

SWEN 563/CMPE 663/EEEE 663
REAL-TIME & EMBEDDED SYSTEMS
ROCHESTER INSTITUTE OF TECHNOLOGY

[PROJECT-5: QNX NEUTRINO PROGRAM TO MEASURE THE DISTANCE](#)



Submitted to: Larry Kiser(llkiee@rit.edu)

Submitted by:

Mokshan Shettigar (mds8353@rit.edu)

Nicolas Delanou (nxd7208@rit.edu)

D.O.S: 11/28/2016

Areas of Focus

Writing of the main code:

- Nicolas Delanou: 100%

Writing of the report:

- Mokshan Shettigar: 100%

Analysis/Design

Project is about designing a system which can stand-alone QNX Neutrino program to measure the distance between the rear bumper of your car and any objects behind the vehicle while parking.

Analysis:

Driver: When the vehicle is placed into reverse, the parking sensor is activated, providing a continuous stream of data on the distance of any objects behind the vehicle.

Car: The parking sensor is mounted on the rear bumper of the car.

Parking Sensor: When activated, the parking sensor measures the distance between itself (as located on the rear bumper) and any objects within its field of view. The distance is reported to the driver on a continuous basis.

To measure the distance, an ultrasonic sensor is being used. Its accuracy is designed in the software and the range for which it gives the measurement is predefined, for a particular range beyond which the sensor won't detect.

Design:

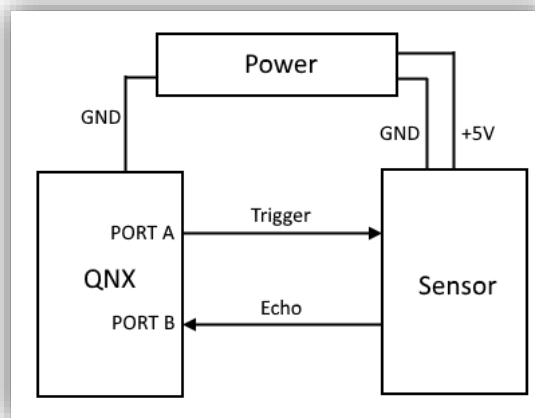
The Design of the system is such a way that the sensor detects any obstruction in the path and returns back the distance from the object, starting point from the sensor. The Software is designed in such a manner that the minimum and maximum accuracy is kept in mind thus making the system as precise as it can be. The Specification of the sensor gives the knowledge about the range, voltage supply and frequency at which it works. Further details about the design is discussed in the following sections.

Test plan

Hardware Description

For this project, we are using an HC-SR04 ultrasonic sensor. This sensor has 4 pins:

- +5V
- GND
- Trigger pin: start a measurement if a 10 μ s long or longer signal is sent on this pin.
- Echo pin: write the measurement result. Result gave by the width of the signal.



Main Function

```
int main( )
{
    /* Give this thread root permissions to access the hardware */
    int privity_err = ThreadCtl( _NTO_TCTL_IO, NULL );
    if ( privity_err == -1 )
    {
        fprintf( stderr, "can't get root permissions\n" );
        return -1;
    }

    /* Time base */
    cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    min_accuracy = cps / 10000;
    max_accuracy = cps / 55.55;

    /* Initialize port A and B */
    InitPorts();
}
```

ThreadCtl(_NTO_TCTL_IO, NULL)

In the main function, first we allow the thread root to access the hardware by using the privileges function “ThreadCtl”. ThreadCtl(_NTO_TCTL_IO, NULL) is used to request the I/O privileges and super locks the process’s memory. This function gives the thread to execute the following I/O opcodes on the architectures in, ins, out, outs, cli and sti and let it attach to the IRQ handlers. If a thread does an attempt to use the opcodes without making a proper call to this threads, the thread faults with SIGSEGV. A SIGILL signal is also caused due to lack of I/O privileges.

SYSPAGE_ENTRY(qtime) ->cycles_per_sec

This is the System Page Entry that returns an entry from it. The entry that is being accessed is the “qtime”. It contains of the several members in the structure, but it contains at least the following :

(a) unsigned long boot_time, (b) uint64_t cycles_per_sec.

We are interested in cycles_per_sec which returns the value of the free running 64-bit clock cycle. Now the maximum and minimum accuracy is defined using the value of the clock cycle.

Send Pulse

```
void* send_pulse(void* data) {
    /* Give this thread root permissions to access the hardware */
    int privity_err = ThreadCtl( _NTO_TCTL_IO, NULL );
    if ( privity_err == -1 )
    {
        fprintf( stderr, "can't get root permissions\n" );
    }
    else {
        /* Send a pulse on the trigger pin */
        while(sensor_activated){
            time_start_measurement = ClockCycles( );
            out8(data_handle_A, 0x01);
            usleep(10);
            out8(data_handle_A, 0x00);
            while((ClockCycles( ) - time_start_measurement) < cps / 10 ); // every 100ms
        }
    }
}
```

When the sensor is activated the function send pulse checks if it has been given the permission to access the hardware. Every time the sensor is activated the data is being sent into the Port A which is set as output Channel. For every 10 microsecond the pulse remains high for which it uses the usleep() function.

