# Interrupts and Real Time
EE/CSE 474 Lab 4

Daisy Xu, Yunie Yi, Abigail Santos

November 18, 2016

# Table of Contents:

## 1. Introduction

The purpose of this laboratory was to learn/familiarize ourselves with interrupts and real time machine controls in user space. In the first part, we utilized distance sensors connected to the ADC ports of our beaglebone. One sensor was attached to each of the sides of the tank, from which we took reading samples at approximately 333 Hz. A moving average of the values read within the last second is calculated for each of the sensors. This information can be used to interrupt the master program to control our logic for the tank motor.

In the second part of the lab, we implemented an H-bridge, a circuit that allows for current to flow in two opposite directions. This is what enables the tank wheels to turn both forwards and backwards. The H-bridge is driven from GPIO inputs from the beaglebone and is supplied by an external battery pack.

Now having the sensors and H-bridge set up, we designed our master program motor.c. Using the information from the sensors which is sent as interrupts, our code was designed with an automatic driving functionality to navigate around objects detected in the distance sensors. As an additional functionality, we added a warning sound that beeps whenever the tank is moving too close to an object in front of it.

## 2. Discussion Of the lab & Specifications

### 2.1 ADC Sampling and Interrupts

Each of the four distance sensors, one on each side of the tank, was connected to an ADC pin on the beaglebone as shown in **Table 1** below. “Front” corresponds to the side of the tank closest to the beaglebone.

| **Sensor Connection** | **GPIO Pin** |
|---|---|
| Front Sensor Output | AIN0 |
| Back Sensor Output | AIN6 |
| Right Sensor Output | AIN2 |
| Left Sensor Output | AIN4 |
| Sensor Power | VDD 3.3 V |
| Sensor Ground | DGND |

**Table 1: Implementation of Distance Sensors**

The distance sensors output a value from 0 to 1799 representing the current voltage of the pin in mV. A larger voltage corresponds a closer object and vice versa. Our slave program, handler.c, uses a general interrupt that can feed these values into a pipe linked to a master program. A summary of handler.c is shown below in **Table 2**.

| **Method Name** | **Method Description** |
|---|---|
| void timer_Init() | Creates a timer every 3 ms (approx. 333 Hz) that calls adc_handler to sample the ADC pins. |
| int readADC(unsigned int) | Opens and reads from the 4 ADC pins. |
| void adc_handler(int) | Stores the last 300 read values for each sensor and calculates their means, which is written to a pipe to communicate with a master program. |

**Table 2: Summary of Handler Methods**

## 2.2 H-bridge

After connecting the distance sensors, we had to implement the H-bridge. A diagram of the H-bridge is shown below in **Figure 1**. A description of the pins and connections is also shown below in **Table 3**.

