \frac{LI}{SS} * (SS - DD)$$

After these calculations, ND is stored in TIMER0_TAILR, which is responsible for the durations of timeout interrupts. Therefore, time interval between two successive motor steps is adjusted based on the distance between measured value and threshold.

When UPDATE_SPEED subroutine is finished and system goes back to UPDATE_BAR, measured distance and threshold is compared and based on this difference appearance of the bar changes. System goes back into MEAS_2 subroutine.

After updating the status of the bar and adjusting the rotation speed of the step motor, MEAS_2 compares the measured distance and threshold again to see whether the brakes must be engaged or not. If it detects that measurement is lower than the limit, rotation of the motor is stopped and BRAKES_DISP subroutine, which is responsible for the displaying of the message "BRAKES ON", is called.



4.3 Threshold Adjustment

A push on SW1 or SW2 button initiates an interrupt which is defined previously. This interrupt changes the state of the system from "Normal Operation" to "Threshold Adjustment" if Normal Operation is being conducted and SW1 is pressed or deactivate brakes if brakes are active and SW2 is pressed. These operations are done with the help of subroutine PORTF_HAND which is called directly by GPIO PORT F Interrupt service. In both button cases state of the system is changed accordingly.

State of the system is kept in a specific memory location and updated whenever necessary. 0 represents the Normal Op. state, 1 represents Threshold Adj. state and 2 represents that brakes are on. During the Threshold Adj. state, analog data is created with the help of potentiometer and measurements are obtained via PE3. This obtained data is a value between 0 and 0xFFF, so it must be transformed into a value between 0 and 999. SAVE_SAMPLE subroutine is responsible for this operation. In this subroutine, another subroutine named UPDATE_SAMP is called. With the help of tis subroutine, while the measurement stays still and step motor continues to rotate, transformed measured analog value is displayed on the screen continuously.

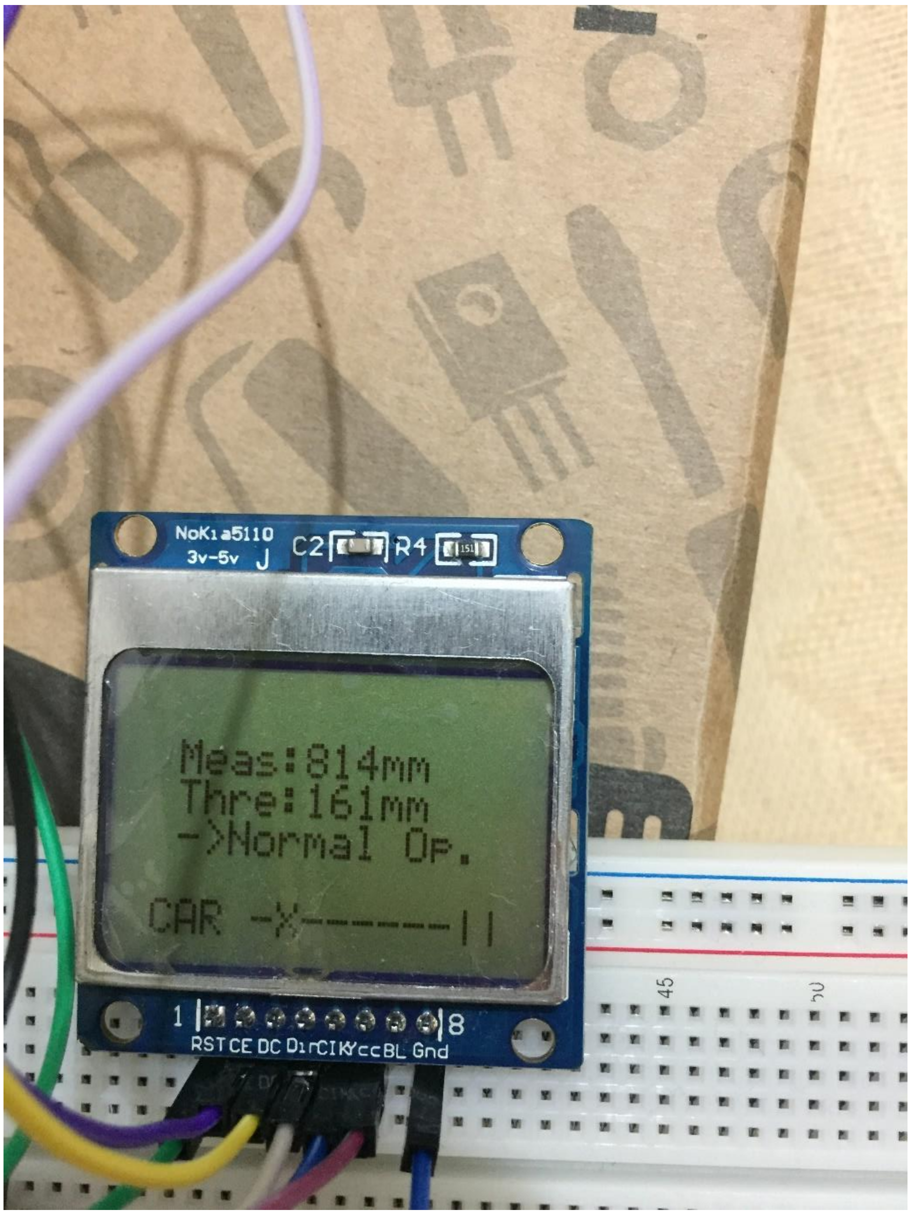
4.4 Main Algorithm

After the initializations, main algorithm works in a really simple way. It checks the value in memory location STAT. This specific location was initialized as 0 and updated with every GPIO interrupt that comes from onboard switch buttons.

4.4.1 If the Value in STAT is 0

Then it means system is in Normal Op. mode. After detecting it program branches to NORMAL_STATE and calls two subroutines which are NORMAL_OP and MEAS_2.

NORMAL_OP subroutine displays the message “->Normal Op.”. Purpose of MEAS_2 subroutine was indicated detailly before. After these operations are done, system branches unconditionally to the beginning where it checks the value in STAT.



4.4.2 If the Value in STAT is 1

Then it means system is in Threshold Adj. mode. After detecting it program branches to THRESHOLD and calls two subroutines which are SAVE_SAMPLE and THRESHOLD_OP. THRESHOLD_OP subroutine displays the message “->Thre. Adj.” and replaces whole bar with “*”. Purpose of SAVE_SAMPLE subroutine was indicated detailly before. After these operations are done, system branches unconditionally to the beginning where it checks the value in STAT.

4.4.3 If the Value in STAT is 2

Then it means the system is in braking mode. It does nothing and loops until an interrupt comes from SW2 which indicates that brakes must be disabled if they are active. This interrupt changes the value in STAT from 2 to 1 and system goes back into Normal Op. mode.