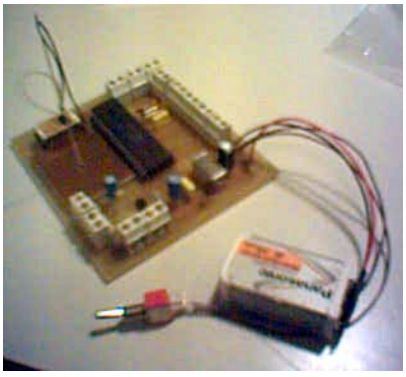
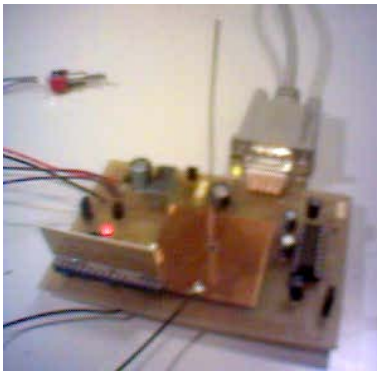
Aalborg University Copenhagen

Medialogy Cand. Sc. – 9th semester project

Project Report

A simple practical approach to wireless distributed data acquisition


18/01/2006

Group number: K986

Student:
Smilen Dimitrov

Project Supervisor:
Stefania Serafin

AALBORG UNIVERSITET

Abstract / synopsis

During the winter semester of 2005, the author of this report assisted with the development of a Medialogy research project, dealing with user's responses to auditory stimuli in a VR environment, conducted by Rolf Nordahl[1]. A specific phase in this research project demanded acquisition of interaction input via wireless sensing, such that the interaction input signal can be processed real-time in Cycling 74's Max/MSP software package for Windows. The investigation within this subject shows that a system of two hardware devices (transmitter and receiver) of prototype quality can be easily implemented, with over-the-shelf and relatively inexpensive electronic parts. The system design revolves around usage of ready-made, coupled radio transmitters and receivers, and usage of a microcontroller with multi-channel A/D conversion capability. A classic microcontroller of the Microchip brand can be programmed via a hardware device (a programmer that connects to a Windows PC), which again can be inexpensively built from over-the-shelf parts - and can be driven by several freeware programs. Eventually, the system can be connected to a serial port on any PC, and the sensed values can be consumed through a serial driver. This report attempts to identify and document the key hardware and software issues in development and use of this wireless data acquisition system, which despite its prototype-grade quality, is still relatively easy and inexpensive to implement – which might motivate its use within student projects that deal with human computer interaction.

Acknowledgments

I would like to thank my parents, Vladimir and Todorka Dimitrovi, for their support during the course of my studies; Christian Poulsen and the rest of BB&S, Copenhagen, for he first practical introduction to hardware programming and communication; project supervisor Stefania Serafin, as well as Rolf Nordahl, for their kind assistance with this report; and all those that helped me through the paper-and-pencil bachelor days at University "Sts. Cyril & Methodius", Skopje, Macedonia.

Table of Contents

1 Introduction 1

2 Analysis 4
2.1 The Kroonde Gamma – a possible solution 4
2.2 The search for other devices on the market, and the classification terminology problem 6
2.3 The terminology problem and the academic perspectives 7
2.4 The shift in focus towards a custom built prototype 9
2.5 Assumptions, risk factors and quality delimitation for development of a custom built prototype 10
2.6 Main problem formulation and delimitations 12

3 Classification of the Teleo device as a data acquisition system 14
3.1 A textbook perspective on data acquisition systems 20

4 Microcontroller as data acquisition hardware 24
4.1 Introduction to microcontrollers 24
4.2 The choice of PIC 16F74 and the corresponding development tools 28
4.3 Hardware programming process of PIC 16F74 via parallel port 33
4.4 Hardware programmer circuit 41
4.5 Microcontroller software – the data acquisition program 51

5 Serial communication and RS-232 62
5.1 Introduction to serial communication 62
5.2 Asynchronous serial communication – PC serial 68
5.3 RS-232 serial - voltage levels, connectors and issues 73

6 System Design and Implementation 80
6.1 An example of a wired multichannel DAQ system 82
6.2 An example of wireless multichannel DAQ system 88
6.2.1 RF devices - introduction and issues 88
6.2.1.1 RF Solutions RTFQ1 and RRFQ1 – transmitter and receiver modules 90
6.2.1.2 Mode of operation of the wireless link 95
6.2.1.3 Decoupling capacitors on the power supply 98
6.2.1.4 Antenna ground plane 99
6.2.2 Transmitter circuit 104

6.2.3 Receiver circuit 106
6.3 Consuming the received digitized values 108
6.3.1 Writing a serial decoder for Max/MSP 108

7 Usage and performance 112

8 Conclusion 114

9 Appendix 120
Appendix A. Schematic of the classic Tait hardware programmer 120
Appendix B. BASIC code listing for microcontroller data acquisition program 120

List of references 122

1 Introduction

The development of a wireless data acquisition system was initiated by a research project conducted at Aalborg University Copenhagen during the winter semester of 2005, dealing with user's responses to auditory stimuli in a VR environment, conducted by Rolf Nordahl[1]. A specific phase in this research project demanded acquisition of interaction input via wireless sensing, such that the interaction input signal can be processed real-time in Cycling 74's Max/MSP software package for Windows. The system that could acquire data in this way would have been implemented in a footstep controller, used in this research project. Here is a basic description of the footstep controller in its original, wired form: "

The controller is made of a pair of open sandals of size 43, which can be comfortably worn by subjects with smaller feet. In each sandal, a force sensitive resistor (FSR) is placed in each heel. The FSR detects the pressure of the corresponding foot while in contact with the floor [...] The sensors are taped to the heel of the sandals, and wired to a microprocessor developed by Making Things. The value of the voltage is converted into a continuous stream of numbers which is read by the Max/MSP platform. The company Making things has developed a set of software (ad-hoc external) that supports this procedure.[1]"

The description helps with charting a simple conceptual block diagram of the footstep controller:

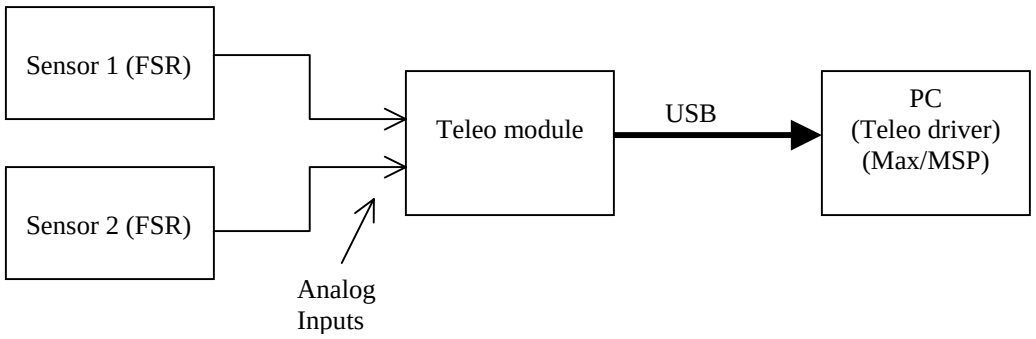


Figure 1. A conceptual block diagram of the footstep controller and PC

In essence, the design of the researchers answers a human-computer interaction (HCI) design issue – the whole system is intended to provide an additional channel of user interaction, within the context of the software environment used in the research project. The intent of the footstep controller is to provide a triggering mechanism for a playback of a footstep sound, whenever the test subject performs an actual step. The problem identified in the research project was, that the FSR sensors need to be attached by wire to the Teleo device, which then again needs to be connected to a PC via an USB cable – which can be also envisioned on Figure 1. This is an additional set of wires that a test subject is burdened with, in addition to the wires attached to the head mount device – as the research was conducted on test subjects in a VR environment. This was identified as a problem, since it limited the freedom of movement in test subjects.

As such, the researchers are faced with, in essence, a usability issue – obviously a test subject which is hindered in movement by cable, is less likely to experience ease of interaction with the system, at least in the sense of interacting via footsteps. Thus the researchers looked into a possibility of a wireless solution for the footstep controller – a solution where the information for a footstep detected by the FSR is sent wirelessly to the PC that renders the footstep sound. With such a design, effectively, the researchers solved the HCI aspect of their problem – as long as any eventual hardware connected directly to the FSR sensors can be worn or carried by the test subject. At this stage, the initial problem formulation that the researchers set forth could be written as “How can we make the footstep controller wireless”, assuming a HCI perspective on the problem.

Certain specific problems were encountered in answering such a problem formulation, discussed further in this report, which shifted the focus from the original HCI perspective towards more low-level, hardware and software development perspective. These problems are partially stimulated by the different academic focus on interaction processing from the perspective of HCI and from the perspective of hardware – in essence, it is not always obvious that the techniques of data acquisition in electronic measurements actually represent the hardware and software backbone of HCI. In that sense, the above initial problem formulation is answered by providing an implementation of a custom system for wireless data acquisition, of initial prototype quality. As such, although it answers the initial problem formulation, it also interferes with other quality demands that the researchers might have from the system. Hence,

the drift in focus from a HCI problem to a hardware/software prototyping problem results with a system that may not be ideal for the research project; however, that focus also makes the system development touch upon the very academic issues that caused the problems in the first place, as some theory needs to be introduced before implementation can start – revealing a possible academic value of this system prototype as an exercise for students of HCI.

Bearing that in mind, this report will summarize all the key issues involved with building of the wireless data acquisition system, referring occasionally to some basic theoretical concepts.

2 Analysis

2.1 The Kroonde Gamma – a possible solution

The initial, wired design of the footstep controller has been laid out in the Introduction, and visualized on Figure 1. Eventually, the Teleo device (and drivers) is the central hardware (and software) unit that provides the connection between the FSR sensors and the software environment used in the research project. Thus, the most sensible thing seems to be replacing the Teleo platform with a similar one that is already wireless. Actually, the department of Medialogy is already in a possession such a device: The Kroonde Gamma[18] produced by the French company Le Kitchen:

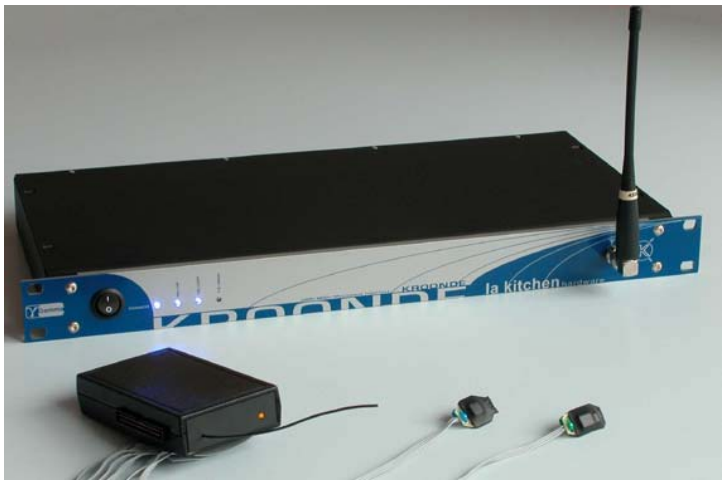


Figure 2. The Kroonde Gamma - a receiver (blue 1U rack), portable transmitter (black box), and sensors (Ref. [18])

Actually, this device perfectly fits into the design of the researchers. With specifications that guarantee:

- portable transmitter (of dimensions 117 x 75 x 25 mm, which means it is also wearable by the test subject, say in a pocket), which allows sensors to be connected to it;
- a 9V battery operation with a run time of 10 hours
- up to 16 sensors at a time with sampling interval of 9 ms (sampling frequency 111 Hz) and 10 bit resolution sampling per channel
- UDP Ethernet connectivity with a PC,

- Dedicated drivers for use with Max/MSP (Cycling 74 actually advertises this product as well)

the device seems perfect for use in the research project, and would probably answer the wireless design the researchers made, without problems – especially since a commercially available system like this can actually be trusted to deliver what it claims in the specifications. The only possible technical drawback that might be identified with the Kroonde is that it is based on UDP[19] (User Datagram Protocol) networking, which is a connectionless protocol which does not guarantee data delivery and is not reliable; in contrast with TCP (Transmission Control Protocol) which is connection oriented protocol, and is reliable [however, that should not be a problem, since reliability of delivery can be implemented into UDP as well, as it is probably the case with a commercially abailable device]. With this the device, the researchers' wireless design can be rendered on the following diagram:

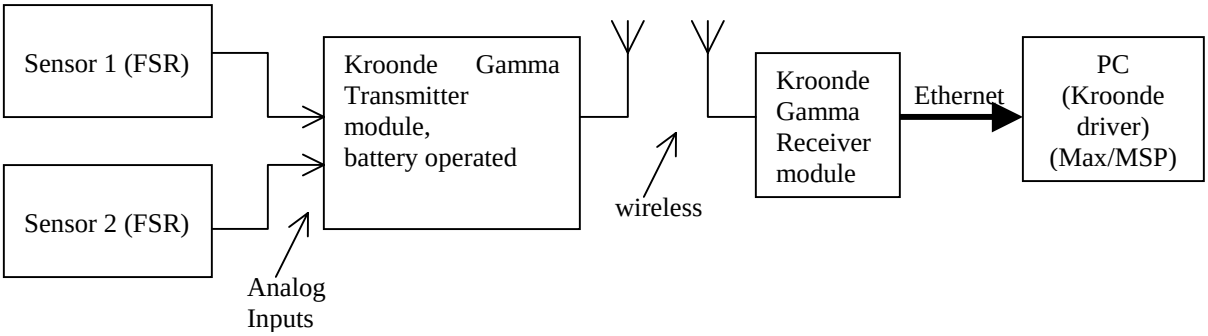


Figure 3. A conceptual block diagram of a wireless footstep controller based on Kroonde Gamma and PC

The utilization of the Kroonde Gamma system would answer to the researchers demands for greater freedom of movement of the test subject, since the test subject can bear the transmitter (which can be worn in a pocket, for instance) as well as the sensors; the battery operation and the radio data transmission would ensure that no additional cabling is necessary between the test subject and the rest of the footstep processing hardware. Thus, the researchers solve their HCI issues by having their own wireless HCI design implemented using the Kroonde Gamma system. However, although this device was present on the department premises, it was unfortunately unavailable for the research project. With a price tag of over 1000 US$ being a possible obstacle to acquiring yet another of these devices for immediate use in the research project – another solution is necessary. Thus, any inquiries into the

Kroonde Gamma were abandoned, and initial focus was dedicated to a deeper analysis of the Teleo as a device.

