**An-Najah National University**

**Faculty of Engineering**

**Computer Engineering Department**

**Graduation Project (Final Report).**

**Instructor**: Dr. Ala’a Al-Masry.

Dr. Hanal Abu-Zant.

**Prepared By:**

Abir T. Sabra.

Iman A. Shtayeh.

Tuesday

May24, 2011

Content…

* Introduction.
* Objectives.
* Protocol (I2C)
* Technologies.

-RFID technology.

* Main components (H.W).

-Main control station

-Trains.

-Railway.

* Software.

-Overview about communication.

-Master.

-Slave.

-C# interface.

* Problems.
* References.

**INTRODUCTION.**

The idea of our project is to apply an idea of controlling railway system. We have used a module of rail and trains and we have achieved our goals, we wish applying this idea in real life in our country Palestine.

**OBJECTIVES:**

Design a network-controlled railway system.

But, allow human interrupt and monitoring.

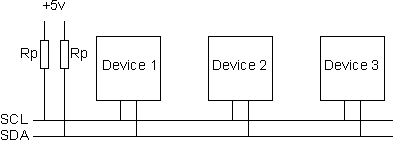
**PROTOCOL**.

As a protocol for communication between microcontrollers on trains and main microcontroller we used I2C protocol for many reasons.

Firstly I would like to give some background about I2C.

So, I2C.

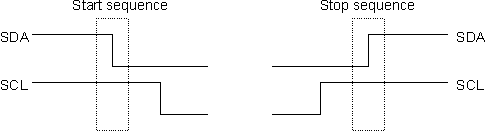
**The physical I2C bus**  
This is just two wires, called SCL and SDA. SCL is the clock line. It is used to synchronize all data transfers over the I2C bus. SDA is the data line. The SCL & SDA lines are connected to all devices on the I2C bus. There needs to be a third wire which is just the ground or 0 volts. There may also be a 5volt wire is power is being distributed to the devices. Both SCL and SDA lines are "open drain" drivers. What this means is that the chip can drive its output low, but it cannot drive it high. For the line to be able to go high you must provide pull-up resistors to the 5v supply. There should be a resistor from the SCL line to the 5v line and another from the SDA line to the 5v line. You only need one set of pull-up resistors for the whole I2C bus, not for each device, as illustrated below:



The value of the resistors is not critical. I have seen anything from 1k8 (1800 ohms) to 47k (47000 ohms) used. 1k8, 4k7 and 10k are common values, but anything in this range should work OK. I recommend 1k8 as this gives you the best performance. If the resistors are missing, the SCL and SDA lines will always be low - nearly 0 volts - and the I2C bus will not work.

**Masters and Slaves**  
The devices on the I2C bus are either masters or slaves. The master is always the device that drives the SCL clock line. The slaves are the devices that respond to the master. A slave cannot initiate a transfer over the I2C bus, only a master can do that. There can be, and usually are, multiple slaves on the I2C bus, however there is normally only one master. It is possible to have multiple masters, but it is unusual and not covered here. On your robot, the master will be your controller and the slaves will be our modules such as the SRF08 or CMPS03. Slaves will never initiate a transfer. Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

**The I2C Physical Protocol**  
When the master (your controller) wishes to talk to a slave (our CMPS03 for example) it begins by issuing a start sequence on the I2C bus. A start sequence is one of two special sequences defined for the I2C bus, the other being the stop sequence. The start sequence and stop sequence are special in that these are the only places where the SDA (data line) is allowed to change while the SCL (clock line) is high. When data is being transferred, SDA must remain stable and not change whilst SCL is high. The start and stop sequences mark the beginning and end of a transaction with the slave device.



Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is then pulsed high, then low. Remember that the chip cannot really drive the line high, it simply "lets go" of it and the resistor actually pulls it high. For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data. If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high then it is indicating it cannot accept any further data and the master should terminate the transfer by sending a stop sequence.

http://www.robot-electronics.co.uk/images/i2cc.GIF

**How fast?**  
The standard clock (SCL) speed for I2C up to 100KHz. Philips do define faster speeds: Fast mode, which is up to 400KHz and High Speed mode which is up to 3.4MHz. All of our modules are designed to work at up to 100 KHz. We have tested our modules up to 1MHz but this needs a small delay of a few uS between each byte transferred. In practical robots, we have never had any need to use high SCL speeds. Keep SCL at or below 100 KHz and then forget about it.

