EEET2256

Introduction to Embedded Systems

Final Project: Micro-controller Input-Output

Marcela Klocker, s3487194

Joshua Robertson, s3486608

Introduction:

In the 21st century technology has become a focal point of many peoples lives. The use of mobile phones, computers, laptops and remote controls have become an aspect of daily life. Self driving cars have even begun to emerge and are increasing in popularity. All these technologies have several things in common, one of those things being embedded systems. Every piece of digital technology can be broken down into different embedded systems each consisting of microprocessors, wires, resistors, capacitors - all sorts of components! For this reason the understanding of these microprocessors and components that make up embedded systems is becoming more and more important. Not only does hardware make up the diverse range of technology used but software is fundamental. Without the software, without the brains to these devices, they are merely lifeless corpses.

This report will have a strong focus around the development of these embedded systems at their most fundamental levels. This revolves around monitoring and controlling a simple I/O device connect via ports of a microcontroller. This includes writing the software that makes decisions based on the inputs it receives and using that information to control the output device.

The microprocessor being used is the ATMEGA32 mounted onto the Open-USB-IO (will be referred to as the OUSB). It is a high performance, low-power Atmel AVR - 8-bit microcontroller. Its features include an advanced RISC Architecture, High endurance non-volatile memory segments, JTAG interface, Peripheral features, special microcontroller features and I/O and packages. More information about this microprocessor can be found in the ATMEGA32 Datasheet. The OUSB board is very versatile. It allows the user to control digital and analogue hardware from the USB port of a windows or a linux PC. The ATMEGA32 can also be directly programmed. It is also completely free to use being an open source project for both hardware and software design. More information about the OUSB board can be found on the website www.piradcliffe.wordpress.com. Figure 1 displays the OUSB board with some labeled features.

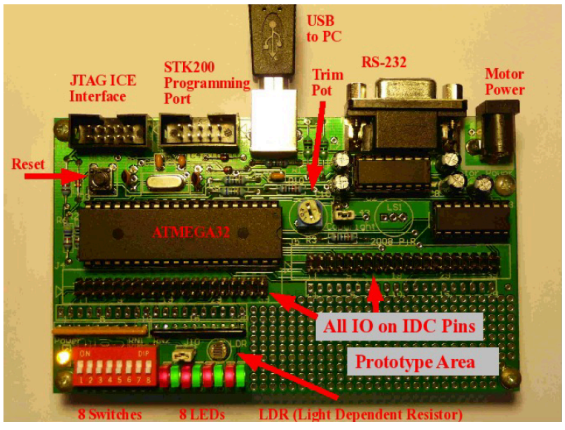


Figure 1. An Open-USB-IO with labeled features.

Using the OUSB and various different I/O devices a simple collision avoiding robot was constructed and programmed using the Assembly language. The robot was named Eebie and will now be referred to as Eebie. The robot was assembled from an Arduino compatible kit, a breadboard, OUSB, and ping sensor. Eebie is shown below in figure 2. He features a metallic body, rubber wheels and four separate motors. Some may say he resembles the popular disney robot WALL-E. Whilst he may not be advanced as a robot like WALL-E there are features the two share in common.