2.2 The search for other devices on the market, and the classification terminology problem

In retrospect, that was not the wisest move to make. If the Kroonde Gamma is unavailable as a system, then the next step would be an inquiry into similar devices available on the market – and in such a search, the proper classification of the device is of essence. The Kroonde Gamma is advertised as a "wireless, high speed, high precision data-acquisition system dedicated to real time applications", where the term "wireless data acquisition system" is essentially the classification we are looking for. The advertising of the Teleo device uses wording that does not include the term "data acquisition system" as a specification, which made searches into a "wireless" version of it fruitless. However, in any case, googling after such devices is a bit troublesome: "portable" does not always mean "wireless" (it can mean portable in the sense that it is used together with a laptop); however, the right type of devices can be tracked down, such as the RadioPod by IPC Systems – and it seems there are plenty of those:

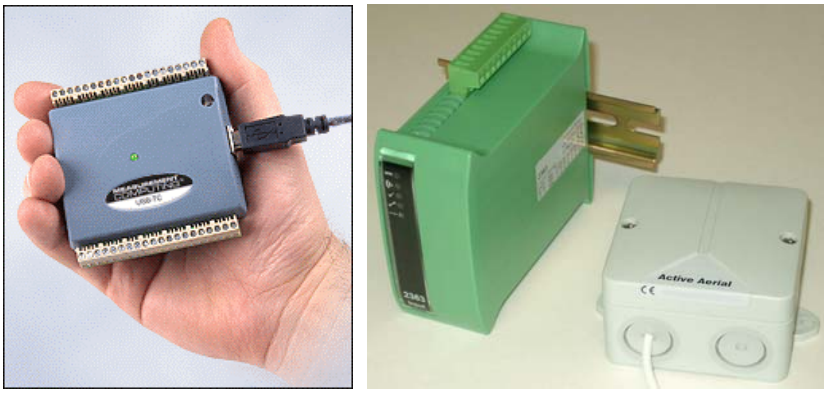


Figure 4. Left: portable wired USB module by Measurement Computing (Ref [21]), right: wireless portable RadioPod by IPC Systems (Ref. [20])

Primarily, the problem with such devices would be that no default connectivity with Max/MSP is available; thus, in spite of the costs involved, a driver for Max/MSP would most likely have to be developed anyways. Then, there is the question of determining the right specifications for the device – if we compare the Teleo and the Kroonde Gamma in terms of sampling rate and resolution, we can see that they both

offer maximum of 10 bits resolution, and a sampling rate of max 1 KHz for the Teleo (200 Hz for the Kroonde). If we consider that the primary usage of these devices is HCI interaction processing (due to their native connectivity with Max/MSP and the market segment that it covers), we should remember that data acquisition is a general field of electric measurements – specifications such as 16 bit, 44 KHz (which would relate to sound data acquisition) would not be surprising, especially since some commercial data acquisition systems go into the MHz sample rate range. If we go by the Kroonde and the Teleo, that is an overhead we do not need for interaction processing. Finally, in the author's own (limited) experience, some companies will avoid selling a single piece of hardware as it does not constitute profit for them – thus imposing a bulk sale on the customer.

2.3 The terminology problem and the academic perspectives

However, as noted before, due to the failure to properly identify the classification of the needed system, further investigation into available devices on the market was not undertaken. This is maybe the right point to recognize that the problematic of this report goes beyond a HCI design problem – actually it looks for a hardware backbone for implementation of a HCI design. The researchers in the VR motion project have identified a usability problem with their wired design of a footstep controller (which obstructs the test subject's motion); and the researchers actually do solve it by coming up with a wireless design of the same. However, it is not the researchers' design per se which is the problem here; it is (parts of) the hardware implementation – and thus, it is a problem closer to electronic engineering, than a problem of HCI. In essence, this hardware implementation, if it is to follow the steps of the Kroonde and the Teleo, is actually a generic device – multiple types of sensors can be utilized with it. Hence, a generic device, in addition to being applicable in the footstep controller, has application in other projects where interaction is designed on the sensor level; the context of usage is determined by the types of sensors used and by the intent of the HCI designer, but its operation from a hardware perspective is still the same.

This can also be supported by the academic treatment of these issues, in the sense of course syllabus topics. Although HCI is a relatively new area, we could

identify an approach we could call “traditional” HCI. Most issues discussed in traditional HCI courses are related to the context of a user operating a desktop PC via standard peripherals (mouse, keyboard, monitor). Whether we talk about a website navigation or a full blown graphic user interface (GUI) of a program, HCI solutions are provided that are centered around the user experience, and implemented using a high-level software package; the issue of sensors, data acquisition and drivers is taken for granted most of the time. In fact, traditional HCI is concerned with evaluating the user experience of desktop PC operation, just as much as providing a high-level technical software solution for it – consider for instance some of the online course notes (mostly undergraduate) posted in [22]. The issue of sensors and data acquisition is not too common in traditional HCI, and is pretty much kept as an separate area, indicating the closer focus to hardware/software issues than traditional HCI usability issues. However, new directions emerge in HCI, which begin to focus on interaction methods other than the traditional ones, and so require a closer connection with hardware and software issues – consider courses such as "Input/Data Acquisition System Design for Human Computer Interfacing" [23] or "Real-Time Computing For Human Computer Interfacing" [24].

Having this in mind, we could postulate that the original basic problem formulation, set forth by the VR research project, which might be devised as “How can we make the footstep controller wireless” and reflects a HCI perspective – in this report would boil down to something like “Which portable wireless data acquisition system can we implement to make the footstep controller wireless”, reflecting a more technical perspective on the problem. Again, the failure to properly identify the “data acquisition” terminology in the research phase for this project, meant that the technical problem formulation could not be answered by investigating the market for a wireless data acquisition system, which would act as an alternative to the Kroonde.

In addition, the failure to correctly define the terminology and classify the type of a device, made the research into previous academic work in hardware implementation of data acquisition devices in the context of HCI impossible; most of the sources used in the development of this project are component datasheets and various Internet resources related to electronics. That means that the hardware development research starts by compiling different resources – essentially from scratch – rather than building up on existing works. That failure brought about Chapter 3: “Classification of the Teleo device as a data acquisition system” in this

report, where the area of data acquisition is shortly presented from the perspective of electronic measurements and instrumentation, a distinct area within electronic engineering; the Teleo is also debated from this perspective. The failure to establish the notion of data acquisition in electronic measurements as the hardware backbone in HCI, led to a much more uncertain research. However, it also puts the focus on the academic value of the hardware prototype described in this report – its simplicity allows it to be used a student exercise, whose goal would be to illustrate this very link between data acquisition and HCI.

2.4 The shift in focus towards a custom built prototype

There is however, another alternative to purchasing a data acquisition hardware – and that is to build a custom hardware and software prototype of a wireless data acquisition system. Thanks to the cooperation with Engineering College in Copenhagen (IHK), the department of Medialogy at AUC has access to a student electronic component store, where low cost electronic components are available to students, at (from the students perspective) no cost. Facilities are also available for etching of printed circuit boards (PCB), as well as standard laboratory equipment (power sources, oscilloscopes etc.). This possibility stimulated the inquiry into whether it is possible to custom build a wireless data acquisition system, using the available components in the store. The no-cost perspective of the available store components almost automatically implied the challenge to build the system *only* from components available at the store – in which case (at least on paper) this project would manage to keep the additional hardware expences at zero. It is at this point obvious that the focus of the problematic shifts drastically from the original HCI perspective into a more electronic hardware domain, resulting in a problem formulation like "How can we implement a custom built hardware and software prototype of a wireless data acquisition system".

As a starting point, an eventual custom prototype for a wireless data acquisition system would aim to clone the three essential properties that make the Kroonde desirable for use in the research project:

- Wireless transfer of several (multichannel) sensor signals

- Portable (and wearable – in the sense of 'can be worn in a pocket'), battery operated transmitter, where sensors are attached by wire
- Connectivity with Max/MSP on a Windows PC

It is important to recognize at this point that a decision to develop a custom hardware instead of purchasing it, along with the low price tag also bears a great risk – which is the risk of the development not being successful at all. Failure to adhere to any of the three points mentioned above, would render the system completely useless for the VR research project.

2.5 Assumptions, risk factors and quality delimitation for development of a custom built prototype

The factors that contribute to the risk of total futility are many. First of all, designing a system on a hardware and software level demands some engineering background. Based on that, and having the previous discussion in mind, the author of this report based the possibility of building such a system on several simple assumptions:

1. A single microcontroller can be specifically programmed to perform a role of a multi-channel data acquisition system
2. Previous point (1) could be demonstrated with
    a. Circuit that would connect to one of the ports of the PC (specifically serial, since it needs only one signal wire for data transmission), and would represent a wired data acquisition system
    b. PC software which could utilize / display the acquired data
3. If (2) works, then the serial data signal wire can be replaced with a pair of ready made RF digital transmitter/receiver components, thus rendering a wireless system
    a. Thereby issues with HF electronic design are mostly avoided
4. Previous point (3) could be demonstrated with
    a. A hardware prototype that necessarily divides the (2) demonstration circuit into two distinct circuits, built around the respective components: a transmitter and a receiver circuit

5. If a demonstration for (4) works, the challenge is to have the transmitter battery operated and as small as possible
6. If a demonstration for (4) works, Max/MSP has native support for the PC serial port, thus a simple driver/decoder can be written directly in Max/MSP
7. Most, if not all, of the hardware components necessary for implementation of the given circuits, can be easily obtained (either at the component store or via external purchase).

An engineering background makes such assumptions an educated guess (and the project outcome shows that they do come out to be true); however, dealing with those assumptions on a hardware level, does require practical experience – which in the case of this report's author, at the current time, is indeed limited. And that is one of the crucial risk factors: while the assumptions may seem sound on paper, dealing with the issues that they raise in implementation does depend on practical experience; and this determines whether a prototype would work at all. Let us also mention that the requirement to go wireless means we reach the domain of high frequency (HF) electronic design, which is a very tricky and specialized area, where practical experience is crucial – and an area in which the author has even less experience in; hence 3a in the assumptions aims to compensate for this lack. That is yet another risk factor. Availability of components as they are assumed, but also the uncertainty in whether they operate in the same way as assumed, is also a serious risk factor; just as the limited time for development. The failure to establish the notion of data acquisition in electronic measurements as the hardware backbone in HCI, led to a much more uncertain research, and hence is an additional risk factor.

Provided that so many things can go wrong, development starts from scratch, and time is limited – the development of a custom hardware prototype of a wireless data acquisition system can only be undertaken at the level of a simplest possible hardware and software, with the minimum necessary number of components, that simply demonstrates that it does the job; however at the expense of quality of operation of the system. That means that the only requirement is that the prototype is working in the sense of the three previously mentioned essential properties that the Kroonde also has. As such it would be able to replace the role of the Teleo as a data acquisition system, so it implements the wireless design of the researchers and resolves their HCI issues. However, that also means that all efforts would be directed

in implementing a system that could provide a steady operation, but does not guarantee it. This report describes the prototype of the system on that very level, since that is the level the project managed to reach in the allocated time. Once the prototype is at this level, the parameters of its operation can be evaluated; only thereafter could possible improvements to the quality of operation be undertaken.