**I2C Device Addressing**  
All I2C addresses are either 7 bits or 10 bits. The use of 10 bit addresses is rare and is not covered here. All of our modules and the common chips you will use will have 7 bit addresses. This means that you can have up to 128 devices on the I2C bus, since a 7bit number can be from 0 to 127. When sending out the 7 bit address, we still always send 8 bits. The extra bit is used to inform the slave if the master is writing to it or reading from it. If the bit is zero are master is writing to the slave. If the bit is 1 the master is reading from the slave. The 7 bit address is placed in the upper 7 bits of the byte and the Read/Write (R/W) bit is in the LSB (Least Significant Bit).

http://www.robot-electronics.co.uk/images/i2cd.GIF

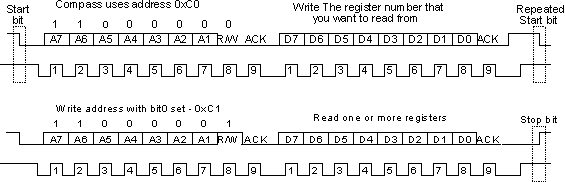
The placement of the 7 bit address in the upper 7 bits of the byte is a source of confusion for the newcomer. It means that to write to address 21, you must actually send out 42 which is 21 moved over by 1 bit. It is probably easier to think of the I2C bus addresses as 8 bit addresses, with even addresses as write only, and the odd addresses as the read address for the same device. To take our CMPS03 for example, this is at address 0xC0 ($C0). You would uses 0xC0 to write to the CMPS03 and 0xC1 to read from it. So the read/write bit just makes it an odd/even address.

**The I2C Software Protocol**The first thing that will happen is that the master will send out a start sequence. This will alert all the slave devices on the bus that a transaction is starting and they should listen in incase it is for them. Next the master will send out the device address. The slave that matches this address will continue with the transaction, any others will ignore the rest of this transaction and wait for the next. Having addressed the slave device the master must now send out the internal location or register number inside the slave that it wishes to write to or read from. This number is obviously dependent on what the slave actually is and how many internal registers it has. Some very simple devices do not have any, but most do, including all of our modules. Our CMPS03 has 16 locations numbered 0-15. The SRF08 has 36. Having sent the I2C address and the internal register address  the master can now send the data byte (or bytes, it doesn't have to be just one). The master can continue to send data bytes to the slave and these will normally be placed in the following registers because the slave will automatically increment the internal register address after each byte. When the master has finished writing all data to the slave, it sends a stop sequence which completes the transaction. So to write to a slave device:   
1. Send a start sequence  
2. Send the I2C address of the slave with the R/W bit low (even address)  
3. Send the internal register number you want to write to  
4. Send the data byte  
5. [Optionally, send any further data bytes]  
6. Send the stop sequence.

As an example, you have an SRF08 at the factory default address of 0xE0. To start the SRF08 ranging you would write 0x51 to the command register at 0x00 like this:  
1. Send a start sequence  
2. Send 0xE0 (I2C address of the SRF08 with the R/W bit low (even address)  
3. Send 0x00 (Internal address of the command register)  
4. Send 0x51 (The command to start the SRF08 ranging)  
5. Send the stop sequence.

**Reading from the Slave**  
This is a little more complicated - but not too much more. Before reading data from the slave device, you must tell it which of its internal addresses you want to read. So a read of the slave actually starts off by writing to it. This is the same as when you want to write to it: You send the start sequence, the I2C address of the slave with the R/W bit low (even address) and the internal register number you want to write to. Now you send another start sequence (sometimes called a restart) and the I2C address again - this time with the read bit set. You then read as many data bytes as you wish and terminate the transaction with a stop sequence. So to read the compass bearing as a byte from the CMPS03 module:  
1. Send a start sequence  
2. Send 0xC0 ( I2C address of the CMPS03 with the R/W bit low (even address)  
3. Send 0x01 (Internal address of the bearing register)  
4. Send a start sequence again (repeated start)  
5. Send 0xC1 ( I2C address of the CMPS03 with the R/W bit high (odd address)  
6. Read data byte from CMPS03  
7. Send the stop sequence.

The bit sequence will look like this:



Example of code (PICC). From master to slave.

i2c\_start()

delay\_ms(5)

i2c\_write(0xA0) // address of slave.

delay\_ms(10)

i2c\_write(0x43); // send char C

delay\_ms(20)

i2c\_stop()

delay\_ms(50)

**TECHNOLOGIES.**

As a technology we use RFID technology because we need it for addressing, so in each train we have RFID reader. And RFID tags on railway.

So firstly I would like to give you some background about RFID technology.

**RFID technology**

**Radio-frequency identification** (**RFID**) is a [technology](http://en.wikipedia.org/wiki/Technology) that uses communication through the use of [radio waves](http://en.wikipedia.org/wiki/Radio_waves) to exchange data between a reader and an electronic tag attached to an object, for the purpose of identification and tracking.