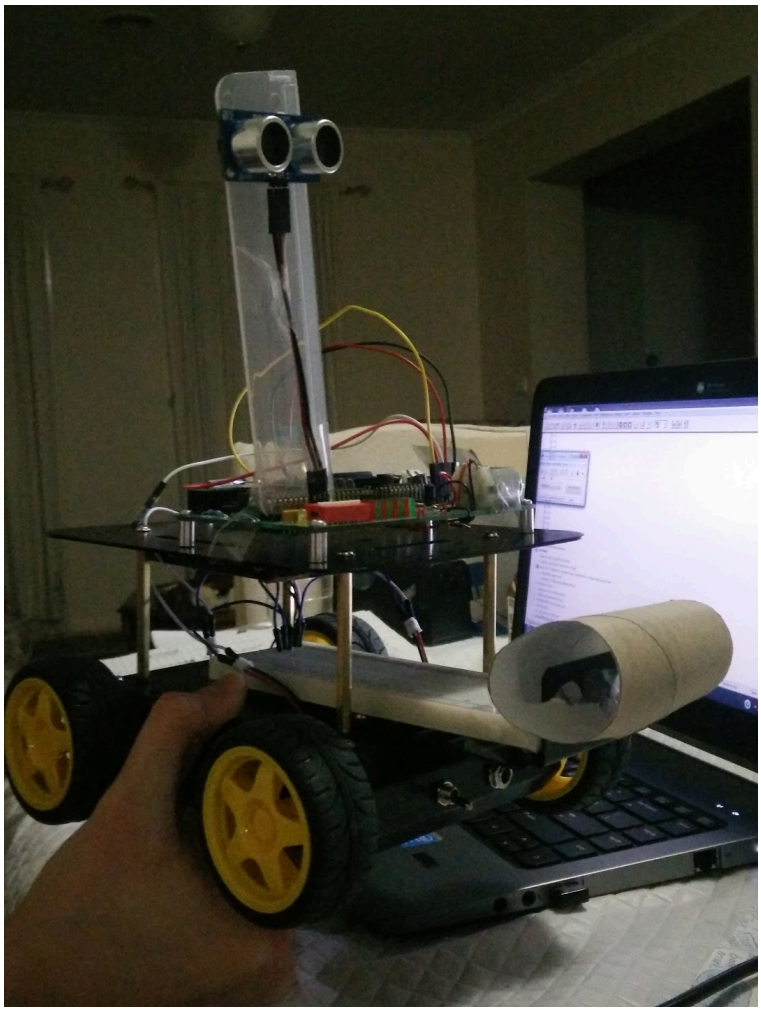


Figure 2: Eebie

Eebie’s main functionality is to avoid large obstacles. He is able to move around a room in straight lines, turning and reversing in order to avoid large obstacles like walls. This is achieved by programming the ping sensor to continuously take readings of the distance between it and the obstacle directly in front of it. It is important to note the location of the ping sensor, it is elevated above the robot's main frame and is facing directly forward. The sensor elevation is to avoid reflections off the floor as warned in its datasheet. A ping sensor functions by sending an ultrasonic pulse and waiting its echo to return. The time taken for the echo to return is measured and can be used to find the distance between the sensor and a surface. This means Eebie is only capable of measuring the distance to objects directly in front of him. Objects that may be on a diagonal or slightly too far to the right or left of Eebie will not be detected by the ping sensor. When an object is detected, Eebie continues its path directly towards it until it reaches a certain distance. Once this distance is reached the robot will turn leftward. If it turns leftward and an obstacle is detected immediately within a close proximity , the robot will reverse and continue to turn. It is continuously measuring distances and moving. Hence it will continue to turn and reverse until there is no longer a nearby obstacle.

Eebie is still far from being a perfect collision avoiding robot. There are many improvements to be made. An example of an improvement is the addition of more ping sensors so that the robot can better decide on a clear path to travel. A simple addition would be a rear sensor so that the robot cannot get stuck reversing into an obstacle when the front sensor detects something close. A larger improvement would be the capability of Eebee's motors to rotate bidirectionally so that turning wasn’t such an effort. He however serves the small purpose of avoiding large obstacles and moving slowly around a room.

How Eeebie works:

Left over right.