2.6 Main problem formulation and delimitations

Having the previous discussion in mind, and considering all the aspects involved, it was decided that attempting to build a custom wireless data acquisition system is reasonable to pursue, and as such, that problem constitutes the main problem area of the project described in this report. We can already here provide a conceptual model of the system demanded, based on the previous discussion of the Kroonde Gamma:

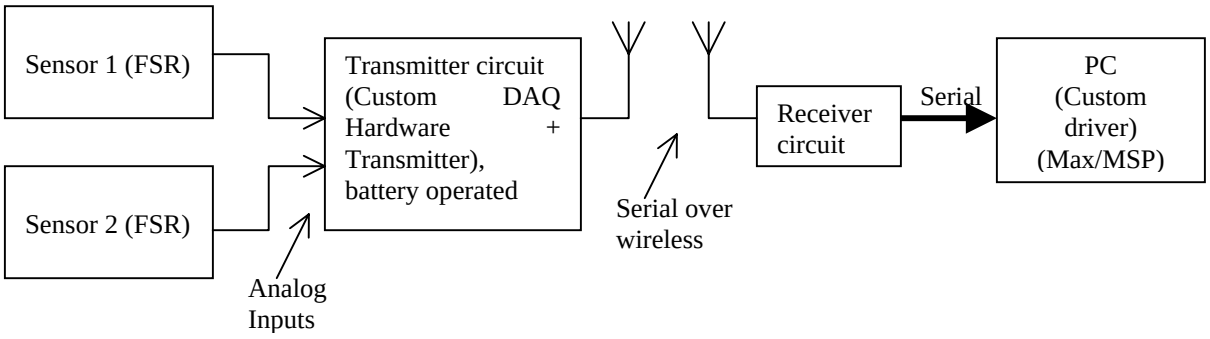


Figure 5. A conceptual block diagram of a custom wireless data acquisition system, in the context of the footstep controller and a PC

We can now proceed with the following main problem statement that guides the rest of the report:

How can we design and implement, both in hardware and software, a working prototype of a wireless data acquisition system of minimal complexity, using parts mostly available in an electronic component store, with the following properties:

1. Wireless transfer of several (multichannel) sensor signals

2. Portable (and wearable – in the sense of 'can be worn in a pocket'), battery operated transmitter, where sensors are attached by wire
3. Connectivity with Max/MSP on a Windows PC

The main delimitation in regard to the main problem statement is that the development aims only towards a demonstration of a working initial prototype of the system described – additional quality parameters cannot be taken into account at this level. Hence, the success criterion is simply to provide a working demonstration of the system; that is, it will be possible to read out several sensor values in Max/MSP using this system, in real-time fashion. Here we can also reaffirm that the main focus of the development is the electronic and software side, not so much the product design side of the interface.

The main problem formulation, in conjunction with the previously given assumptions of the author, breaks down into several formulations of problems, which represent the consecutive phases in the project:

- How can we program a microcontroller to behave like a multichannel data acquisition system?
- How can we implement the necessary hardware and software to actually utilize the programmed microcontroller as a multichannel data acquisition system, so that only one wired data transmission line of serial communication with the PC is used?
- How can we replace the wired data transmission line with a wireless one – or in other words, how can we implement RF transmitter / receiver IC modules to transform the wired data acquisition circuit into two, wireless connected circuits (a transmitter and receiver circuit) – making sure that the transmitter is portable, and the whole system can be used in Max/MSP?

These problems are discussed in the subsequent parts of this report in more detail.

3 Classification of the Teleo device as a data acquisition system

The crucial element of the design of the footstep controller is that it uses a Teleo as an intermediate device that passes the signal from the FSRs back to the computer. Hence, it is this device that we need to take a closer look at, in order to come up with a possible wireless solution. The manufacturers of the Teleo device, Making Things, themselves market this device as "a line of hardware tools that provide an easy way to create unique interactive devices, machines, and environments by controlling a wide range of sensors and actuators[2]" or "These tools, labeled under the Teleo™ brand name, consist of modular and networkable components, ranging from input and output controllers to a wide range of sensors and accessories[2]". This of course, does not provide too much technical information about the nature of these devices. Some more information can be obtained from the Teleo Starter Kit user guide, where a more detailed description is provided, which provides the information that the Teleo Introductory Module (the model of the actual Teleo device used in the research project) has a possibility for multichannel input/output operation:

"Teleo Introductory Module - A Teleo module with:

4 Analog Inputs (capable of reading 0-5V in 1024 steps)
2 Digital Inputs (which detect the presence or absence of a 5V signal)
2 Digital Outputs (which can switch devices requiring up to 2A of current on and off) and
2 PWM Outputs (which can smoothly vary the degree to which a device (like a motor or a light) is on or off).[5]"

Both the sampling rate and sampling resolution of the device can be changed via software (from Max/MSP), but are limited. The sampling resolution in bits per sample is given as "Minimum 1, Maximum 10[5]"; whereas the specification for the sampling rate is "the measurement is read every 100ms - or ten times a second. This can be changed to 1ms or up to 10000ms[5]" – that is to say the max sampling rate is 1 KHz.

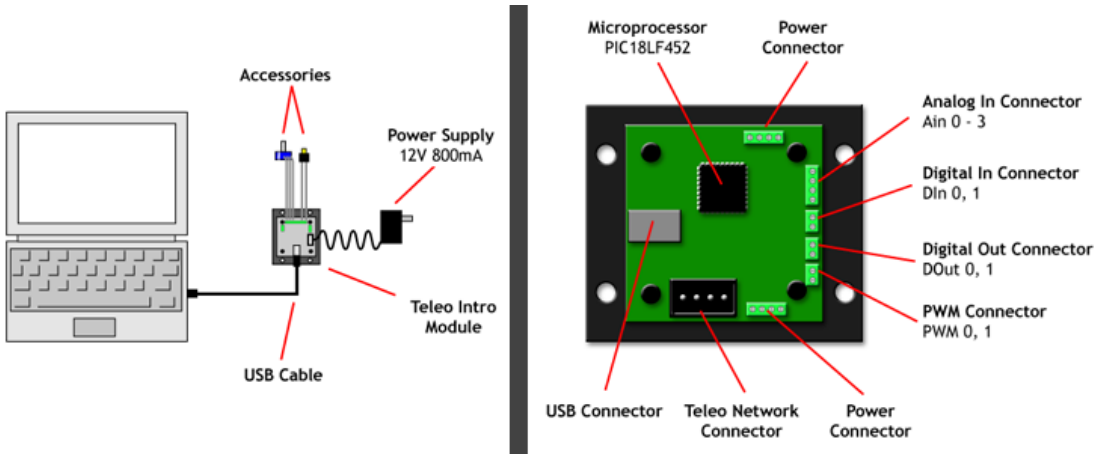


Figure 6. Connection diagram (left) and a detailed view of the Teleo Introductory module (From Ref. [5])

In essence, what the Teleo Introductory Module can do, is accept up to 4 analog voltage inputs on its analog input pins, perform analog to digital conversion on them, and pass them to a PC via USB, where company produced drivers take care of making these values available to a given software platform, such as Max/MSP. It can, as well, provide data the other way around – that is, it can reflect a given voltage (though not analog) as it is dictated by the PC; that part is however of no importance here, since the Teleo device within the VR research project is used only to digitize the values from the FSRs and pass it back to the PC.

Thus, at least within the intent of usage in the VR research project, the Teleo device acts as data acquisition (DAQ) hardware – and already from Figure 6 it is obvious that a microprocessor is at the heart of this device. The term "data acquisition" is mostly connected to the part of engineering that deals with, not surprisingly, electronic measurements – since even the role of the Teleo module within the footstep controller is basically to perform that operation: to measure the voltages produced by the sensors, and to report them to the PC. To some degree, all human-computer interaction is based on interaction with sensors being interfaced to a computer – regardless if we talk about standard peripherals (like keyboard or mouse) or more atypical devices, like the footstep controller in question. Thus, on a general level, we could state that the basic practices in data acquisition technically do constitute the base of interaction processing (on a low level) as well.

An excellent introductory tutorial ([6]) on the subject of data acquisition can be found on the Internet, provided by the National Instruments company (which deals with, again not surprisingly, measurement instrumentation – and is also the producer of the LabView software package, commonly used in engineering education for visualization and other processing of sensor data), where we can find a nice working definition of data acquisition: "Data acquisition involves gathering signals from measurement sources and digitizing the signal for storage, analysis, and presentation on a personal computer (PC).[6]". Another definition is given by Ozkul [7], which is slightly more general: "A data acquisition system is a system that acquires data from the outside world for a purpose, usually for the purpose of controlling a system ... The concept of a data acquisition system, as far as it is applied to industrial systems, means acquiring data from the outside world by means of sensors and turning it into useful information for the purpose of controlling a process or a system. [7]". Finally, a compact definition is also given by James [8], where he notes that data acquisition "at its simplest level, […] involves reading electrical signals into a computer from some form of sensor[8]".

Furthermore, when regarding hardware connectivity, "data acquisition (DAQ) systems come in many different PC technology forms for great flexibility when choosing your system. Scientists and engineers can choose from PCI, PXI, CompactPCI, PCMCIA, USB, Firewire, parallel, or serial ports for data acquisition in test, measurement, and automation applications[6]". These options can be further summed up according to the type of buses, which "can be divided into general categories:

- Stand-alone buses that are used to communicate with rack and stack instruments. These include ... PC standard buses such as Serial (RS-232), Ethernet, USB, Wireless, and IEEE 1394.

- Modular buses that incorporate the interface bus into the instrument itself. These include PCI ...[14]",

keeping in mind that we use the term bus here with the meaning it has in computer science, that is "a collection of wires through which data is transmitted from one part of a computer to another[15]".

The National Instruments tutorial provides insight into the basic theoretical concepts behind a DAQ system: "There are five components to be considered when building a basic DAQ system:

· Transducers and sensors
· Signals
· Signal conditioning
· DAQ hardware
· Driver and application software [6]"

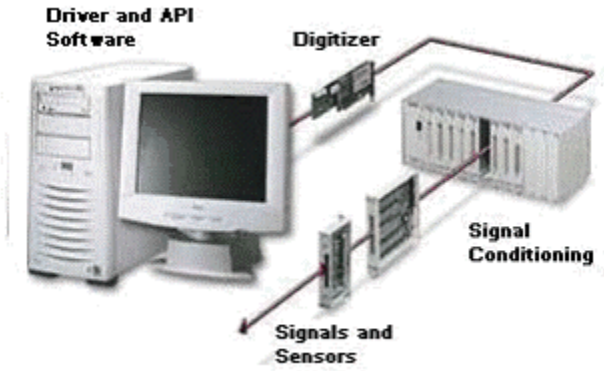


Figure 7. A data acquisition system (Ref. [6])

At this level, the similarity between the role of the Teleo in the footstep controller and the role of DAQ hardware in a basic DAQ system becomes more obvious – "The DAQ hardware acts as the interface between the computer and the outside world. It primarily functions as a device that digitizes incoming analog signals so that the computer can interpret them[6]". When we talk about the digitizing function – that is, the function of analog to digital conversion of signals, often times the unit kS/s (meaning kilo Samples per second) is used instead of kHz (kilo Hertz) to describe the sampling rate of the system, which is not a SI unit (see [25]).

Note that signal conditioning is performed in the footstep controller, in the sense that a FSR is in essence a resistor, and it changes its resistance based on pressure, and thus it does not provide voltage on its own. So, it has to be wired together with a resistor (in this case 45 KΩ) in order to form a voltage divider, which is fed by the +5V power source that is found on the Teleo (the power connectors noted in Figure 6) – however, thanks to the connector based architecture of the Teleo, this is

an operation that does not require soldering, and can be performed mechanically. This voltage divider can thus provide a voltage swing within the entire range of 0V to 5V that the Teleo can read on its analog inputs – and thus it performs the role of a signal conditioner circuit.

