DIT085 Ed 2016 Practical Phase 1

Group 3.1
Gabriel Marian Bulai
Enrique Cordero Miranda
Hampus Gunnrup
Marcus Nilsson

1. Introduction

The development of the Arduino communication module involves the implementation of four interfaces: sending sensor data, reading speed data, removing and adding bits from and to the buffers.

2. Interface and Test Design

The packet structure consists of a series of 8 bits representing the input parameters(torque, ultra distance and ir distance) and an error detection(1 byte)each of them separated by a delimiter of one byte for a total amount of 30 bits.
The structure would be like D+P+D+P+D+P+D+E+D
1+ 8+1+ 8+1+ 8+1 +1+1= 30 bits
code structure 3 params: /00000000,00000000,00000000,0*
code structure 2 params: /00000000,00000000,0*

Description: Send sensor data
Pre-Condition: torque, ultra_distance, ir_distance
Post-Condition: binary 30 bits packets
Test-Cases:

	TC1	TC2	TC3	TC4	TC5	TC6	TC7
T(-30…30)	T	-	-	-	-	-	-
T(<-30)	-	T	-	-	-	-	-
T(>30)	-	-	T	-	-	-	-
IR(-1…40)	T	-	-	-	-	-	-
IR(<-1)	-	-	-	T	-	-	-
IR(IR>40)	-	-	-	-	T	-	-
Ultra(-1…50)	T	-	-	-	-	-	-
Ultra(<-1)	-	-	-	-	-	T	-
Ultra(>50)	-	-	-	-	-	-	T

Send Data	T	F	F	F	F	F	F

Description: Read speed and torque
Pre-Condition: bitstream (binary 30 bits packets)
Post-Condition: object with speed and torque attributes that searches for packets, reads the bits, runs error detection, returns reference.

Test-Cases:

	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
Amount of bits ok	T	T	T	T	T	T	T	F
bits in front	F	T	-	-	-	-	-	-
Start found	T	T	T	T	T	T	F	-
end found	T	T	T	T	T	F	-	-
1st comma found	T	T	T	T	F	-	-	-
2nd comma found	T	T	T	F	-	-	-	-
parity check	T	T	F	-	-	-	-	-
return an successful object	T	T	F	F	F	F	F	F

Description: Write to input buffer
Pre-Condition: two parameters -> n(non negative number), s(bitstream)
Post-Condition: returns object containing 0 if the operation has been successful or a constant ≠ 0 value denoting an error code.

Test-Cases:

	TC1	TC2	TC3
n>=0	T	T	F
negative n	F	F	T
s<n	F	T	F
s>=n	T	F	T
error code	0	1	2

Description: Read from output buffer
Pre-Condition: n(non negative number)
Post-Condition: returns object containing 0 if the operation has been successful or a constant ≠ 0 value denoting an error code and a bitstream.

Test-Cases:

	TC1	TC2	TC3
n >= 0	T	T	F
Bitstream >= n	T	F	T
Object.code	0	1	2
Object.stream	n bits	n bits	NULL

3. Test Driven Development

Assumptions made:
Packets size that represent interface of send sensor data are 30 bits each.
The ones that represent the interface reading speed and torque are 21 bits each.
The size of the buffer allows the transmission of the whole packet at once without splitting.
Error codes: 1 if streambuffer < n
2 if n is negative
Ranges in the test cases:

- Torque: (-30, 30)
- Ir Distance: (-1, 40)
- Ultra Distance: (-1, 50)

See code appended to the archive.

4. Step By Step Implementation

For each of the functions we implemented, we worked in a TDD manner. First a jUnit test was created to fulfill one of the test cases regarding a method. Then we ran the test and watched it fail. After a test failed we implemented enough code for the test to pass. Finally we started on the next jUnit test for the same method. This was done for each method being implemented, and the end result was a fully tested implementation of a simulated arduino communication.

