TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 61, Issue I, March, 2018

CONSIDERATIONS ON THE SERIAL PC - Arduino UNO R3 INTERACTION, IN JAVA, USING JDEVELOPER, FOR A 3R SERIAL ROBOT, BASED ON THE ARDULINK LIBRARY

Tiberiu Alexandru ANTAL

Abstract: *The paper presents a serial communication method in the Java programming language, based on the Ardulink library, using JDeveloper IDE, to program the position of 3R serial manipulator servomotors using the ARDUINO / GENUINO UNO microcontroller. The Java and the C code implementations under the JDeveloper and the Adruino IDEs are given, for the serial, bidirectional communication, between the PC and the GENUINO UNO microcontroller.*

Key words: *Ardulink, Arduino, Java, JDeveloper, Microcontroller, Serial.*

1. INTRODUCTION

The paper aims to deal with the problem of serial transfer between a PC, running a Java application, and a Arduino UNO R3 microcontroller[1] that controls the servomotors of a serial 3R robot (three rotations and one servo for the end effector servo). The serial robot is from a kit called "DIY Control Robot Arm kit for Arduino-Rollarm" and found at https://www.sunfounder.com/learn/category/DIY-Control-Robot-Arm-kit-for-Arduino-Rollarm.html. The GENUINO UNO is a Harvard architecture with its memory [1] organized as a 32 KB (with 0.5 KB occupied by the bootloader) flash memory, 2 KB of SRAM and 1 KB of EEPROM. In the case of the von Neumann architectures, the program data and code are stored in the same internal memory. This takes two sequential fetches, one for the code, and one for the data, to do most operations. The von Neumann architecture machines are more flexible for general computing by owning a common memory used entirely for large programs or large data sets. In addition, because the compiled programs contain both the data and the code, the data does not require any special processing (no need to move it to run after compiling) in order to make the code run. The Harvard architecture has two memories one for the data and one for the code; this allows parallel access to data and code. Flash memory is used to store code and initial data. Programs stored here can be run but the initial data stored here can't be changed by the running code. Flash memory is non-volatile, so the code stored there will persist when the system is powered off. The SRAM or Static Random Access Memory is the data memory, so it can be read and written by the running code in the Flash memory. It is used for:

- Static Data - for all the global and static variables in the code. Values of the initialized variables in the Flash memory will be copied to the SRAM by the system when the code starts;
- Heap – for dynamically allocated data items. The heap grows from the top of the static data area up as data items are allocated;
- Stack - maintained by the LIFO policy - is used for local variables and for maintaining a record of interrupts and function calls. The stack grows from the top of memory down towards the heap.

EEPROM is another non-volatile memory that can be read or written by the running code. However, this can be done only as byte data type.

2. SOME HARDWARE ASPECTS ON THE USB COMMUNICATION FOR THE ARDUINO UNO R3 (Mars Board)

The robot arm kit is using the SunFounder Mars Board (https://www.sunfounder.com/sunfounder-mars-board-compatible-for-arduino-uno-r3.html) an Arduiono Uno compatible board using the ATmega328P as processor and the same Optiboot bootloader as Uno (as a result it is programmable with the Arduino IDE). The board has a better hardware design compared to the original Arduino Uno as it is using the more stable and reliable FTDI232R for USB-to-serial and Type-C USB Port (supports reversible plug orientation). The Mars Board is connected to the servos by using the Extension Board and it is powered by two 18650 batteries. With the power switch on and plugged into the USB Port of the PC, the Arduino IDE will identify the Mars board as Arduino/Genuino Uno board a will communicate over the COMx serial port. 0.5 K of the Flash memory contains the bootloader. A bootloader is a program that runs in the microcontroller to be programmed. It receives the new program from the exterior via USB and writes that information to the microcontroller. The bootloader always runs at reset so it is the first program that runs on the microcontroller. This will communicate over USB with the Arduino IDE and if the IDE responds properly it will receive the data and code from the Arduino IDE and load it to the proper memories of the microcontroller in order to make it work. The USB port is used to program the microcontroller with the help of the Arduino IDE as well as to interact with the Java application running on the PC in order to exchange data with the microcontroller. Asynchronous communication over the USB port is trying to avoid transmitting a synchronization clock between the transmitter and the receiver. Each byte of data is transmitted in a packet or frame of bits. Frames are created by appending synchronization and parity bits to data. The frame begins with 1 start bit, continues with 5-9 data bits, followed by 0-1 party bits and 1-2 stop bits. The sender and the receiver clock frequency must be the same that is they must use the same fixed bit rate (clock periods per second) or bps (bits per second) so that communication can take place. The most popular baud is 9600 bps. The higher the baud rate goes, the faster data is transmitted however, for most microcontrollers, the speed won't exceed 115200 bps (over that speed communication starts to give errors as clocks are not providing the right sampling periods anymore). The protocol is highly configurable and one critical part is making sure that both the transmitter and the receiver on the serial bus use the same protocol and parameters.