Another important distinction applicable to the Teleo device, is noted in the National Instruments tutorial, when they classify the devices they offer as hardware platforms – that is the distinction between a desktop and a distributed DAQ hardware platform: "The most readily available platform is the desktop computer. National Instruments offers PCI DAQ devices that plug into any desktop computer … For distributed measurements, National Instruments Compact FieldPoint platform delivers modular I/O, embedded operation, and Ethernet communication. [6]". In a desktop DAQ hardware platform, the DAQ hardware represents a PCI (PCMCIA for laptops) card, which is basically seen as part of the PC system, since it has the possibility to directly interact with the PC architecture, as the following figure shows:

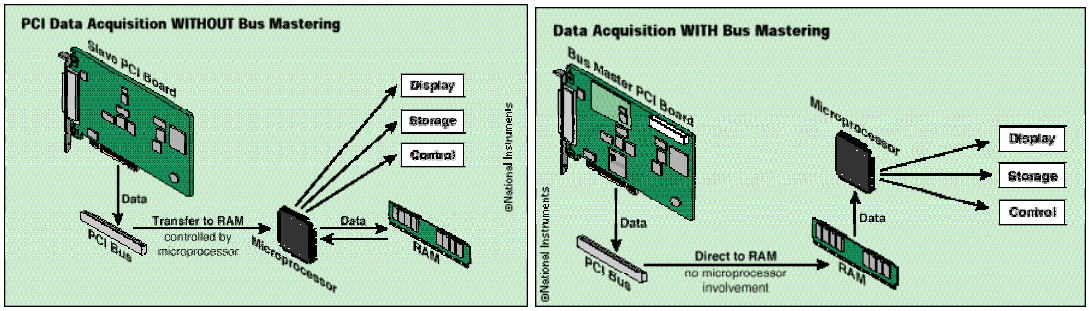


Figure 8. PCI Data acquisition(Ref. [9])

On the other hand, the term distributed DAQ hardware platform is closely related to the term distributed processing, or distributed computing – as the Encyclopedia of Microcomputers notes when discussing this subject: "Indeed, data acquisition is often particularly well suited to distributed processing, because one subsystem [...] may be only loosely coupled to another[13]". In the most general sense, distributed processing "refers to any of a variety of computer systems that use more than one computer, or processor, to run an application[10]", or in different words, it is "the ability to have several computers working together in a network, where each processor runs different activities for a user, as required[11]". Note that such simple definitions only scratch the problematic of distributed computing and the possible mechanisms involved – in [12] Wu starts introducing these systems with "when

discussing distributed systems we encounter a number of different types identified by such adjectives as: distributed, network, parallel, concurrent and decentralized ... these terms overlap in scope and are sometimes used interchangeably[12]". However, a simpler definition would help us realize that a distributed DAQ system, would involve more than one computation entity in order to perform the acquisition, and the results would be communicated through some form of a networking communication standard.

This eventually allows us to understand the Teleo device as a distributed DAQ hardware, or rather, DAQ hardware which is a part of a distributed DAQ system. This is supported not only by a presence of a microcontroller on the device, but also the fact that it uses the USB communication standard to communicate (and effectively form a network) with the host PC. Thus, the host PC has to activate a driver to receive the results from the Teleo device, to decode them and to pass them on further to the host PC operating system and API software (which in this case is Max/MSP) – unlike the desktop DAQ platform case, where a PCI card might have direct access to the host PC memory. The possibility to conceptualize a class for the Teleo device, already brings us a step closer to a possible wireless solution for the footstep controller – had we not been previously aware about a system like the Kroonde Gamma.

Let us finally note that due to the possibility of reading several sensors, one might be tempted to name the described DAQ system as a "sensor array". However, this term is mostly reserved for arrays of sensors, which are specifically placed so as to measure a property of a given wavefield, or "a sensor array is used to measure wavefields and extract information about the sources and the medium through which the wavefield propagates.[16]". For instance, a 2D array of microphones would be able to extract acoustic wavefield information. On the other hand, the option for wireless communication might tempt to describe the process as "remote sensing", however, "remote sensing is, broadly but logically speaking, the collection of information about an object without making physical contact with it[17]". In addition it is mostly used for techniques applied to sensing of Earth's surface and atmosphere, which "make use of the information impressed in some way on electromagnetic radiation ranging from utlraviolet to radio frequencies. [17]". As such, photography is an example of remote sensing; whereas in the case of the footstep controller, although we have a wireless data link, the sensors are still in physical contact with the object they are sensing.

Hence, the most appropriate term for the system described in this report would be a wireless distributed multichannel data acquisition system.

3.1 A textbook perspective on data acquisition systems

The general structure of a data acquisition system is a issue discussed in standard instrumentation engineering textbooks, like for instance [26]. Let us briefly reiterate some of those basic concepts here, to reaffirm the above discourse for the classification of the Teleo. First and foremost, in that context, the definition of a data acquisition system is much more bound by electronic engineering details, to make a distinction from a mere process of analog to digital conversion: “In addition to the A/D conversions, the conversion of analog signals into digital signals requires further operations (such as amplification and filtering of the analog signals provided by the sensors or the sampling of these signals) to be carried out, before introducing them into ADCs. The systems carrying out *all* these operations are called data acquisition systems. These systems can manage one or multiple analog signals. In the former case they are called single-channel systems, and in the latter, multichannel systems [26]”. The basic structure of a single-channel DAQ system is presented below:

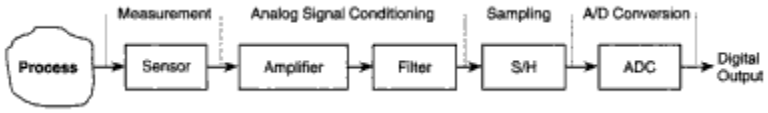


Figure 9. Basic structure and function of a single-channel data acquisition system (Ref. [26])

In essence, a physical process causes a change of some electrical property of a sensor. By performing signal conditioning, this change is manifested as an analog change of voltage, adjusted to the technical requirements for analog-to-digital conversion: “In a great number of applications, the adaptation between the analog signal generated by a sensor and the type of input demanded by an ADC requires two basic operations: amplification and filtering[26]”. By performing sampling and A/D conversion, we obtain a digital signal representation of the analog change of voltage. Although not present in this perspective, the fact that this digital signal is consumed by another machine (like a PC) is implied. It is interesting to note the need to isolate the sampling as a separate process, since usually the mention of A/D conversion implies it: "An ADC requires a finite time, called conversion time ... to achieve

conversion operations. The input voltage change during the conversion process introduces an undesired uncertainty into the generated output[26]". It is actually important to keep this in mind when trying to determine the effective sampling rate of a custom-built data acquisition system. Finally, the fact that we obtain information in digital form about a single physical process, represents the single channel of operation of this DAQ system.

Let us now take a look at the case of multichannel data acquisition, which is of interest to us when discussing the Teleo: "When there are several channels in a data-acquisition, frequently all the channels share one or more resources of the system. In this case, a switch to assign the common resource alternately to each channel is required. The function carried out by this switch is called multiplexing. Two approaches exist to achieve the multiplexing function: analog and digital multiplexing[26]". The topic of multiplexing is a very important distinction to recognize, when programming a microcontroller to act as a data acquisition device. Let us take a look at analog multiplexing first: “Analog multiplexing means that the multiplexing function is carried out on analog signals [26]”.

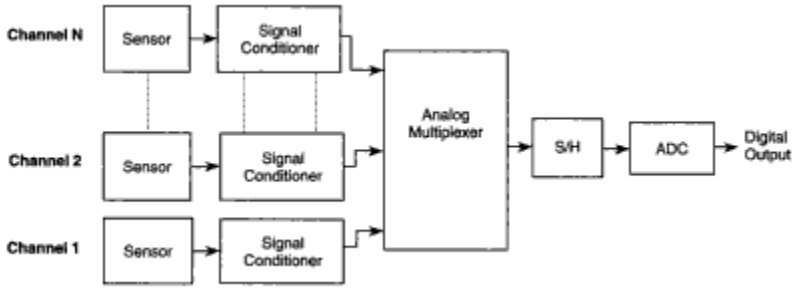


Figure 10. Common configuration of a multichannel data-acquisition system with analog multiplexing (Ref. [26])

The obvious benefit of this approach is an economical one – only one sample-and-hold (S/H) and analog-to-digital converter (ADC) circuit is used, to acquire data from multiple sensors. It is actually this architecture we are looking for in a microcontroller to allow it to be used as a data acquisition device.

However, this approach is not applicable in all cases, such as the case when a lot of sensor signals need to be processed; then a digital multiplexing approach might be better: “When there are many acquisition channels, both analog multiplexer and shared resources must be very fast, even if the signals in the channels are slow. In this

case, an alternative configuration [...] can be used [...] based on using one ADC for each channel of the acquisition system. The outputs of the converters are connected to a common bus [...] Using the bus, the digital processor can alternately access the outputs generated by each ADC, thus carrying out the digital multiplexing of signals. [26]". This case is shown on the following figure – obviously resulting with a more expensive configuration:

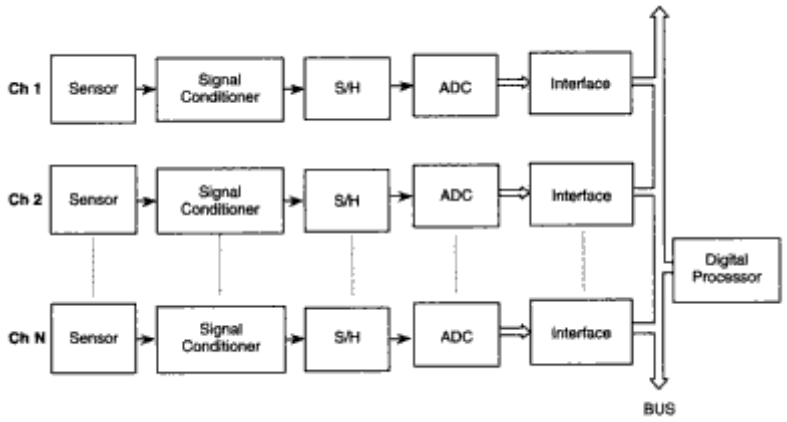


Figure 11. Common configuration of a multichannel data-acquisition system with digital multiplexing (Ref. [26])

By introducing the notion of a data-acquisition board we come closer to a concept of an actual product used for data acquisition used in conjunction with a PC: "A data-acquisition board is a plug-in board, which allows the treatment of a set of analog signals by a computer. So these boards are the key elements to connect a computer to a process to measure or control it.[26]"

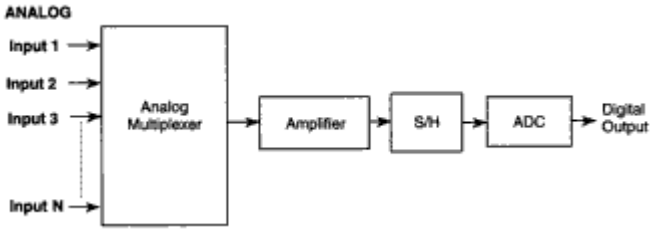


Figure 12. Input system of a data-acquisition board (Ref. [26])

An important distinction for data acquisition boards, also present in the Teleo is that "Data-acquisition boards normally operate on conditioned signals, that is, signals that have already been filtered and amplified[26]". Other attributes of data

acquisition boards also fit the Teleo device: “Some data-acquisition boards also offer analog outputs [...] [and] supply digital I/O lines [...] Digital lines are used for process control and communication with peripheral devices[26]”

Hence, from this perspective, the Teleo is simply a data acquisition board. In light of the previous discussion, we might also call it a distributed data acquisition board – to stress the fact that there is a separate processing entity that performs the data acquisition, and then communicates the results to a PC.

The previous discussion provides enough information to start looking into actual microcontrollers and the possibility to use them as data acquisition hardware.

4 Microcontroller as data acquisition hardware

4.1 Introduction to microcontrollers

Let us first introduce some basic definitions for microcontrollers, which will help us follow the discussion better. In order to do that, we will mostly borrow the from the introductory articles on the subject from Wikipedia, the online encyclopedia, for a quick and concise overview of the terms of importance. The basic notion is given as: "A microcontroller is a computer-on-a-chip used to control electronic devices[27]", where "chip" means integrated circuit – "a miniaturized electronic circuit which has been manufactured on a thin substrate of semiconductor material[28]", and "computer" is defined as "machine capable of processing data according to a program — a list of instructions[29]". Furthermore, a microcontroller "is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor, the kind used in a PC. A typical microcontroller contains all the memory and I/O interfaces needed, whereas a general purpose microprocessor requires additional chips to provide these necessary functions[27]". They are very common in the modern world, as they represent the majority of processor units sold: "A typical home in the Western world is likely to have only one or two general-purpose microprocessors but somewhere between one and two dozen microcontrollers. They can be found in almost any electrical device, washing machines, microwave ovens, telephones etc[27]"

Figure 13. The look of a microcontroller - just like any other IC – here Microchip PIC 16F74 (Ref.[37])

The perspective of self-sufficiency is elaborated further: “Most microcontrollers today are based on the Harvard architecture, which clearly defined the four basic components required for an embedded system. These include a CPU core, memory for the program (ROM or Flash memory), memory for data (RAM), one or more timers […] as well as I/O lines to communicate with external peripherals and complementary resources — all this in a single integrated circuit. A microcontroller differs from a general-purpose CPU chip in that the former generally is quite easy to make into a working computer, with a minimum of external support chips. The idea is that the microcontroller will be placed in the device to control, hooked up to power and any information it needs, and that's that[27]”. Let us note that not that long ago, microcontrollers did not always contain memory on board, which in practice forced usage of additional external EPROM memory chips (see [45] for a good overview).