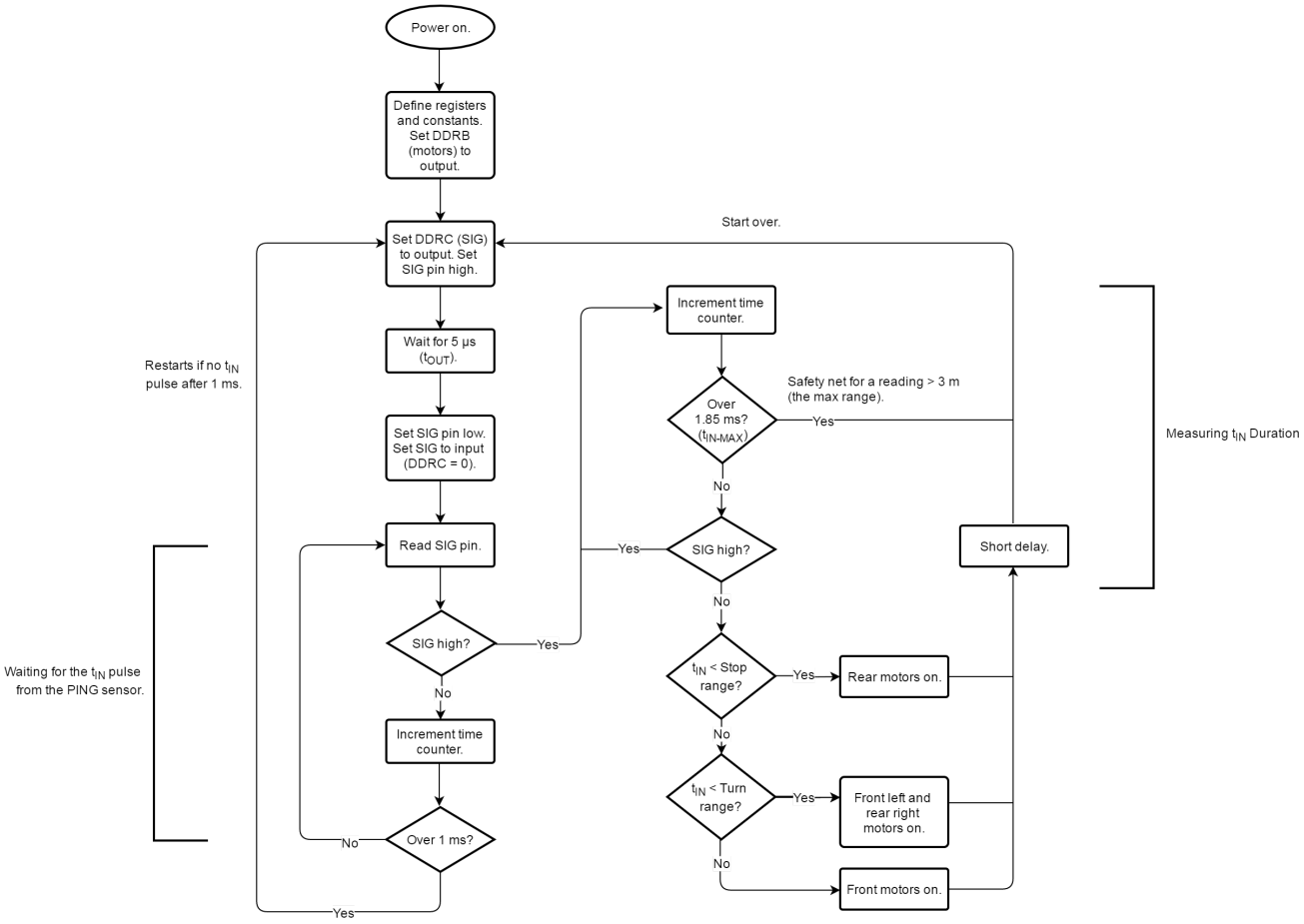
Eebie’s motors are driven by PB0-PB4 Open Collector drivers. These drivers could not be used as grounds meaning the motors couldn’t each be connected to two drivers allowing them to move both directions by having only one or the other connected driver high while the other was grounded. This means that Eebie is limited in how he can move. We chose 3 main movements; forward, reverse and rotate left. To achieve this the 4 tyres and therefore the 4 motors are controlled independently and the front and rear motors spin in different directions. The front motors spin to go forwards, back motors to reverse and the front right and rear left motors spin to rotate left on the spot. Eebie could turn right but it was deemed pointless without additional ping sensors or some kind of long term memory in Eebie’s programming. Eebie turns left to go

right. Eebie's path sometimes curves and this is due to switching between rotation and movement commands that become indistinguishable while Eebie is rolling.

