POLITECNICO DI TORINO

OSES ASSIGNMENT 1 – PROXIMITY ALERT SYSTEM

SHAHBOZBEK ABDUNABIYEV

DECEMBER 18, 2015

1. **PROBLEM DESCRIPTION**

A proximity alert system must be designed and implemented using the following elements:

•The HC-SR04 Ultrasonic Ranging Module;

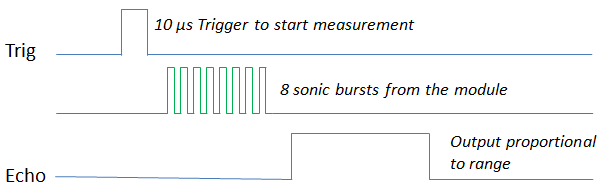
•The FRDM-K64F board;

•The Micrium µC/OS.

The FRDM-K64F shall read the distance of any object located in front of the HC-SR04 sensor, and then it shall operate the RGB led.

1. **SOLUTION**
   1. **Pin Configuration**

In order to configure the pins, we need to know the working principle of the proximity sensor. The following is timing diagram of the sensor:



So, the sensor must be triggered on TRIG input by means of the trig signal and it gives output on the ECHO output. For this reason, we have to configure two gpio pins of FRDM K64F board. One for TRIG of the sensor, another one for ECHO of the sensor.

Consequently, I did the following changes:

Input pin configuration on PORT B, PIN 9 with interrupt enabled on both edges. And it is configured to receive echo signals from ECHO pin of the sensor.

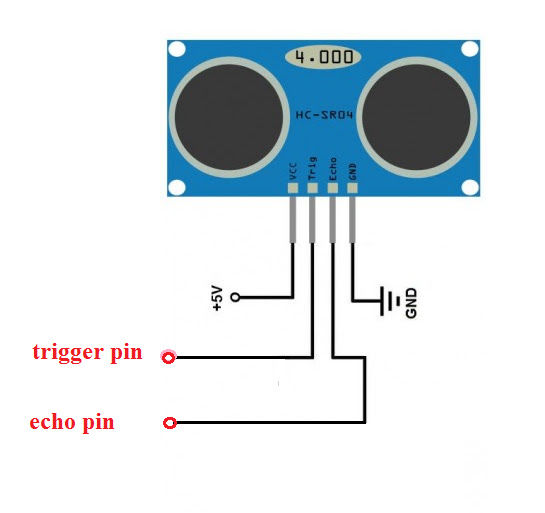
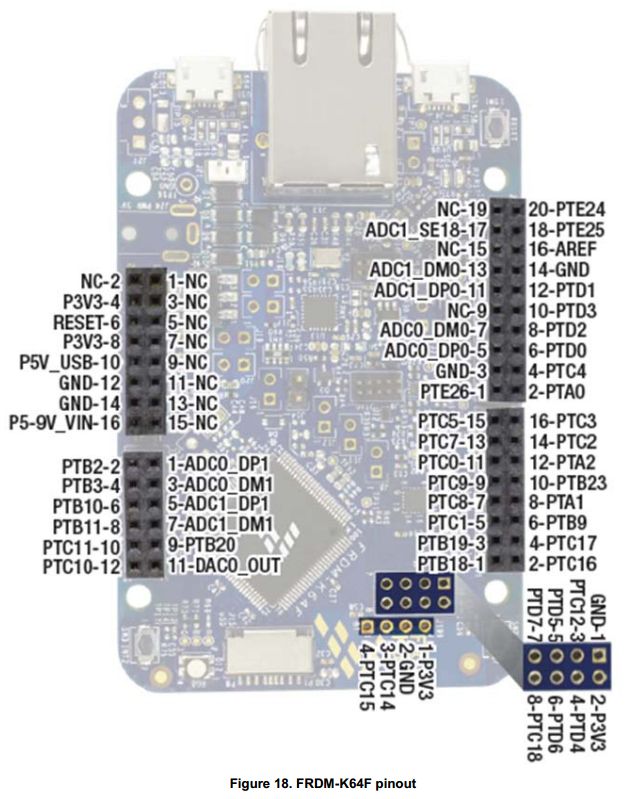
**inPTB9 = GPIO\_MAKE\_PIN(HW\_GPIOB, 9U)**

**PORT\_HAL\_SetMuxMode(PORTB\_BASE,9u,kPortMuxAsGpio)**

Output pin configuration on PORT B, PIN 23. It is used to send a signal to TRIG pin of the sensor.

**outPTB23 = GPIO\_MAKE\_PIN(HW\_GPIOB, 23U)**

**PORT\_HAL\_SetMuxMode(PORTB\_BASE,23u,kPortMuxAsGpio)**

After all, we have following set up : 

* 1. **Software architecture**

I used state machine for the implementation as following:

They define different states of the sensor and there are two use cases: RISE-FALL-OFF, RISE-OVERFLOW-OFF. I used following data structures to keep track of the state machine:

**typedef enum{ //enumeration type for the states**

**RISE, //sensor started measuring**

**FALL, //sensor finished measurement**

**OFF, //measurement finished, ready for the next measurement**

**OVERFLOW // if the obstacle is too far, and there is overflow in the sensor**

**}echo\_state;**

**typedef struct {**

**led\_color current\_led; //current blinking led**

**led\_color new\_led; //new led**

**uint16\_t freq; // blink rate**

**echo\_state state; //state**

**float distance; //measured distance**

**uint8\_t start; // flag to define start of measurement**

**}sensor\_state;**

Moreover, I used two tasks and an interrupt handler:

**static void AppTaskStart (void \*p\_arg);**

The task is responsible for starting the other task and initializing interrupt service routine. Moreover it takes control over RGB LED.

**static void TaskMeasure (void \*p\_arg);**

The task is responsible for measurement. It checks the RISE and FALL time of the sensor and calculates the distance.

**void BSP\_inputB9\_int\_hdlr( void ) ;**

The handler handles interrupts of PORT B, PIN 9 on both edges and changes the state. Here is the main logic of the handler.

**if(sensor\_data.state==OFF){**

**sensor\_data.state=RISE;**

**}else if(sensor\_data.state==RISE){**

**sensor\_data.state=FALL;**

**}**

* 1. **How it works.**

We can see the summary of execution steps of the application:

-I send signal to ECHO pin of the sensor by setting PORT B, PIN 9 to logic 1

-Wait for the interrupt, on RISE get start time, and wait the next interrupt

-Wait for the interrupt, on FALL get end time, calculate distance, set distance, set state to OFF

-Set RGB LED color and rate based on distance

-Wait 50 ms for the next measurement

-In case of the overflow, set state to OFF and restart measurement

We can see the continuous output on terminal. The terminal set for receiving signals from serial port.