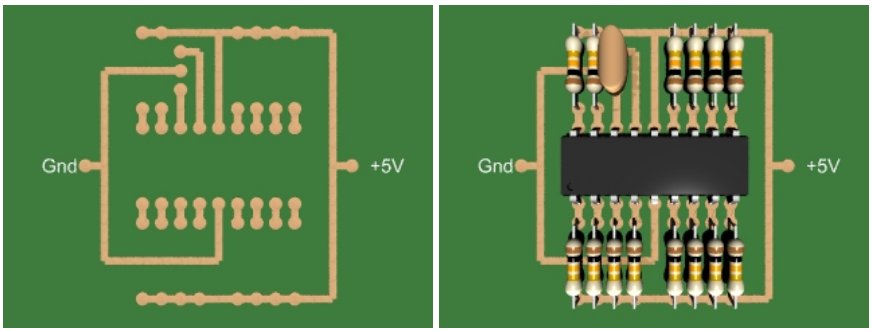


Figure 14. A basic circuit for a microcontroller, without and with components (Ref.[38])

Thus, all that one would need to make a microcontroller work is power, a program, and a clock that times the execution of a program: “a typical microcontroller will have a built in clock generator and a small amount of RAM and ROM meaning that to make it work, all that is needed is some control software and a timing crystal[27]”. Such a basic circuit is described in the excellent online tutorial [38].

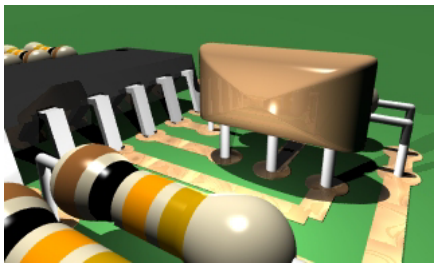


Figure 15. 3D View focused on a timing crystal (here a crystal resonator, Ref. [38])

Additionaly, they include built in functionality, which is crucial in the intent to use them as DAQ hardware: "Microcontrollers will also usually have a variety of input/output devices, such as analog-to-digital converters, timers, UARTs …[27]". We would need ADC's on board to implement DAQ functionality, but also we would need a way to communicate to a computer, which is why we should also focus on the UART as well, which stands for "Universal Asynchronous Receiver-Transmitter [,] a piece of computer hardware that translates between parallel bits of data and serial bits. A UART is usually an integrated circuit used for serial communications over a computer or peripheral device serial port. UARTs are now built into some microcontrollers[30]" – obviously, this is the part would ensure connectivity with a PC. Let's finally mention that recent advances in microcontroller technology have also brought about microcontrollers with built in radio transmitters – which obviously would fit the purpose in the project perfectly.

As microcontrollers represent programmable devices, the program they run is an integral part of their functionality: "Originally, microcontrollers were only programmed in assembly language, or later in C code … More recently, however, some microcontrollers have begun to include a built-in high-level programming language interpreter for greater ease of use. BASIC is a common choice[27]". Thus programming the microcontroller is a crucial part in implementing a given function. It consists of essentially two steps: the first step is *coding* the software, which means writing code in a high level language, and compiling it to assembler (machine) instructions – which is mainly a software operation. Typically the manufacturer will provide some software platform for coding development, with the necessary compilers, which runs on a PC:

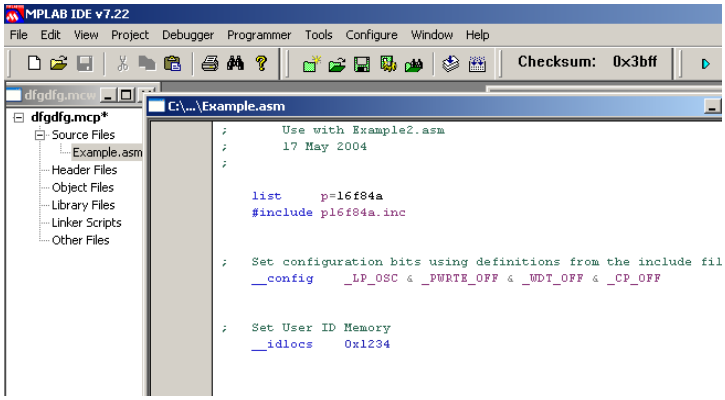


Figure 16. Look of a assembler coding platform - Microchip MPLab IDE

Basically, whatever we write needs to be translated into the simplest commands – commands from the instruction set that define the machine language that the microcontroller understands. Let's look at these terms a bit closer: "Machine code or machine language is a system of instructions and data directly understandable by a computer's central processing unit. The 'words' of a machine language are called instructions, each of which cause an elementary action by the CPU, such as reading from a memory location. A program is a sequence of instructions that are executed by a CPU. Humans use mnemonic codes to refer to machine code instructions. Such a more readable rendition of the machine language is called an assembly language and consists of both binary numbers and simple words whereas machine code is composed only of the two binary digits 0 and 1. For example, on the Zilog Z80 processor, the machine code 00000101 causes the CPU to decrement the B processor register. In assembly language this would be written as DEC B.[34]" It is actually here that we could establish a simple relationship between execution of machine language instructions, and the need for a clock to make the microcontroller work; for instance, a certain microcontroller may have "a RISC design that runs one instruction per cycle (4 oscillator cycles)[27]" – that is, there is a direct relationship between the oscillator clock timing, and execution of machine instructions.

The second step is *programming* the code in the microcontroller (also known as "burning" the microcontroller), which involves a process of electrical transfer of the compiled machine instructions into the microcontroller – obviously an operation that is on a hardware level. Thus, in this context, we should think about two programming aspects: which platform do we use to write and compile the program, and on the other hand, which platform do we use for burning the compiled program into the microcontroller. Most of the time, both of these issues may involve some considerable one-time costs, related to the two programming aspect: and that is acquisition of a software development environment (compiler); and acquisition of a hardware programmer ("burner") and corresponding to software to drive it. Occasionally, all this software functionality may be merged as one package, known as integrated development environment (IDE). An IDE will more than often also contain a simulator, which allows one to virtually run the compiled the program, with the possibility to visualize the microcontroller operation, as it would behave in an actual circuit.

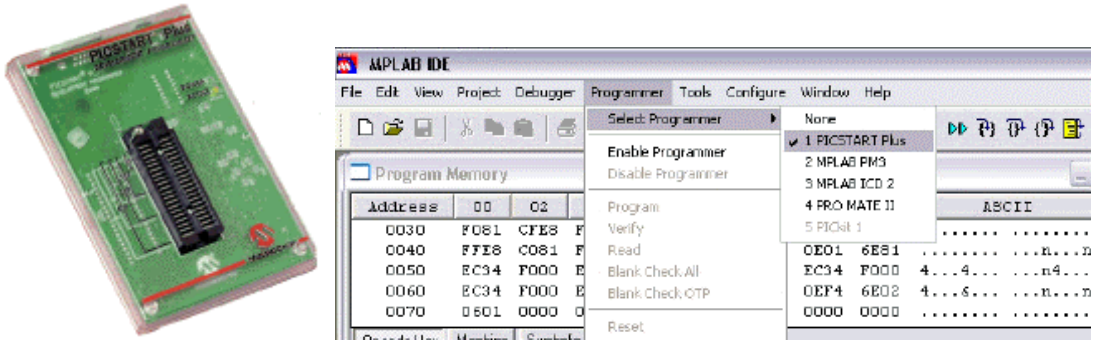


Figure 17. The look of a commercial hardware programmer (Microchip PICStart Plus) and the corresponding driving software (Microchip MPLab IDE, Ref. [32])

There is a number of producer that manufacture microcontrollers; the Wikipedia entry lists 19 of them as producers of "common" units, among them Atmel, Freescale Semiconductor (Motorola), Intel, Microchip, Infineon (Siemens), Philips Semiconductors, etc. In addition to these, there are also "endless BASIC programmed MCUs - for almost every bare microcontroller manufacturer, there are a dozen little companies repacking them into a more hobbyist friendly package. Their product is often an MCU preloaded with a BASIC interpreter[27]" – some of the more famous companies being Parallax, Inc. with their Basic Stamp device. Every manufacturer of microcontrollers will of course provide hardware and software development tools for coding and programming their microcontrollers. In any case, when talking microcontrollers, the Microchip company is interesting, since they are the manufacturer of the microcontroller used in this project (and for that matter, the microcontroller present on the Teleo).

4.2 The choice of PIC 16F74 and the corresponding development tools

Having the previous discussion in mind, an inquiry into possible microcontrollers for use in the project was undertaken. Obviously the availability of microcontrollers with built in RF transmitters, like Microchips rfPIC12C509AF [39] would seem an obvious choice in regards to the goals of the project. However, their availability was seen as a risk, and in any case the determination was already stated to build on components available in the component store. In retrospect, that seems to be a good decision – for instance the mentioned rfPIC12C509AF is adjusted for use with a loop antenna, which again asks for certain experience in HF electronics; such issues

were in this project luckily avoided. The component store had several microcontrollers available, among others Infineon 80C535 [40], and Microchip PIC 16F74, PIC 16F84, and PIC 16F876. The Infineon chip is discontinued, and searches on it indicated that this device may not be all too popular. On the other hand, searches for Microchip PICs returned many Internet sources on usage examples, many by hobbyists and enthusiasts as well. A quick peek in the specifications determined PIC 16F74 (currently priced at around 5 Euro) as the best candidate for usage in the project: "The PIC16F74 features 8 channels of 8-bit Analog-to-Digital (A/D) converter with 2 additional timers, 2 capture/compare/PWM functions and the synchronous serial port can be configured as either 3-wire Serial Peripheral Interface (SPI™) or the 2-wire Inter-Integrated Circuit (I^2C™) bus and a Universal Asynchronous Receiver Transmitter (USART) [41]". Features like 4K on board memory, multiple A/D conversion channels and possibility for serial communication is exactly what we need from a data acquisition device (the 16F84 microcontroller for instance neither has ADCs nor an USART).

Microchip microcontrollers are often referred to as PIC microcontrollers or simply PICs: "PIC, is a family of RISC microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division. Microchip Technology does not use PIC as an acronym; in fact the brand name is PICmicro. It is generally regarded that PIC stands for Peripheral Interface Controller, although General Instruments' original acronym for the PIC1650 was 'Programmable Intelligent Computer'[27]".

In regard to hardware and software tools for coding and burning, as most manufacturers, Microchip too will give out the IDE for free, but with certain limitations: Mirochip's MPLab[31] is for instance free, but it will let us compile programs only in assembler (which is a rather difficult matter); if we want to program in C, then there is an option for a C compiler, which is not free (although a student version is available, which as a demo, cripples some compiler optimizations after 60 days – it will however, still work). There are also C platforms for Microchip PICs by other manufacturers which are freeware, such as Picc-lite [43] by HI-TECH – but again, the freeware version is limited to only a certain number of models. In any case, the hardware programmer is not free – the entry level Picstart Plus Development System is priced at 170 Euro. However, at least for Microchip PIC there are alternatives, many of them to be found on the Internet: "PICs are well-documented on

the Internet [...] the now obsolete PIC16F84 was the first widely available microcontroller that could easily be re-programmed by hobbyists[27]". The Internet offers a multitude of designs for hardware programmers that are commonly used by hobbyists, as well as free "burning" software that can be utilized with them.

In these sources, we often find the name of David Tait, a lecturer at Manchester School of Engineering, as historically one of the first people that stimulated this development by providing free schematics and software on the Internet, allowing hobbyists to program PICs with home made hardware. David Tait has since stopped his activities in this matter, so his webpage [36] is not maintained any more – however its contents are mirrored on several sites. Tait has come up with several designs for a hardware programmer, of which "the classic Tait programmer" (see Appendix A), which is connected to the parallel port of a PC, represents a solution with relatively few components, which also allows relatively easy hardware troubleshooting. This circuit can be found in many different incarnations on the Internet, one of which is the hardware programmer that was built for usage in this project – and of course, it can be used to program a whole lot of PICs, among others the chosen 16F74.

The initial possibility for a complete development platform came from a company called Oshonsoft, which offers software for PIC development along with programmer circuit schematics. Their Parallel Port PIC Programmer [44] software is offered as freeware, along with the corresponding programmer hardware schematics – which is a version of the classic Tait programmer:

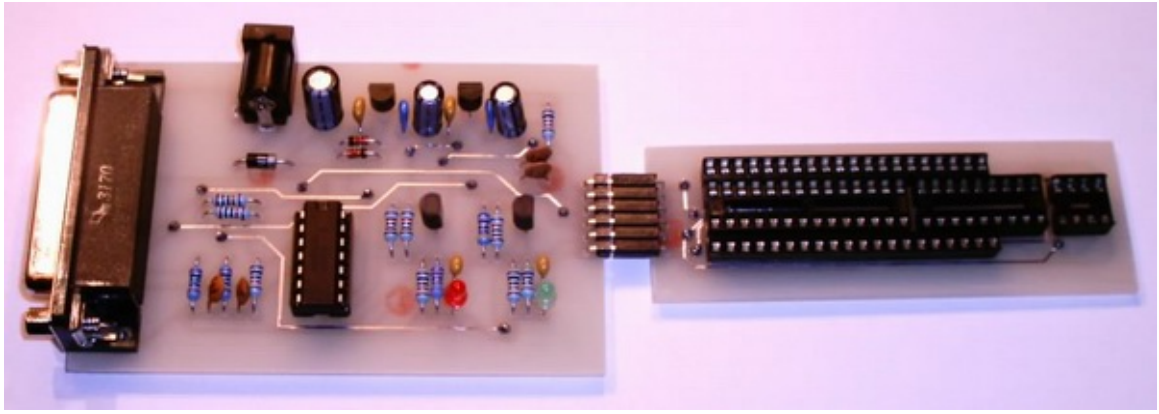


Figure 18. The Oshosoft version of a parallel port Tait programmer (Ref [44])

Oshonsoft also offers an IDE that is called PIC Simulator IDE: “PIC Simulator IDE is powerful application that supplies PIC developers with user-friendly graphical development environment for Windows with integrated simulator (emulator), Basic compiler, assembler, disassembler and debugger[46]” It is attractively priced at 29 Euro, and there is also a demo limited to a 30-day evaluation period – and it is certainly worth the investment. This IDE is very easy intuitive, quick to learn and easy to use, and the wealth of the examples available allowed that several versions of the data acquisition program were developed in BASIC and compiled. In addition, they were simulated as well; since a software UART emulator is present on this IDE, as well as oscilloscope emulation, one could simulate and check whether a PC would understand the serial code generated by the program, before anything was actually built.