3. SOFTWARE ASPECTS ON THE ARDUINO UNO BOARD SERIAL COMMUNICATION

The Arduino IDE originated from the open source Processing programming language that used an integrated development environment (PDE not IDE) to teach the fundamentals of computer programming in a visual context. Processing language was based on Java, but used a simplified syntax and a graphics user interface (GUI). Processing used the concept of *sketch* for programs and that of *sketchbook* for organizing programs in projects stored in a directory. The Processing application was based on the PApplet class where the programmer had to override two methods: *draw()* and *setup()*. The code in the *draw()* method runs continuously until the program is stopped while that in the *setup()* method runs once when the code is started. Processing was the one that influenced the creation of the Wiring development platform (containing a programming language, an IDE and a microcontroller). The Wiring IDE is a cross-platform application written in Java which is derived from PDE that includes a C/C++ library called "Wiring" to make input/output operations easier. Wiring programs are written

in a mixed dialect of C and C++. A minimal program requires only two functions, *setup()* and *loop()*. The *setup()* runs once at the start of a program and is used to define the initial settings for the code in the microcontroller. The *loop()* function is called repeatedly until the microcontroller is powered off or reset. The Arduino IDE uses the same functions as Wiring to setup and run an application. The communication between the microcontroller (also called board) and the PC is serial and is achieved in the Arduino programming language with the help of the `Serial` object. The `begin()` function of this object allows to define the serial communications parameters like the speed (300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200), the data, parity, and stop bits. By default these values are 8 data bits, no parity, one stop bit. A piece of code that wants to communicate over the serial port must define the speed as in the following code:

```
include <Servo.h>

  Servo servo1; //servo1-4 objects
  . . .
  Servo servo4; // 4

void setup()
{
//attach servo objects to pins
  servo1.attach(4); // servo1 on pin 4
  . . .
  servo4.attach(7); // servo4 > 7

//USB serial communication speed
  Serial.begin(9600);

//initialize the servo positions
  servo1.write(90); //  servo1 > home
. . .
  servo4.write(150);// servo4 > closed

}
```

The `setup()` must also contain any other initializations necessary to interact with the hardware connected to the board. The `Serial` object has the `available()` function to get the number of bytes available for reading from the serial port (stored in a 64 bytes receive buffer). This is useful if bytes are sent to the board over the USB port. In order to read these bytes the `read()` function of the same object must be used. For each call `read()` will read one byte from the receive buffer. The `write()` function is used to send a byte or a series of bytes over the serial port while the `print()` function is used to send a character. The following code tests if characters are present on the serial port. If no characters are in the buffer it will send back the positions of the four servos over the serial to the PC and the Java application will read it.

```
void loop()
{
  if (Serial.available() == <0) {
    servostate = servo1.read();
    Serial.write(servostate);
    Serial.write(',');
. . .
    servostate = servo4.read();
    Serial.write(servostate);
    Serial.write(255);
  }
// check if data is sent
  if (Serial.available() > 0) {
    int index = 0;
// wait for the buffer array to fill
    delay(500);
// number of sent characters
    int numChar = Serial.available();
    if (numChar > 24) {
      numChar = 24;
    }
// fill the buffer with the characters
    while (numChar--) {
      buffer[index++] = Serial.read();
    }
  . . .
}
```