It is possible in the near future, RFID technology will continue to proliferate in our daily lives the way that bar code technology did over the forty years leading up to the turn of the 21st century bringing unobtrusive but remarkable changes when it was new.

RFID makes it possible to give each product in a grocery store its own unique identifying number, to provide assets, people, work in process, medical devices etc. all with individual unique identifiers - like the license plate on a car but for every item in the world. This is a vast improvement over paper and pencil tracking or bar code tracking that has been used since the 1970s. With bar codes, it is only possible to identify the brand and type of package in a grocery store, for instance. Furthermore, passive RFID tags (those without a battery) can be read if passed within close enough proximity to an RFID reader. It is not necessary to "show" the tag to the reader device, as with a bar code. In other words it does not require line of sight to "see" an RFID tag, the tag can be read inside a case, carton, box or other container, and unlike barcodes RFID tags can be read hundreds at a time. Bar codes can only read one at a time.

Some RFID tags can be read from several meters away and beyond the line of sight of the reader. The application of [bulk reading](http://en.wikipedia.org/wiki/Bulk_reading) enables an almost-parallel reading of tags.

Radio-frequency identification involves the hardware known as interrogators (also known as readers), and tags (also known as labels), as well as RFID software or RFID middleware.

Most RFID tags contain at least two parts: one is an [integrated circuit](http://en.wikipedia.org/wiki/Integrated_circuit) for storing and processing information, [modulating](http://en.wikipedia.org/wiki/Modulation) and [demodulating](http://en.wikipedia.org/wiki/Demodulation) a [radio-frequency](http://en.wikipedia.org/wiki/Radio-frequency) (RF) signal, and other specialized functions; the other is an [antenna](http://en.wikipedia.org/wiki/Antenna_%28radio%29) for receiving and transmitting the signal.

RFID can be passive (using no [battery](http://en.wikipedia.org/wiki/Battery_%28electricity%29)), active (with an on-board battery that always broadcasts or beacons its signal) or battery assisted passive (BAP) which has a small battery on board that is activated when in the presence of an RFID reader. Passive tags in 2011 start at $ .05 each and for special tags meant to be mounted on metal, or withstand gamma sterilization go up to $5. Active tags for tracking containers, medical assets, or monitoring environmental conditions in data centers all start at $50 and can go up over $100 each. BAP tags are in the $3–10 range and also have sensor capability like temperature and humidity.

The term RFID refers to the technology. The tags should properly be called "RFID tags" not "RFIDs".

**MAIN COMPONENTS(H.W)**

Our project consists of:

1. Main control station.

2. n- trains.

3. Railway (communication medium + power).

I will move to each one in details.

**Main control station.**

In main control station we have.

Main microcontroller (central unit)

Main computer for human interrupt and monitoring.

**H.W**:

1- PIC18f4620

2- H-bridge (L298)

3- Dc motor.

4- Inductors to isolate circuits (for back driving current from motor.)

5- Regulators (L7806C).

As we say main microcontroller work as central unit (bridge) between trains themselves and connect them with main pc for monitoring.

Main microcontroller has multiple connections.

* Connection to main pc.
* Connection to each train (I2C protocol).
* Connection to Railway direction controller (in our case we consider mechanical motor).

**TRAINS.**

Each train has these components:

1. Microcontroller(PIC18f4620)
2. RFID reader(ID-12)
3. H-bridge chip (L298).
4. Dc motor.
5. Inductors to isolate circuits (for back driving current from motor.)
6. Regulators (L7806C).

Microcontroller on train has multiple connections:

- Connection to master (I2C).

- Connection to RFID reader.

- Connection to local motor through H-bridge.

**RAILWAY**.

We have used the railway as a closed medium for train’s movement.

So to address position on railway we need to arrange the RFID tags on railway in some order as shown below.

Also on railway we have another branch for maintenance or stopping.

**SOFTWARE**.

As software for our project we have software for main microcontroller (PICC) and software for each train and another one for monitoring and controlling from human (C# application).

So software can be described as:

* **One** master represents main M.C.
* **Three** slaves each one represents microcontroller for train.
* **I2C** as protocol for communication.
* **C# application** for human main controlling.

**Overview about communication.**

**Master.**

* Master keeps requesting data from each slave through a specific order and gives slave period of time to get data from.
* Master sends each new position of trains to main PC (serial connection).
* Master sends each new data about any train to slaves.
* Master send command to any train through our interface on main PC.