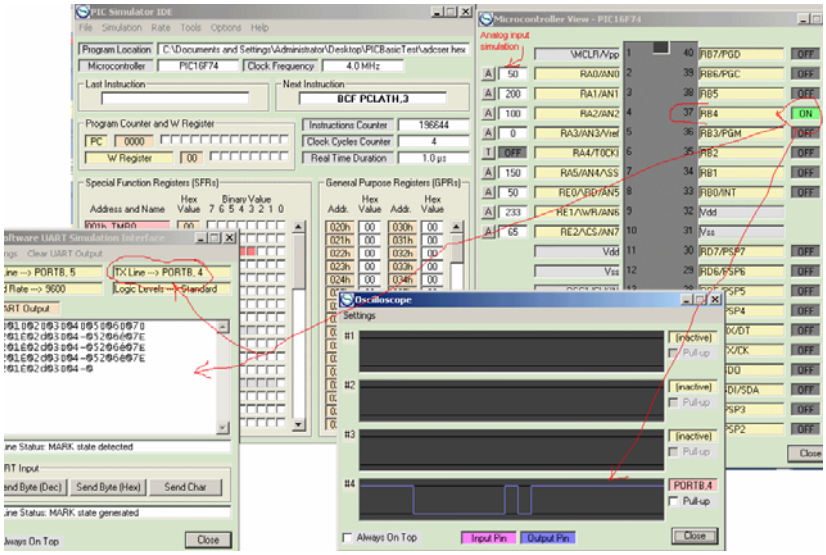


Figure 19. PIC Simulator IDE simulating the data acquisition program; the oscilloscope shows the serial electric signal, and the UART emulator shows how would that signal be decoded as characters

This IDE does not integrate the “burning” function – instead the free Parallel Port PIC Programmer is offered; unfortunately the eventual programmer hardware that was built in this project would simply refuse to be driven by it. Hence other options were researched for programmer software that could drive the classic Tait programmer through the parallel port of a PC (Microchip’s MPLab is bound to its own hardware programmers). It is possible to use other programmer software, although the code was developed in BASIC using PIC Simulator IDE – since the result of the compilation of the BASIC program, is actually a generation of an assembler program (which is a translation of the one in BASIC); and finally, a machine code

translation which is intended to drive a programmer hardware – this format is known as the Intel HEX format (see for instance [47] for a quick intro). All of the programmer software packages had support for the HEX file format. The following free programmer software platforms for Windows XP were tried out:

- WinPicProg (1.95c) by Nigel Goodwin [48]
- WIN PIC Programmer (Nov-2005) by Wolfgang Buescher [49]
- PICALL Programmer (0.16) by Bojan Dubaj [50]
- IC-Prog (1.05D) by Bonny Gijzen [51]

All of these platforms can target the programmer via the parallel port; they all have some distinct features, like the possibility to extend the clock cycle for programming, or possibility to trigger individual pins of the parallel port. However, for some reason, only IC-Prog managed to work completely and actually "burn" the microchip with the built programmer hardware; all the other failed. There could be several reasons for this. As first, there were quite some problems during hardware development of the circuit, which were traced back to poor manufacturing of the PCB. It is possible that the total reliability of the programmer hardware circuit was so low, that it couldn't respond properly to the signals sent by all other programming software. The second reason may be traced to the complications of using the parallel port in the PC: there are several modes in which it can work (like EPP or ECP), which can be set in via BIOS; and Windows is also known to interfere with the parallel port operation, also during the process of programming.

In any case, it was possible to write the microcontroller, and successfully verify the programmed microcontroller, so this step of the development process was successful. The microcontroller program that defined the data acquisition operation was written in BASIC – developed, compiled and simulated in PIC Simulator IDE on WinXP. The compiled HEX program was programmed into the microcontroller using IC-Prog 1.05D on WinXP, driving an unstable version of the Tait classic hardware programmer (originally the Oshonsoft version, but eventually stripped down to classic Tait) through the PC parallel port.

The following sections will discuss the microcontroller programming part of the development process, as well as the data acquisition software, in more detail.

4.3 Hardware programming process of PIC 16F74 via parallel port

This section will describe the hardware programmer circuit, and introduce some concepts behind its operation. First of all, let's introduce the pin layout of the chosen PIC 16F74, given in this products datasheet [42] that is central in this discussion:

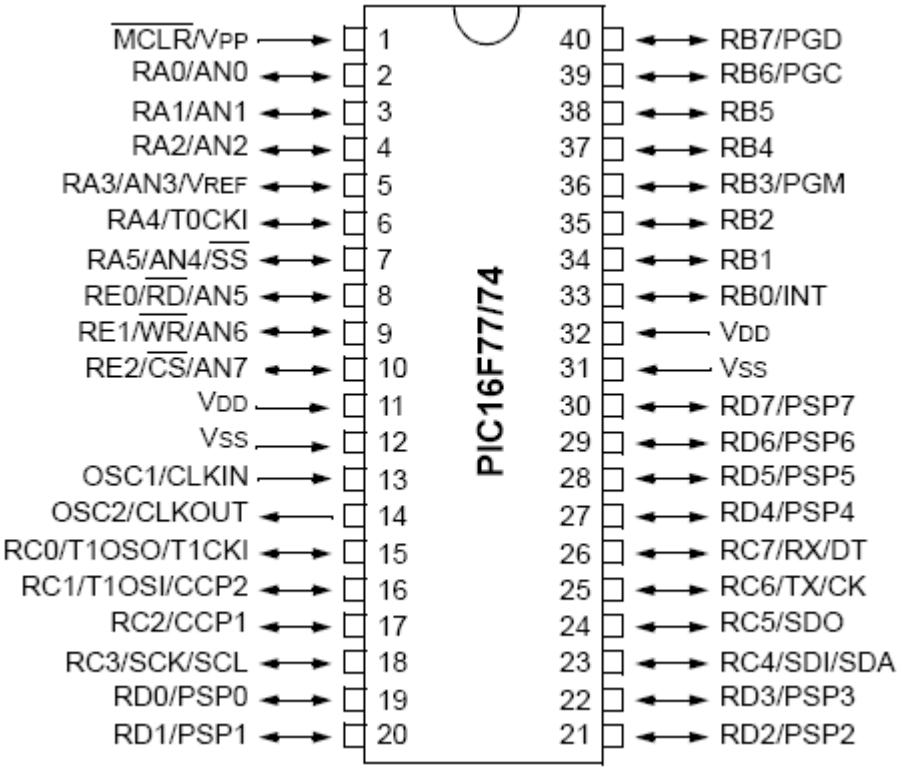


Figure 20. Pin layout of the 16F74 (Ref.[42])

The pin layout is important, since that is how we communicate with the microcontroller. In interaction terms, we apply and read voltages as input and output actions of the microcontroller, just as we use the keyboard and the monitor in the case of a desktop PC. Each pin can either accept a voltage, acting electrically as a load, in which case the microcontroller could "read" the input on that pin; or can generate a given voltage, according to a running program, and in that sense the output is available to the outside world. The arrows by the pins indicate this directionality – whether the pin can act as an input or as an output. If it can act as either, it can be set for such operation via software.

Several pins have a special role, and are given special acronym, which are often the same for most Microchip PIC models:

- VDD (pin 11 and 32) - Positive supply for logic and I/O pins.
- VSS (pin 12 and 31) - Ground reference for logic and I/O pins.
- MCLR/VPP (pin 1) - Master Clear (input) or programming voltage (output).
- OSC1/CLKI (pin 13) - Oscillator crystal or external clock input.
- OSC2/CLKO (pin 14) - Oscillator crystal or clock output.
- AN0-AN7 (pin 2-5 which is PORTA and 7-10 which is PORTE) – Analog inputs
- RB3/PGM (aka LVP) (pin 36) - Digital I/O or Low voltage ICSP programming enable pin.
- RB6/PGC (pin 39) - Digital I/O or RB6/PGC In-Circuit Debugger and ICSP programming clock.
- RB7/PGD (pin 40) - Digital I/O or In-Circuit Debugger and ICSP programming data.

The first important issue is what kind of a power supply does the 16F74 need. In the datasheets it is noted as VDD supply voltage, and is allowed to be between 4 and 5.5V. Mostly we would aim to provide 5V of supply to the VDD and VSS pins (since there are two pins for each of these, it simply means that in the circuit we have to connect each pair of pins between themselves with a conductive trace, in addition to the connection to the power supply). The microcontroller needs this power regardless of whether it is being programmed, or it is executing a program.

The next important pin is MCLR/VPP – pin 1. Through it, we set whether the microcontroller should be programmed (programming mode) or it should execute a program (let's call it execution mode). When the microcontroller is executing the program, MCLR is kept at 5 V via a pull up resistor, so, in addition to power supply brought to VDD and VSS, we need to utilize OSC1 and OSC2 pins to bring a timing signal to the microcontroller. While we are in execution mode, we can use PGM, PGC and PGD just like any other I/O pin (that is, in user mode those pins are treated as RB3, RB6, RB7 – part of PORTB I/O port).

We express the desire to enter programming mode by setting MCLR to a high voltage known as VPP - programming voltage ("High voltage on MCLR for chip erase and program write operations") – ranging from 12.75V to 13.25V. Such information is part of the hardware programming specification for the 16F74, which is a separate document called "PIC16F7X FLASH Memory Programming Specification" [52] and is not included in the datasheet. There we find that "the Program/Verify mode is entered by holding pins RB6 and RB7 low, while raising MCLR pin from VIL to VPP. Once in this mode, the user program memory and the configuration memory can be accessed and programmed in serial fashion [...] The sequence that enters the device into the Programming/ Verify mode, places all other logic into the RESET state. [52]" VIL is defined as input low voltage, and in the datasheet it has the range between VSS (which is ground, that is 0V) and 0.2*VDD (for VDD=5V this is 1V); let's note that VIL depends on VDD – so if we want to program, that means we also have to set VDD at 5V as well. In other words, to enter the programming mode, we hold RB6 and RB7 between 0 and 1V, we set MCLR to between 0 and 1V and then we raise MCLR to between 12.75 and 13.25V, and VDD to 5V. In reality, we will aim to have 0V for a low level, and around 13V for a programming voltage VPP. Let's also note that entering the programming mode also resets the programs counter: "A device RESET will clear the PC and point to address 0x0000 [52]".

Once in programming mode, we can issue commands to the microcontroller and program it in that way. Since we are talking about controlling the microcontroller through changing voltages here, it's important to notice how the hardware requirements for programming are formulated in the specifications: "The PIC16F7X requires two programmable power supplies, one for VDD (2.0V to 5.5V) and the other for VPP of 12.75V to 13.25V [52]". It is through these programmable power supplies that we program the microcontroller – and that is why a hardware programmer is necessary. In essence, a battery with a switch is a programmable power supply – by turning a switch on and off we can "program" a sequence. The power supply term stresses that whole circuits are to be fed with these voltages; this might demand additional current – hence power requirements need to be met. The datasheet contains information for proper calculation of these power requirements, but as we are going to use an ready schematic for the programmer, we need not be concerned with it further. The procedure described so far (known as HVP or High Voltage Programming) is not the only way to program a PIC, there are other ways as well; discussing them is out of the scope of this report. However, let's note that the

described HVP procedure implies that a separate circuit (the hardware programmer) needs to be used to program the chip; when programmed, the chip is removed and placed in the actual circuit that allows the execution of the microcontroller program.