4. SERIAL COMMUNICATION OVER USB IN JAVA BASED ON ARDULINK LIBRARY

Java programming language can be used for serial communication over the USB with the help of Ardulink library [2]-[5]. The following Java code is stored in the `RobotComV0` class. USB port communication is made using the `Link` class. This will register a `RawDataListener` to receive data from Arduino and then the connection to the board will be made with the `connect()` method. Data is sent to the serial port using the

writeSerial() method, while data received from the USB port is processes using the parseInput() method. This method implements an interface for the RawDataListener class and returns the string read from the serial port to the Java code and sets the values of the q1, q2, q3, q4 instance variables as shown in the following piece of code:

```
import java.util.List;
import org.zu.ardulink.Link;
import
org.zu.ardulink.RawDataListener;

public class RobotComV0 implements
RawDataListener {
  final Link link =
Link.getDefaultInstance();
  StringBuilder build;
  int q1, q2, q3, q4;

public RobotComV0() {
. . .
  boolean connected =
link.connect(port, 9600);
. . .
    }

public void program() throws
InterruptedException {
. . .
  link.writeSerial("q450");
. . .
  link.disconnect();
}

public static void main(String[] args)
{
  RobotComV0 ar = new RobotComV0();
  try {
    ar.program();
  } catch (InterruptedException e) {
  }
}
public void parseInput(String string,
int nBytes, int[] message) {
  build = new StringBuilder(nBytes +
1);
  for (int i = 0; i < numBytes; i++) {
    build.append((char) message[i]);
  }
  q1 = (int) message[0];
  q2 = (int) message[2];
  q3 = (int) message[4];
  q4 = (int) message[6];
  }
}
```

REFERENCES

[1] https://learn.adafruit.com/memories-of-an-arduino/arduino-memory-architecture

[2] ANTAL, Tiberiu Alexandru. *Arduino Leonardo programming under Windows, in Java, from JDeveloper using Ardulink. ACTA TECHNICA NAPOCENSIS* - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, Nr. 60, Vol. 1, 2017, p.7-2, ISSN 1221-5872.

[3] ANTAL, T. A., *Elemente de Java cu Jdeveloper - îndrumător de laborator*, Editura UTPRES, 2013, p.150, ISBN: 978-973-662-827-6.

[4] ANTAL, T. A., *Java - Inițiere - îndrumător de laborator*, Editura UTPRES, 2013, p. 246, ISBN: 978-973-662-832-0.

[5] ANTAL, Tiberiu Alexandru, CHELARU, Julieta Daniela. *A multithreaded java client-server model for robot interaction*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, Nr. 60, Vol. 3, 2017, p.7-2, ISSN 1221-5872.

CONSIDERAŢII PRIVIND INTERACŢIUNEA SERIALĂ DINTRE PC ŞI MICROCONTROLERUL Arduino UNO R3, ÎN JAVA, FOLOSIND JDEVELOPER, PENTRU UN ROBOT SERIAL 3R UTILIZÂND BIBLIOTECA ARDULINK.

Rezumat: Lucrarea prezintă o modalitate de comunicare serială, în limbajul de programare Java, bazată pe biblioteca Ardulink, folosind mediul JDeveloper, pentru programarea poziţiei servomotoarelor unui manipulator serial 3R care foloseşte microcontrolerul Arduino UNO R3. Codul dă implementarea în C sub mediul Adruino şi cel Java sub mediul JDeveloper pentru comunicaţia serială bidirecţională dintre calculatorul pe care rulează Java şi microcontrolerul Arduino UNO R3.

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@bavaria.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.