In master we used a serial interrupt to get command from main pc directly as soon as data is available on serial port.

List of functions used in master code.

* openMainMotor() // move motor to change railway direction
* releaseMainMotor()
* serialInt() // get data from main pc through serial
* sendCmdToSlave() // send command received from pc to slave with specific address.
* sendReadyToSlave() // to move slave after change rail direction
* sendToAll() // send new position of any slave to all slaves
* requestSlave () // request position from slave.
* sendToserial ()// send each new position of train to pc.
* Main()

Flow chart of master attached in CD…

**Slave.**

* Give data to master when request it.

- if new data available, then give master the new position.

- if no new data just give master ‘keep a live’.

* Get data from master which may be :

- Command from main station, execute command.

- Position of other train, store the position and release the previous position.

In slave we used two interrupts, one to get available data from RFID reader and the other for I2C data from master.

List of functions used in master code.

* IncrementPosition() // increment position of train
* Stop () // stop train.
* Movebackword (int8 temp1) // reverse train direction.
* nextBusy() // to check if next tag is busy
* nextStopStation() // check if next tag is on stop station
* CurrentStopStation () // if current position of train is stop station.
* checkTags(char tg1,char tg2) //return the position number of tag
* checkSpeed()
* RDA\_isr () // interrupt function to get RFID data from reader.
* SSPintFunction () // interrupt function to get data using I2C protocol from master.
* Main()

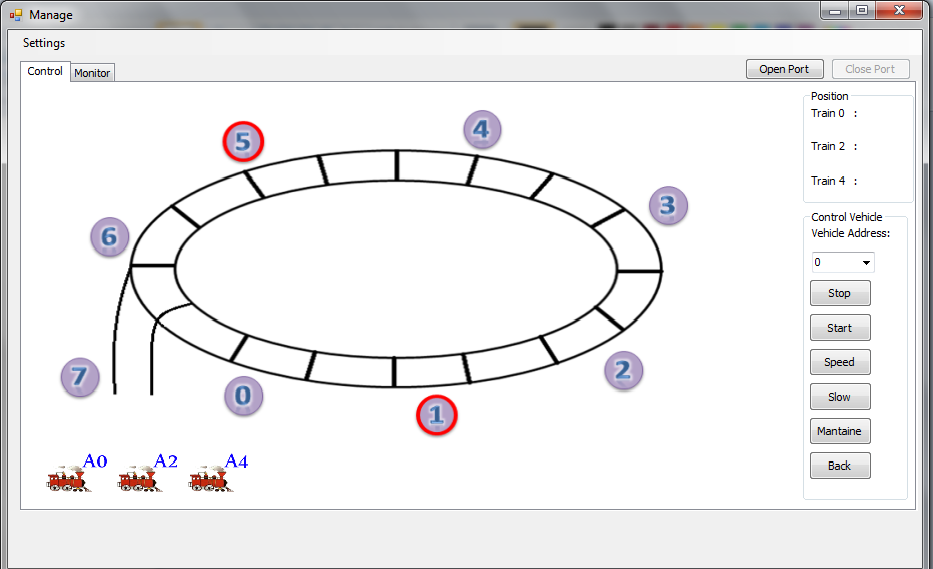
Flow chart of slave attached in CD…

**C# application (interface).**

Finally we have a c# interface that enable human controlling and monitoring.

By this interface Human controller could:

* Stop, start, slow, speed any train on railway.
* Also enters any train to maintenance through secondary branch and return it back when needed.
* Constantly monitors trains on railway.



**PROBLEMS**.

In our project we have many problems some of them in noise, H.W our S.W

So I will list here some of our problems.

1. Does no guarantee getting the data from reader or serial as possible it is available (S.W problem). We solve this problem by using interrupt (RD interrupt to get data from RFID reader directly when data available and SSP interrupt to get data from master and solution was effective).
2. Back driving current from motors(H.W problem) as we know motors have shared GND with main circuit, so motors may cause noise to circuit when return some current throw GND . We solve this problem by using inductors between two GNDs (main cct and H-bridge cct) and we ensure to use capacitors as shortest as possible to each VCC connection. (Another solution is to use auto coupler as good isolation device).
3. Have a good medium for communication. As we used I2C protocol for communication we need 3 wires connection between master and slaves and during the movement of trains the connection may be cut off because the module we have used was not prepared to help us. We decided to move the tags across rail instead of trains but the idea still as it the software and the hardware (mechanical problem).

**REFERANCES**.

* PIC18f4620 datasheet.
* ID-12 datasheet.
* L298 datasheet.
* L78xx - L78xxC datasheet.