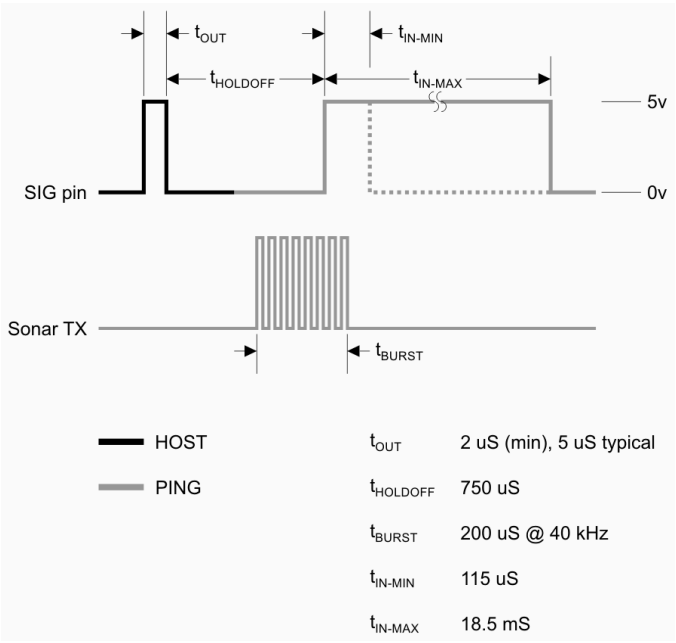
Sound over sight.

Eebie chooses which movement is best by using it's ping sensor to tell how far in front the nearest obstacle is. If there isn't anything too close he drags himself forward on his front tyres. If there's something coming up he will rotate until either there is room in front to move forward or something is found that's too close. When something is too close Eebie reverses using his rear tyres until the obstacle is within rotating range. This wasn't the first design of Eebie's decision making process; Originally Eebie's tyres all faced forward and he could speed forward and would never struggle to turn on carpet. However with the original logic if an obstacle got too close the only option was stop and this would tend to result in short lived adventures. Being able to reverse meant Eebie would generally much greater amounts of time. Much of the time when Eebie does stop it's due to a surface making rotating too difficult.

The flowchart below details the logic behind using the ping sensor and controlling Eebie.



The image to the right shows the functionality of the ping sensor. The HOST section of the top signal is marked as t_{OUT} in the flowchart. The PING section of the signal is highlighted in the flowchart by the two annotations marking when the system is waiting for the t_{IN} pulse and when it is measuring the t_{IN} pulse. The different times in the datasheet figure on the right lead to much of the constants seen in the flowchart.



Testing:

Testing was broken up into three main components, testing the ping sensor, testing the motors, and testing the integrated design. However the approach wasn't in a waterfall method, rather a more agile method. That is that despite breaking the testing up into three different phases, each phase was continuously being re-checked and altered to fit the design better. The process taken was fairly informal. There was no real documentation of the testing (as there would be in a professional environment) and the testing itself wasn't conducted to a strict time line either.

Ping sensor:
The first part that was trialed and tested was the ping sensor. This was done while it was detached from the main body of the robot. The first thing that was tested were the delays needed to get the ping sensor to work. In order for the ping sensor to work, a high pulse must be sent to the sensor for 5 microseconds, then after delay of 750 microseconds the ping sensor returns a pulse with duration that of the ping's travel time..
Getting these delays correct was the first thing that was tested. The delay loops were coded to be as accurate possible, however they needed to be tested. The distance was displayed on the LEDs. If the object was very close, all the LEDs would turn on. The further away the object the less LEDs that turned on. Testing this went smoothly and quickly. A few things were noted during this phase. Firstly, the ping sensor is "touchy". It became confused with objects that were not directly in front of it. Smaller objects too caused issues with the sensor, producing incorrect readings. It was found it worked best when trialed in front of large objects like a wall.

