POLITECNICO DI TORINO

OSES ASSIGNMENT 2 – PROXIMITY ALERT SYSTEM

SHAHBOZBEK ABDUNABIYEV

JANUARY 03, 2015

1. **PROBLEM DESCRIPTION**

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

•The HC-SR04 Ultrasonic Ranging Module;

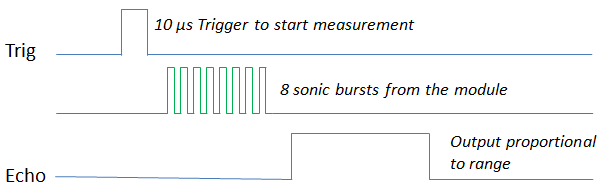
•The STM32F746-DISCOVERY board;

•The Linux OS.

The STM32F746-DISCOVERY shall read the distance of any object located in front of the HC-SR04 sensor, and then it shall operate the user 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 STM32F746-DISCOVERY board. One for TRIG of the sensor, another one for ECHO of the sensor.

Consequently, I did the following changes:

Input pin configuration on D8 (130) with interrupt enabled on both edges. And it is configured to receive echo signals from ECHO pin of the sensor.

**#define ECHO 130 //unique identifier of the d8 pin**

**//reserving pin for usage**

**gpio\_request( ECHO, module\_name )**

**// Set ECHO gpio as input**

**gpio\_direction\_input( ECHO )**

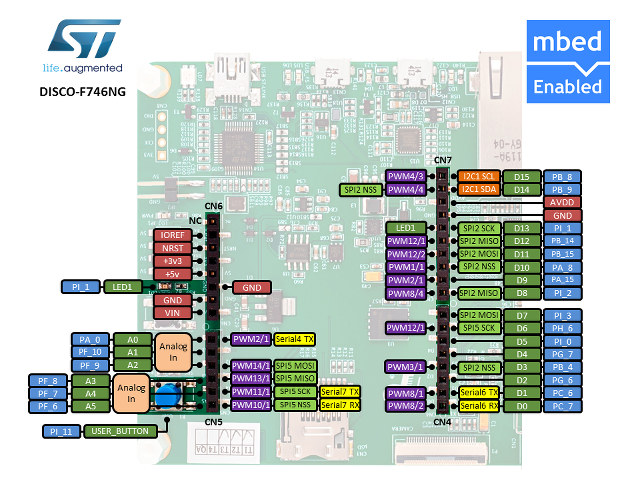
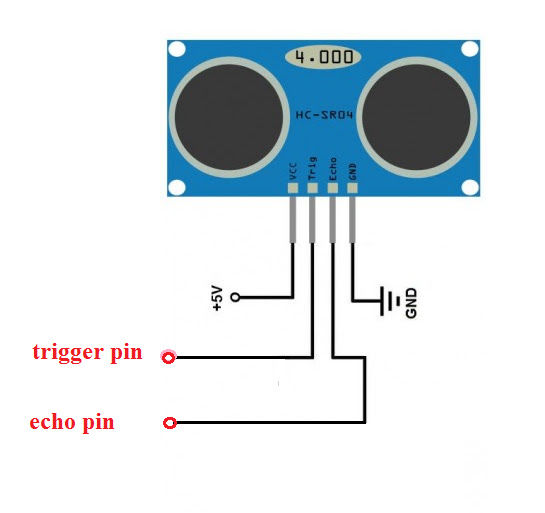
Output pin configuration on D7 (131). It is used to send a signal to TRIG pin of the sensor.

**#define TRIG 131 //unique identifier of the d7 pin**

**//reserving pin for usage**

**gpio\_request(TRIG, module\_name**

**gpio\_direction\_output(TRIG)**

After all, we have following set up : 

* 1. **Software architecture**

Software consists of two parts: linux device driver for hc-sr04 and example application for using the driver. The following are file operations provided by the driver:

* **static int \_\_init dist\_me\_init\_module(void)**

It registers the device and requests and reserves pins for the driver. At the end it creates the work queue for handling user LED tasks. In this way, we can avoid blocking of driver while performing LED toggling. The return value is <0 in case of failure.

* **static void \_\_exit dist\_me\_cleanup\_module(void)**

It frees up all resources and destroys work queue. At the end it unregisters the device.

* **static int dist\_me\_open(struct inode \*inode, struct file \*file)**

It is called when device file opened. It locks the device file and returns file descriptor to user.

* **static int dist\_me\_release(struct inode \*inode, struct file \*file)**

It is called when device file closed. It releases the device file lock.

* **static ssize\_t dist\_me\_read(struct file \*filp, char \*buffer, size\_t length, loff\_t \* offset)**

It triggers the sensor and calculates the time period of the echo signal and returns it to the user:

**ktime\_to\_us(ktime\_sub(echo\_end,echo\_start))**

Where echo\_start and echo\_end are set in the interrupt handler.

**ktime\_dummy=ktime\_get();**

**if (gpio\_get\_value(ECHO)==1) {**

**echo\_start=ktime\_dummy;**

**} else {**

**echo\_end=ktime\_dummy;**

**measure=1;**

**}**

* **static ssize\_t dist\_me\_write(struct file \*filp, const char \*buffer, size\_t length, loff\_t \* offset)**

It receives LED blink rate from the user application and sets the blink rate of the LED and it starts the work in the work queue, if it has not started yet.

There is an interrupt handler used for reacting input signal from ECHO. It sets start and end time based on edge of the signal.

Furthermore, user application receives time period from the driver and calculates the distance:

**read(fd,&distance,sizeof(distance));**

**sscanf(distance,"%lld",&some);**

**rd = (float)some/58;**

Based on the distance it sets the blink rate and passes it to driver.

* 1. **How it works.**

Let’s see working principle of the application:

* **Installing the driver**

insmod /mnt/app/sample.ko

* **Running the application**

/mnt/app/app #of measurements (0 for infinite)

* **Removing the driver**

rmmod /mnt/app/sample.ko