The question is then, how to control two power sources independently from a computer? Actually, we will soon see that we need more than just two control lines, which makes the choice of the parallel port of the desktop PC reasonable. Let us take a look at the pinout of a PC parallel port:

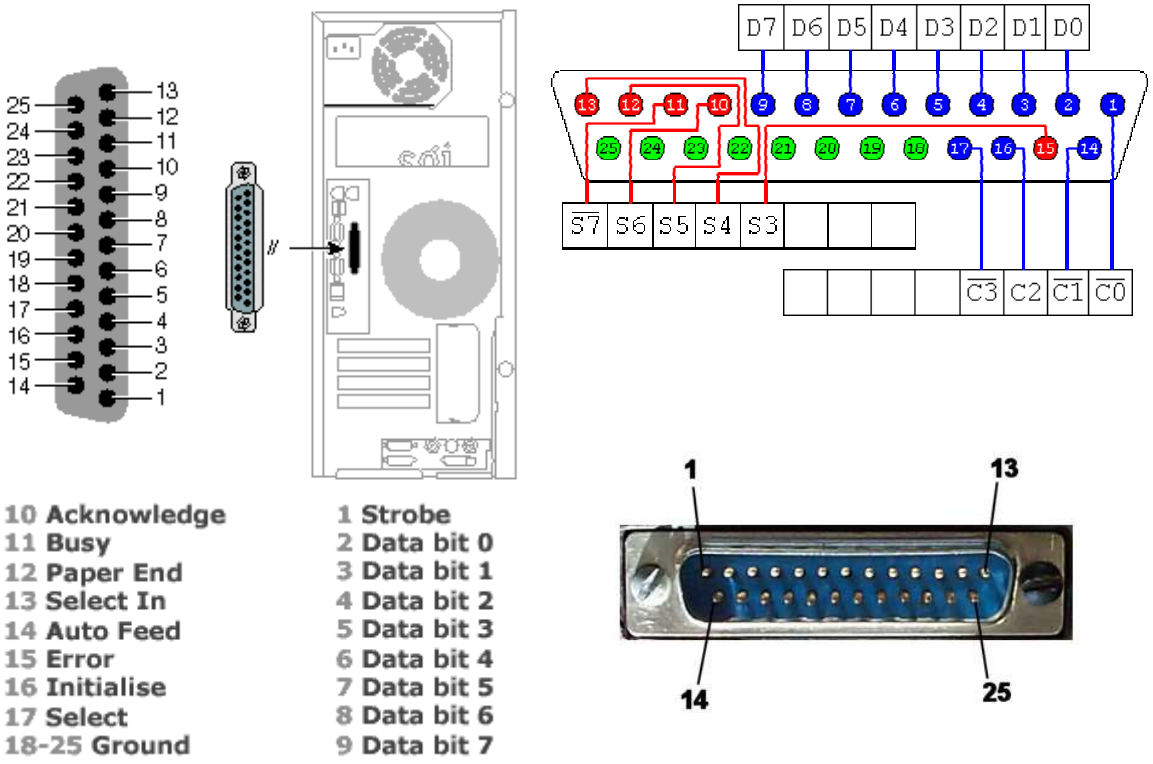


Figure 21. Left: Parallel port connector on a PC (Ref.[53]), right up: pinout for a female DB25 connector (Ref.[54]), right down: pinout for a male DB25 connector (Ref.[55]), down: pin names (From Ref. [56])

The parallel port on a PC is represented with a 25-pin connector known as DB25 (or D Type connector or 25 pin DSub). The parallel port should enable parallel communication, meaning transfer of several bits of information simultaneously. Hence we need just as many independent digital communication lines (wires) in a communication cable – but also, there is a need for programmable sources to drive them. Usually, in a parallel port, it is pins 2-9 (data bits 0 to 7) which are used for this purpose. With the parallel port, we have at least 8 programmable sources by software;

by looking into the electrical characteristics of the parallel port we can see that it is in accordance with TTL logic levels, where logical high and low are represented with voltages between 0 and 5V or more precisely:

> "An electrical "high" (on a pin or line) is TTL high, +2.4 to +5 volts.
> An electrical "low" is TTL low, 0 to +0.8 volts.[57]"

Quite often, the measurements of these voltages show that in reality a high level on a pin can well be around 3V, which shows why these tolerance ranges are important. In any case, we have at least 8 programmable sources, which can be switch from 0V low level to around 3V high level, which is definitely not enough to act as neither VDD (5V) nor VPP (13V) power supply for the microcontroller. To verify the programmability, we can do test measurements via a parallel cable and software that can individually control the pins of the parallel port. Several of the programmer software platforms noted before offer this option, just as well as other utilities, such as VBPortTest [58]:

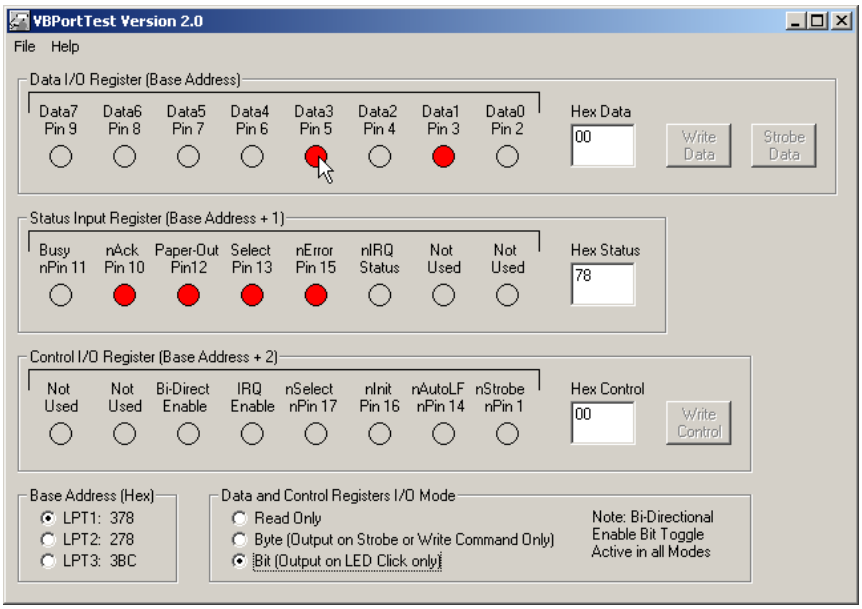


Figure 22. Screenshot of VBPortTest utility

By simply clicking on the corresponding buttons, we can activate individual pins high and low, and confirm that via measurements of the pin voltages. It is obvious from this that we have independent sources that are controllable via software as the parallel port pins; the software programmer will control the pin voltages and create

the sequences needed for programming the microcontroller, and the hardware programmer will simply condition these signals, and make sure that the signals are adjusted to what the microcontroller expects (that is VDD and VPP).

Aware of this, we can now proceed describing the rest of the hardware programming process. Once in programming mode we can proceed with the serial program/verify operation: "RB6 is used as a clock input pin, and RB7 is used for entering command bits and data input/output [52]", hence the names PGC (programming clock) and PGD (programming data) for them. The programming process consists of sequential entering of commands and data, which have different formats: "To enter a command, the clock pin (RB6) is pulsed six times. Each command bit is latched on the falling edge of the clock (RB6), with the Least Significant bit (LSb) of the command being input first. The data on pin RB7 needs a minimum setup (tset1) and hold time (thold1) with respect to the falling edge of the clock [52]". For those commands that are followed by data, there is a delay time specified, after which "the clock pin is cycled 16 times, with the first cycle being a START bit (0) and the last cycle being a STOP bit (0). Data is transferred LSb first [52]". That process is illustrated on the following diagram:

FIGURE 5-1: LOAD DATA COMMAND MODE (PROGRAM/VERIFY)

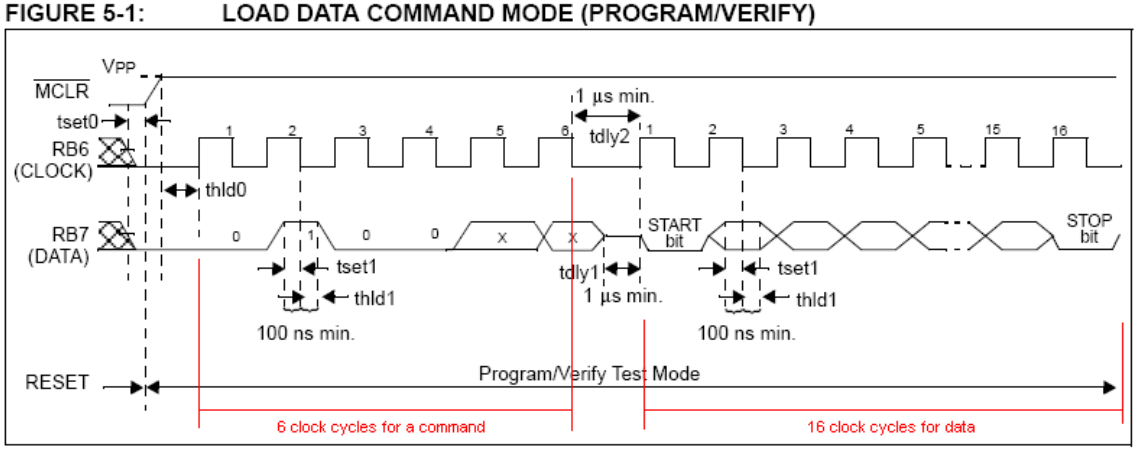


Figure 23. Time diagram of MCLR, clock and data signals for a Load Data command – the X from the command spec means that the actual value doesn't matter (From Ref. [52])

In programming mode, the 16F74 microcontroller understands several commands, which are represented in the following table:

TABLE 2-2: COMMAND MAPPING FOR PIC16F7X

Command	Mapping (LSb ... MSb)						Data (LSb first)
Load Configuration (Set PC = 2000h)	0	0	0	0	X	X	0, data (14), 0
Load Data for Memory	0	1	0	0	X	X	0, data (14), 0
Read Data from Memory	0	0	1	0	X	X	0, data (14), 0
Increment Address	0	1	1	0	X	X	
Begin Programming	0	0	0	1	X	X	
Bulk Erase Program Memory (Chip Erase)	1	0	0	1	X	X	
End Programming	0	1	1	1	X	X	

Figure 24. Command mapping for PIC 16F74, the Xs indicating a value that can be either (Ref. [52])

Using these commands the microcontroller can be programmed, for instance one program memory word at a time, by using a sequence like this: "

1. Set a word for the current memory location using the 'Load Data' command.
2. Issue a 'Begin Programming' command to begin programming.
3. Wait tprog.
4. Issue an 'End Programming' command.
5. Increment to the next address ('Increment Address' command)
6. Repeat this alternative sequence as required to write program and configuration memory. [52]"

In essence, that is the task of the programmer – the pins that are used for programming (VDD, MCLR, PGC, PGD) will be mapped to some parallel port pins through the hardware programmer; the software programmer then needs to manipulate the corresponding parallel port pins to first execute a sequence for entering programming mode, and then repeat a sequence like the above until the entire HEX file is written. We "exit the Program/Verify Test mode by taking MCLR low [52]".

Let's note that due to code protection, which is by default turned on, once programming mode is exited, consequently entering programming mode and trying to read the chip will result with all 0's, although the chip is not empty: "Once code protection is enabled, all program memory locations read all '0's; further programming of program memory is disabled [52]". That is why code protection should be disabled before start of programming, and that is done by executing the Chip Erase command, however "all program memory will be erased when this procedure is executed and

thus, the security of the code is not compromised [52]". Most software programmers will do this by default, however it is important to keep in mind, otherwise the attempts to read a chip can sometimes be very surprising.

Let's finally mention that it is possible to use an application that allows individual manipulation of the parallel port pins (like some of the software programmers or VBPortTest), to manually enter sequences for programming and reading the microcontroller, using the hardware programmer. The programmer design features a feedback from programming data PGD to the corresponding parallel port pin, so one can enter programming mode, save some word on the first address, read the same address and have the saved value indicated on the parallel port pin state in the software – by manually clicking the parallel port pins (that correspond to clock, data, VDD and VPP) in the application on and off. Although a bit tedious process, it can be used to confirm that the hardware is working properly, if there otherwise may be a problem with the software programmer.

It is also important to note that programming via the parallel port can sometimes fail due to hardware problems with the parallel port. Occasionally, the parallel port mode (like ECP (Extended Capability Port), EPP (Enhanced Parallel Port) etc) will not be compatible with the hardware. Laptops are also rumored to have worse parallel port performance than desktop computers. Finally, Windows XP is also rumored to mess with the parallel port operation, which can destroy the programming process. In this project, a laptop was used, with the parallel port set to ECP mode in BIOS. Nothing special was undertaken in regard to Windows XP or drivers, although occasionally, some software programmers would report such a problem – as they have that possibility as well. That could also be one reason why only one software programmer out of 4 managed to work.