Motors:
The hardest part about getting the motors running was figuring out which pins are dedicated motor pins on the OUSB board. One this was established the motors were wired up and integrated into the code. They ran smoothly first go!

Integration:
This was probably where more of the testing happened. The first test was simple. We had programmed Eebie to move forward until it reached a certain distance from an obstacle. When this occurred Eebie was support to turn left. However the first run didn't go as planned. He approached the wall and at an appropriate distance he began turning left. However he did not turn enough and hit the wall on an angle. Fortunately we had a bumper bar to protect the wall and Eebie! Test number two worked well. The code was updated to turn for longer. Eebie turned and avoided the wall, however collided with a couch that was on an angle to him. The ping sensor did not work well with diagonals. The next software update was alternating between turning and measuring the distance. This did not resolve the bumping into the couch issue as Eebie turned more than usual but still hit the couch once the angle between the couch and the ping sensor was too great. Resolving this took a few different tries. Eventually reversing was implemented and this resolved the final issues.

Challenges:

For the most part, Eebie was designed, constructed and programed with no real issues. He intially worked well in a room full of square objects perpendicular to each other. The real challenges revolved around programming eebie to avoid obstacles perpendicular to him. Or if eebie turned left to avoid an obstacle and found another obstacle directly left. Turning left more may result in a collision, and turning right would result in a collision with the first obstacle. Working around this took longer than anticipated. The solution was implementing a reverse feature.

Another challenge, one that was resolved more easily, was figuring out which pins to connect the motors to. Also how much power they needed to rotate. A battery pack was needed as Eebie needs to be able to move around without being attached with a computer.

Reflections:

Marcela:

After completing this project I would now rate the difficulty as a 3 for so-so. My original evaluation rated the difficulty as a 4. The difference between these two is small. I think i've rated it as easier as everything appears easier once you have already done it. I initially gave it a difficulty of 4 because everyone around me was telling me it was very hard and very time consuming. I feel i disagree with those people now. It was a very enjoyable and fun task. Yes we hit several roadblocks but where's the challenge in a project that runs super smoothly? I feel this project was a good level of difficulty for my level but definately not 4. I feel more confident now about taking on "hard" tasks. The hardest part was not procrastinating. Which is generally the hardest part about everything.

If i was getting paid to do this in a professional environment I would do a lot differently. My planning and documenting would be much more detailed. I would have also stuck to the timeline. Conducted more research before starting. And made the final product look more pretty and nice. For example an exterior case or something to hide all the wires.

Joshua:

I personally found the most difficult/challenging part of this project to be finding out what resources we had available to achieve our tasks. I generally knew exactly what kind of functionality I was looking for whether it was code or hardware however I feel it was time consuming discovering everything I ended up using. Examples of this include finding the motor drivers on the board and the code necessary to control them.

If this were a paid project I think I would ask many more questions about what exactly the customer wanted. In this project we had a good degree of freedom to allow our design to evolve as we ran into problems and came up with improvements. In reality I think the customer would have a very specific functionality in mind and you'd have to get a detailed description out of them and also have them more involved in the whole process.

I now rate the project a 2-3 since I know more what's available to use and adding to the robot isn't daunting at all. I rated it initially as a 4 because I didn't know how to code/build what we had in mind. Next time I'll have a better foundation and I'll know to research my resources better before jumping straight into the design.

Evaluate the complexity of the question once again using the scale from Step 1. Reflect (in writing) why not you have evaluated the question as one level **more** difficult this time. Also reflect on the reasons for any discrepancy between the original (Step 1) and the final (Step 2) evaluation and on the actions you need to undertake to become more confident with a similar task next time.