Receive Echo

```
void* receive_echo(void* data) {
    /* Give this thread root permissions to access the hardware */
    int privity_err = ThreadCtl( _NTO_TCTL_IO, NULL );
    if ( privity_err == -1 )
    {
        fprintf( stderr, "can't get root permissions\n" );
    }
    else {
        uint64_t time_echo_up, time_echo_down;

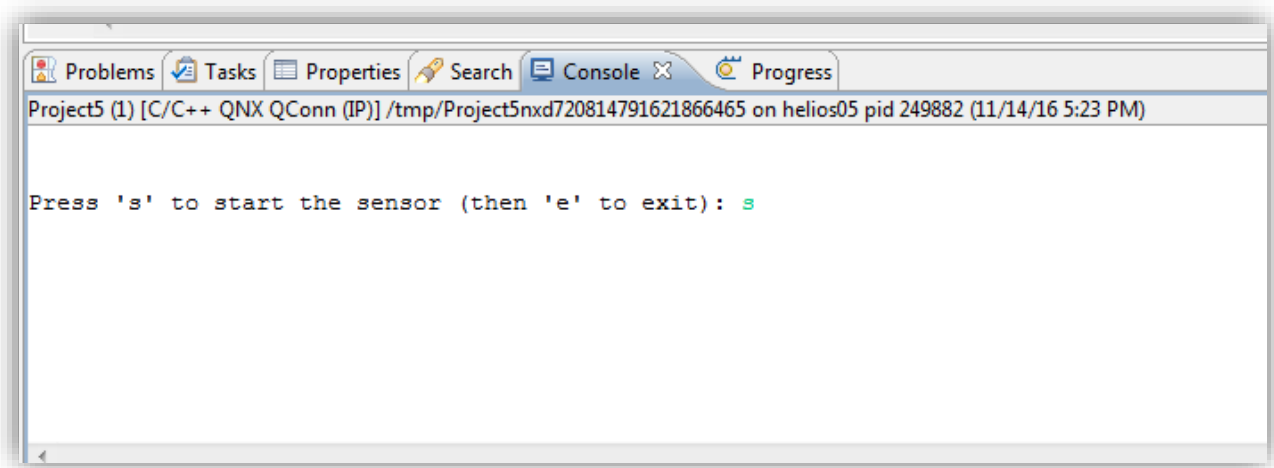
        while(sensor_activated) {
            /* Wait for the echo */
            while(!(in8(data_handle_B)&0x01)); // Use pin 1 of port B
            time_echo_up = ClockCycles( );
            while((in8(data_handle_B)&0x01) );
            time_echo_down = ClockCycles( );

            uint64_t res = (time_echo_down - time_echo_up);

            if(res < min_accuracy || res > max_accuracy ) {
                printf("\n\tdistance: *****" );
            }
            else {
                int val = (res*6750/cps);
                printf("\n\tdistance: %d inches", val);
                min_distance = min(min_distance, val);
                max_distance = max(max_distance, val);
            }
        }
    }
}
```

This function is to receive the echo from the sensor that measures the distance between the sensor and the obstruction. The echo is divided into 2 parts for sending the echo and again for receiving the echo back the object which obstructs it. Now the minimum distance for which the distance sensor can sense is 1 inch and beyond that range it gives an error which will be shown in the below section in the results. After the sensor measures the distance till the time it is being prompted to stop, it will display the minimum and the maximum among all the readings it has measured.

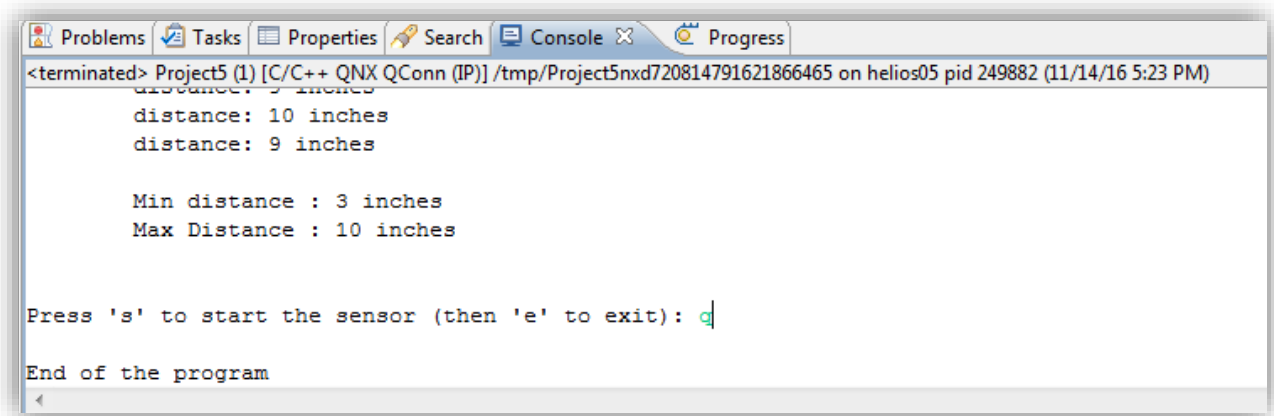
Project Results



```
Project5 (1) [C/C++ QNX QConn (IP)] /tmp/Project5nxd720814791621866465 on helios05 pid 249882 (11/14/16 5:23 PM)

Press 's' to start the sensor (then 'e' to exit): s
```