The Arduino receives data from the sensors and communicates with Odroid	Action	Expected Response(output)
	Arduino receives sensor values within ranges	returns a bitstream with converted values
	Arduino receives sensor values out of range	returns an empty bitstream

The Arduino reads speed and torque from Odroid system and returns references	Action	Expected Response(output)
	Arduino receives an uncorrupted packet	returns an object with the speed and torque values
	Arduino finds bits in the front of the uncorrupted packet	returns an object removing the garbage and saving with the correct values
	Arduino finds an even amount of ones and the parity check number is 1	it fails the parity check and returns an object with success = false
	Arduino finds an odd amount of ones and the parity check number is 0	it fails the parity check and returns an object with success = false

The Arduino writes data into the input buffer and returns error codes	Action	Expected Response(output)
	number of bits (n) ≥ 0 bitstream (s) ≥ n	it returns an error code 0 (success)
	bitstream (s) < n	it returns an error code 1
	number of bits < 0	it returns an error code 2

The Arduino reads data from the output buffer and sends an error code and a bitstream to Odroid system	Action	Expected Response(output)
	number of bits (n) ≥ 0 bitstream (s) ≥ n	it returns an object with an error code 0 and a bitstream s
	bitstream < number of bits	it returns an object with an error code 1 and a bitstream
	number of bits < 0	it returns an object with an error code 2 and bitstream = NULL

5. Code Snippet Explanation

Write to Input Buffer

```
//Write to Input Buffer
public int writeToInputBuffer(int n, String s) {
    if (n<0) {  //TC3
        return 2; // error code for trying to write negative amounts of bits
    }
    else if (s.length() < n) { //TC2
        inputBuffer = inputBuffer + s;
        return 1; // indicate not enough bits received
    }
    else { //TC1 N > 0 & S >= N
        inputBuffer = inputBuffer + s.substring(0,n);
        return 0; // successful test
    }
}
```

This function is one of the clear ones since there are three test cases identified directly with three different comparisons (if clauses). In two of the test cases there is enough information (a valid n) to write something to the buffer. The test cases test both that the expected output is there but also that the buffer is in the correct state (right information) after the write.

```
String expectedBuffer = "11110000";
Assert.assertEquals(expected, actual);
Assert.assertEquals(expectedBuffer,arduino.inputBuffer);
```

Here is an example of the check of the output buffer. There is an expected buffer value and it checks against the actual buffer value.

Read from Output Buffer

```
if (n < 0) { //TC3 trying to read negative bits
    answer = new ReadAnswer(2, ""); //2 indicating negative bits
}
else if (outputBuffer.length() < n) { //TC2 //n>0 s<n
else { //TC1 n>0 s>=n
```

This is the three tests implemented in the actual code to read from the output buffer. It has three checks which represent the three tests cases. The test cases also take into account that if the read is possible then the buffer should be edited to show the removal of the read bits. In case the packet is corrupted then it just deletes the bits up to the start of the packet. This because the next iteration the start of the packet will be the next step to start so all the bits in front will be ignored.

Read Speed and Torque

```
if (bitstream.length() < minbitstreamsize) {//TC8
    answer = new SpeedTorque();
}
else if (bitstream.indexOf(start_Del)<0) {//TC7
    answer = new SpeedTorque();
}
else {
```

This is the start of the method where it checks that the amount of bits is the correct amount. It complies with two of the test cases.

```
else if (ParityCheck(packet.substring(0,18)).charAt(0) != packet.charAt(19)) { //TC3 parity check failed
    answer = new SpeedTorque();
    bitstream = bitstream.substring(start+1);
}
else {//TC1,TC2
    String speedStr = packet.substring(1,9);
    String torqueStr = packet.substring(10,18);
    double speed = bin8_to_dec(speedStr);
    double torque = bin8_to_dec(torqueStr);
    answer = new SpeedTorque(speed, torque);
    bitstream = bitstream.substring(end);
```

This is the end of the method. It displays the two main changes on the bitstream. In the else if since it is a test case of failure it only takes away the start delimiter. If it works then it takes away the whole packet. In case of failure it takes away only the start delimiter because the next iteration it will look for the start delimiter and ignore all the previous bits. This makes it good enough to just delete the start delimiter of the corrupted packet.

Send Sensor Data

```
 */
//Send Sensory Data
public String SendSensoryData (double Torque, double IR_Dist, double ULTRA_Dist) {
    String answer="";
    String T = dec_to_8bbin(Torque);
    String IR = dec_to_8bbin(IR_Dist);
    String Ultra = dec_to_8bbin(ULTRA_Dist);
```

This method receives three parameters, changes them to an 8 bit binary value.

```
if (((MinTorque <= Torque && Torque <= MaxTorque)

&& (Min_IR_Dist <= IR_Dist && IR_Dist <= Max_IR_Dist))

(Min_Ultra_Dist <= ULTRA_Dist && ULTRA_Dist<= Max_Ultra_Dist))
```

Here it checks that all the values are within range to be able to make a package. Otherwise it will ignore the values.