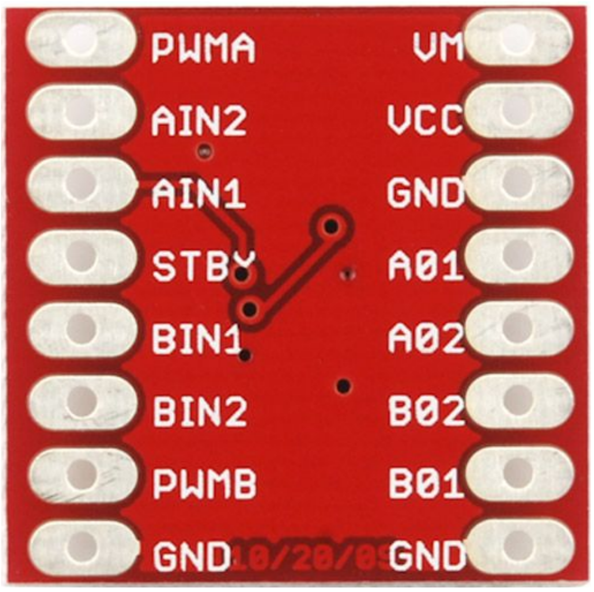

**Figure 1: Pinout of H-bridge**

| H-bridge Pin | Description | Connection |
|---|---|---|
| PWMA | Duty Cycle controls left motor speed | PWM EHRPWM1A |
| PWMB | Duty Cycle controls right motor speed | PWM EHRPWM1B |
| AIN1 | Determines direction of left motor | GPIO 49 |
| AIN2 | Determines direction of left motor | GPIO 48 |
| BIN1 | Determines direction of right motor | GPIO 115 |
| BIN2 | Determines direction of right motor | GPIO 20 |
| AO1 | Output to left motor | Left Motor (Red Wire) |
| AO2 | Output to left motor | Left Motor (Black Wire) |
| BO1 | Output to right motor | Right Motor (Red Wire) |
| BO2 | Output to right motor | Right Motor (Black Wire) |
| STBY | Standby for power saving, allows H-bridge to work when pulled high | GPIO 112 |
| VM | Motor Voltage | External Battery Power Supply |
| VCC | Logic Voltage | SYS 5 V |
| GND | System Ground | DGND |

**Table 3: H-bridge Pin Descriptions and Connections**

As shown in the table, the "A" pins correspond to the left motor and the "B" pins to the right motor. The logic controlling the two motors is summarized in **Table 4** below.

| In1 | In2 | PWM | Out1 | Out2 | Mode |
|---|---|---|---|---|---|
| H | H | H/L | L | L | Short Brake |

| | | | | | |
|---|---|---|---|---|---|
| L | H | H | L | H | Counter Clockwise |
| L | H | L | L | L | Short Brake |
| H | L | H | H | L | Clockwise |
| H | L | L | L | L | Short Brake |
| L | L | H | OFF | OFF | Stop |

**Table 4: Motor Operation Modes**

## 2.3 System Integration, Motor Controls, and Warning Alarm

With the sensors and H-bridge connected, it was time to integrate the system such that the tank could automatically drive and navigate on its own. To do this, we created a master program motor.c that controls the motor of the tank based on information passed to it from the pipe linked to handler.c. A summary of the methods used is found below in **Table 5**.

| **Method Name** | **Method Description** |
|---|---|
| void timer_Init() | Creates a timer every 3 ms (approx. 333 Hz) that calls sig_handler(). |
| void sig_handler(int value) | Reads the front, back, left and right sensor values and prints to console. Calls nextStep() to decide motor behavior having updated the values. |
| int setUp(); | Initializes GPIO and PWM pins. |
| void digitalWrite(FILE*, int); | Writes the given value to the given file. |
| void pwmaWrite(int speed)/ pwmbWrite(int speed); | Takes a speed input and writes to the corresponding PWM in order to control motor speed. |
| void drive(int direction, int speed); | Moves the tank forward or backward at the given speed.<br>Forward: direction = 1<br>Backward: direction = 0 |
| void stop(); | Stops the tank from moving. |

| | |
|---|---|
| void rotate(int direction, int speed); | Rotates the tank left or right at the given speed.<br>Left: direction = 1<br>Right: direction = 0 |
| void nextStep(int front, int back, int left, int right); | Controls the movement of the tank given the current front, back, left and right sensor data. Makes the tank move in order to avoid running into objects. |

**Table 5: Summary of Motor Methods**

## 3 Test Procedure & Cases

### 3.1 ADC Sampling and Interrupts

**Case 1: Read and Print 4 Sensor Values to Console**

Each of the 4 sensors should be read from at a speed of approx 333 Hz. The mean of the last 300 values, which equates to around the last second, should be calculated for each sensor and printed to the console.

### 3.2 H-bridge

**Case 1: Signal Verification w/ LEDs Connected to H-bridge Outputs**

Inputs AIN0, AIN1, BIN0, BIN1 control the outputs AO1, AO2, BO1, BO2, of both motors. As previously seen in **Table 4**, certain outputs should be on whenever the wheels should move. Each combination of inputs can be verified by connecting an LED to each of the 4 outputs to visually confirm the logic of the H-bridge.