The above picture is the user interface which prompts the user to start the sensor and thus the sensor starts taking the measurements. Then it can be stopped by pressing 'q' which is in the picture below.



```
<terminated> Project5 (1) [C/C++ QNX QConn (IP)] /tmp/Project5nxd720814791621866465 on helios05 pid 249882 (11/14/16 5:23 PM)
distance: 3 inches
distance: 10 inches
distance: 9 inches

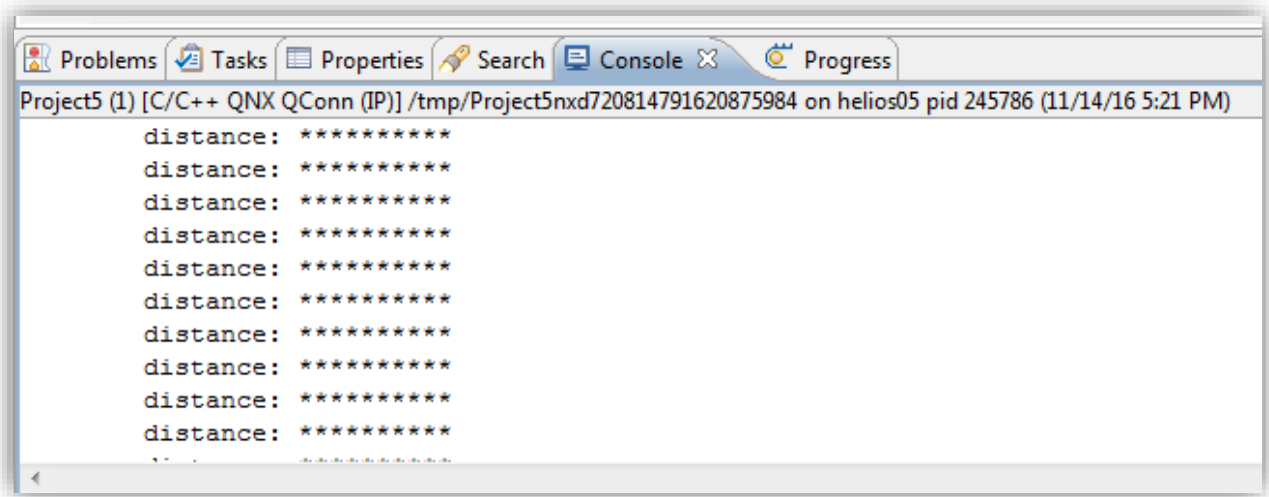
Min distance : 3 inches
Max Distance : 10 inches

Press 's' to start the sensor (then 'e' to exit): q

End of the program
```

As it can be seen that it displays the minimum and the maximum distance out of all the measurements taken and a scale was used to measure the distance before measuring an object was placed at the distance of 3 inches and then advancing the object beyond till the distance of 10 inches.

This sensor can measure distances till 97 inches and after that it gives out of range measures or displays an error as we have designed it to be showing ***** for out of range measurements.



The screenshot shows a QNX IDE console window with the following tabs: Problems, Tasks, Properties, Search, Console, and Progress. The console title bar reads: "Project5 (1) [C/C++ QNX QConn (IP)] /tmp/Project5nxd720814791620875984 on helios05 pid 245786 (11/14/16 5:21 PM)". The console output consists of ten lines, each displaying "distance: *****".

```
Project5 (1) [C/C++ QNX QConn (IP)] /tmp/Project5nxd720814791620875984 on helios05 pid 245786 (11/14/16 5:21 PM)
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
distance: *****
```

This measurement was taken for the distance closest to the sensor for which the system gives out of range output. The results were all precise for all the measurements taken

Lessons Learned

We learned how to program in the QNX environment, using multithreads in real time systems. We also learned how to use QNX Timers but we decided not to implement it in our program.