**Case 2: Direct Power to Motor**

It is easy to confuse the VCC power supply for the H-bridge logic and VM power supply for the motor. That is, the motor should be supplied from an external 6-15 V power supply such as through batteries, while the H-bridge should be supplied from the beaglebone SYS 5 V power. The motor should move when either of the channels are connected to the battery power supply.

### 3.3 System Integration, Motor Controls, and Warning Alarm

**Case 1: Motor Direction/Behavior**

Our program is designed to automatically drive the tank to move and change directions based on the distance sensors' signals. By default, the tank will move forward if the front is not blocked. However, if the front is blocked, our nextStep() method provides the logic to control the motor in four unique cases as described below.

- Case 1: Only Front Blocked
    - Rotates right until front is not blocked
    - Once front is clear, moves forward

- Case 2: Front and Right Blocked
    - Rotates left until front is not blocked
    - Once front is clear, moves forward

- Case 3: Front/Right/Left Blocked
    - Moves backwards

- Case 4: All Blocked
    - Do not move

**Case 2: Test Warning Alarm**

Our pwm.c is designed to generate warning alarm when an object is near the front, i.e. the front sensor average value gets closer to the threshold value. Likewise, once the object is gone or is further away, the alarm should stop ringing.

## 4. Results

### 4.1 ADC Sampling and Interrupts

We successfully implemented all 4 distance sensors and were able to read their values, calculate a mean, and send it through a pipe to the motor.c file. While the process was completely correct, it is worth noting that the sensors are very sensitive and somewhat unreliable because the output voltage vs distance is an exponential relationship. This means that the sensor values can make large jumps at short distances.

### 4.2 H-bridge

The H-bridge was implemented without issue. It worked just as intended with any valid input provided.

### 4.3 System Integration, Motor Controls, and Warning Alarm

The system integration was successful. Our motor.c correctly read from the pipe linked to our handler.c file to gather the mean of the data from the 4 sensors. It used this information to correctly move the tank wheels in order to move away from nearby objects. The warning alarm was also successful in activation, firing off whenever an object is too close to the front of the tank and stopping when the object got further away. However, for an unknown reason, there was lag in reading the sensor values whenever the beeping was active.

## 5. Error Analysis

There were several mishaps throughout the development process of this project. Regarding the distance sensors, we noticed that whenever an object was placed directly in front of one sensor, which would read the max value of 1799 for more than a few seconds, the other sensor values began to approach the max value as well. This occurred even though no other movement happened in front of the other 3 sensors. In order to avoid sensor data corruption, this meant we had to be careful not to place any objects too close to the sensors. We also noticed that occasionally while printing the sensor values to the console, all four values would be reading fine before suddenly resetting and getting stuck at 0. We could not determine the cause of this problem, and while it happened several times, rebooting the beaglebone always fixed this issue.

In the beginning we also had trouble getting our motor to start before enough debugging told us that our batteries were not functioning. Despite having charged the batteries for several hours, they were providing next to no power. This happened when we tried a new pack of batteries as well. Our solution was to buy our own packs of non-rechargeable batteries which was able to successfully power the motor. Occasionally we also thought our motor had stopped working when in fact the batteries had just been used up at a surprising rate.

Our last major error was due to user error. We inadvertently switched up the leads to our power supply, effectively shorting the beaglebone and breaking it. We were fortunate enough to get a new beaglebone, which we were much more careful with.

## 6. Summary and Conclusion

In this lab, we successfully drove the tank by use of 4 distance sensors providing information via interrupts and an H-bridge to drive the left and right motors. The distance sensors gave an analog value to quantify how close objects are to the tank. We used a handler, a "Slave", to feed this information through a pipe by means of interrupts to our motor controls. The "Master", our motor controls, could read and use the information to determine the most appropriate movement for the tank. This was determined by whether or not the sensor values were passed a threshold indicating objects are too close to one of the tank sides. We also implemented an alarm that warns when an object is too close to the front of the tank.

Now having developed a more solid understanding of interrupts, signal communication, and real-time data, we hope to expand upon this lab in the following weeks. Specifically, we hope to add various functionalities including but not limited to the ability to run via bluetooth untethered and make a more impressive system.