4.4 Hardware programmer circuit

The schematic for the hardware programmer is the version found on the Oshonsoft site:

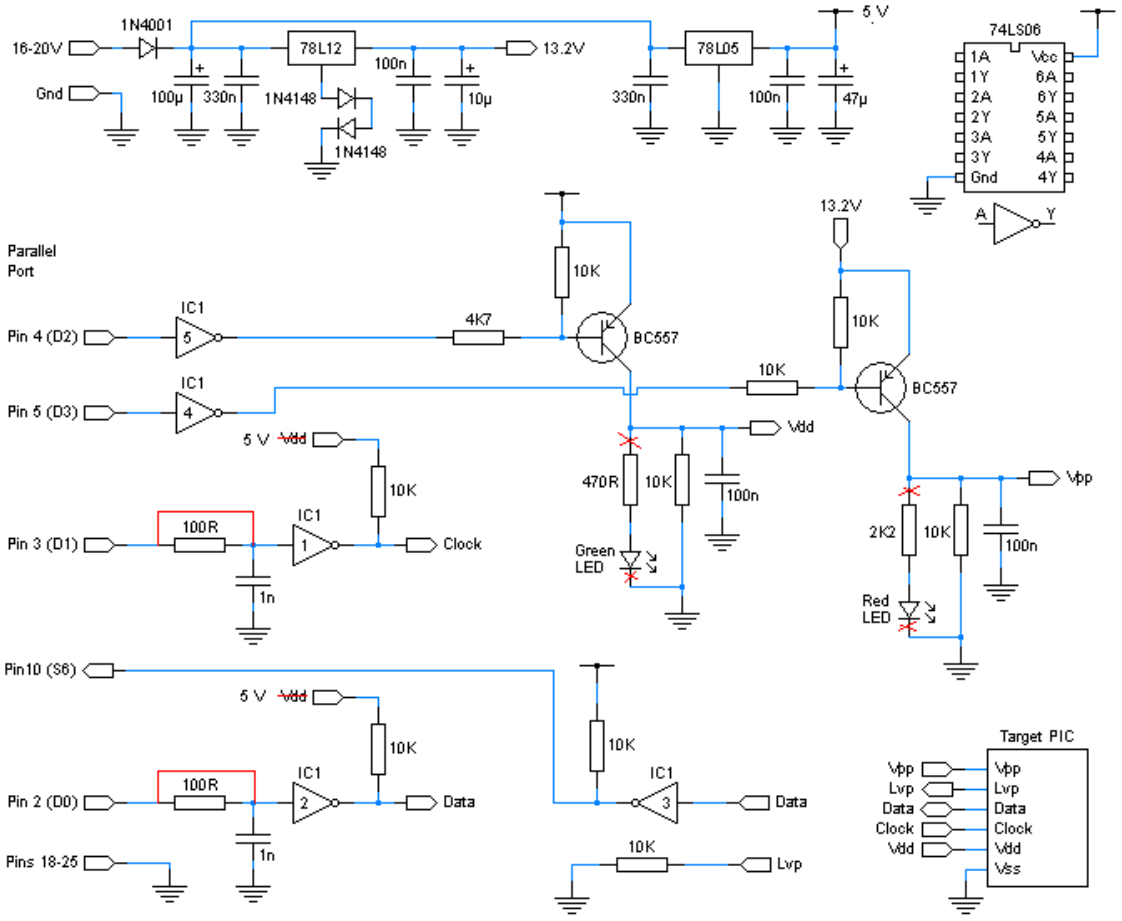


Figure 25. Schematic of the Oshonsoft version of the programmer, with modifications (From Ref. [44])

Certain modifications that were performed to this circuit during the troubleshooting process are also noted. Let us first realize that the schematic sets the hardware mapping between parallel port pins and the microcontroller pins:

Parallel port pin	*-- Microcontroller pin*
D0 (Data bit 0, pin 2)	-> PGD (programming data 0-5V)

S6 (Acknowledge, pin 10) <- PGD (programming data 0-5V feedback)
D1 (Data bit 1, pin 3) -> PGC (programming clock 0-5V)
D2 (Data bit 2, pin 4) -> VDD (Positive supply for logic and I/O pins 0-5V)
D3 (Data bit 3, pin 5) -> MCLR/VPP (Programming voltage 0-13V)

Obviously, we need to somehow generate 5V and 13V. Let's focus on the corresponding parts of the circuit:

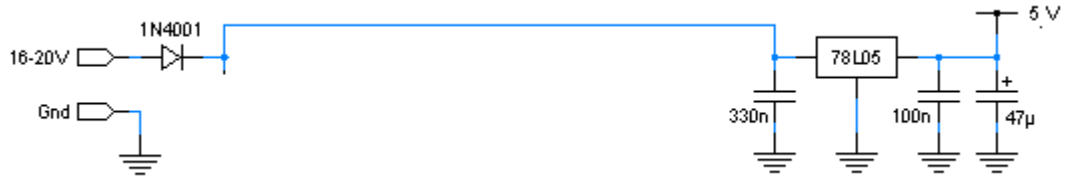


Figure 26. 5V power section of the programmer circuit (From Ref. [44])

This part of the circuit tells us that first and foremost, the hardware programmer needs to be powered with 16-20V. This is one potential problem with the usability of the circuit, since it is rare that a general-purpose adaptor can provide more than 12V, which means that we either need to use a lab power source to make the programmer work, or say two 9V batteries in series. That is in any case of minor importance, since in the context of this project, programming the microcontroller is a one-time operation (although in the sense of developing and changing the program, multiple operations had to be undertaken during development). The operation of this part can be summed up like this: when 16-20V DC power is applied, the 1N4001 diode is turned on, and hence drops the input voltage by about 0.7V (e.g. if the input was 17V, which is applied to diode anode, the potential at the diode cathode is about 16.3V). The input capacitors are charged until they reach this voltage, at which point DC current doesn't flow through them. The 78L05 device is a version of the 7805 integrated circuit. The 78XX family of integrated circuits represents three terminal positive voltage regulator devices, whose function is to "convert varying input voltage and produce a constant regulated output voltage[59]" and is produced by several manufacturers. The part number digits help with identification: "the most common part numbers start with the numbers 78 or 79 and finish with two digits indicating the output voltage [59]", hence 7805 will stand for a positive 5V regulator. Each such device has an input pin, which can accept up to 35V, a common pin that needs to be connected to a reference to ground, and an output pin, that produces the regulators voltage (in this case 5V). Let's also note that the L version (as in 78L05) is a low power

version, which can provide up to 150 mA, and has a smaller packaging (which probably makes it preferable for use in battery powered applications). Let's not forget that the role of the 1N4001 diode is to provide protection in case of inverse polarity – that is, if the 16V power supply is connected wrong.

We should also notice the capacitors role: "For maximum voltage regulation, adding a capacitor in parallel between the common leg and the output is usually recommended [...] This eliminates any high frequency AC voltage that could otherwise combine with the output voltage[59]". These capacitors in parallel, essentially represent small resistances for AC voltage, and are known as *bypass capacitors* or *decoupling capacitors*, which usually are "employed to conduct an alternating current around a component or group of components. Often the AC is removed from an AC/DC mixture, the DC being free to pass through the bypassed component[60]". In a sense, we can also see them as reservoirs, which will come up with extra charge, if the demands on the regulators at the moment happen to rise, thereby keeping the voltage stable. These capacitors are crucial for implementation whenever we have HF operation, however there are different issues to be considered in relation to the geometry of the PCB; generally we want them as close to IC pins as possible (see [61] for more). In this case, the circuit features, besides the 330 nF bypass at the input, a parallel connection of an electrolytic and regular capacitor, 47 µF and 100 nF at the output (the 330 nF can be also seen as coupled to the input electrolytic from the 13V regulator section).

The 13V regulator section is the almost exactly the same design with a voltage regulator:

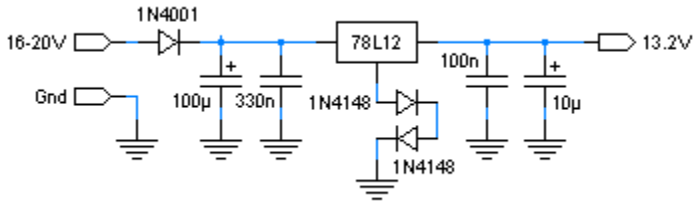
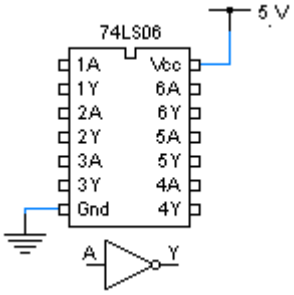


Figure 27. 13.2V power section of the programmer circuit (From Ref. [44])

The principle of operation is pretty much the same as the 5V section, except for the series connection of two 1N4148 diodes. When power is applied, current starts flowing through the common pin of the 78L12 towards ground, which turns both

diodes on. As they will fix around 0.6V on their terminals when turned on, these voltages add, and we can consider the common pin to be on around 1.2V. The 78L12 should provide a regulated output of 12V, but let us remember that those 12V are provided in relation to the common pin potential, which is usually grounded (0V); however, since the common pin is now fixed at 1.2V (due to series diodes), the regulated output voltage is raised by the same amount, resulting in about 13.2V at the output.

These power section blocks are obviously not very complex, are easy to implement, and their operation is rather reliable as well. As such, they will be re-used as power section blocks in the prototype as well.



Let's consider the other parts of the circuit now. First we should be aware of the presence of the 74LS06 integrated circuit. The 74XX is the family of TTL (transistor-transistor logic) logic integrated circuits, which operate within the TTL levels stated previously for the parallel port, which we can repeat here:

"An electrical "high" (on a pin or line) is TTL high, +2.4 to +5 volts.
An electrical "low" is TTL low, 0 to +0.8 volts.[57]"

The device is powered with 5V as other TTL logic, generated by the described voltage regulator. The 7406 is called a "Hex inverter buffers / drivers with high-voltage outputs" and features 6 inverter buffers on the chip, with a standard layout noted on the schematic (note that A designates an input, Y designates an inverter output). Again many producers manufacture it. The single inverter operation consists of the following: when a high electric signal is applied to an inverter input, the output is forced to a low level input. When a low electric signal is applied to an inverter input, the output is forced to a high level; however, due to the open collector design, it effectively floats – allowing that a higher voltage level is set through external components. That would allow us to use the parallel port pins to create both a 13V and a 5V programmable power supply, since by using the inverter, we also perform buffering – meaning that we electrically isolate the parallel port pins from the rest of the hardware. The high-voltage output means that the 7406 can well sustain the 13V

needed for VPP. With this in mind, we can now see how these inverters are implemented for individual pins.

Both the data and clock lines share a similar circuit – with the modifications, it looks like this (it's the same circuit for the data reading via ACK as well):

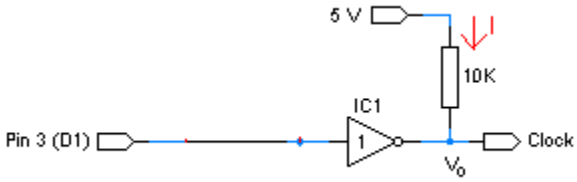


Figure 28. Data and clock lines conditioning circuits

Basically, the parallel port pin (in this case D1) is brought to the input of an inverter from the 7406 (in this case 1). The inverter performs the operation as described above – for a high parallel port pin level (which is usually around 3V) it will force a 0V on its output and thus on the clock line; current will flow through the 10K resistor from the 5V source and will close towards the ground through the converter output. For a low parallel port pin level, the output will effectively float; hence the current through the 10K resistor will be forced toward the microcontroller (in this case, the clock pin). As we can assume that the microcontroller pin inputs probably represent an input of an operational amplifier, thus having big (almost infinite) input resistances, we can safely assume that the current through the 10K resistor will be very small (almost 0). The potential of the output line will be defined as

$$V_0 = 5V - I \cdot 10K \qquad \textit{Eq 1}$$

and it is hence defined through the 5V source, and since the current I through 10K is almost 0, the output potential is almost 5V.

While this procedure is enough to provide control over the data and clock lines of the microcontroller, it is not enough when dealing with VDD and VPP lines, due to their stricter power demands.

Hence the circuit that drives the VDD and VPP lines looks like this (with the modifications):

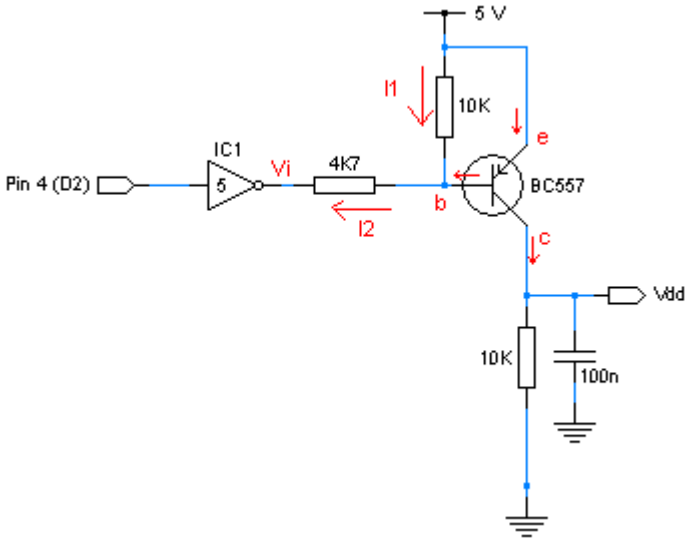


Figure 29. VPP and VDD lines conditioning circuits

The operation of the inverter is the same as described so far. So, for a high parallel port pin level, the inverted output Vi is forced to 0V. Thanks to this, current I2 starts flowing through 4K7 resistor, which is forced from the 5V source. Basically, I2 would be composed of the base current and I1, but we can consider the base current to be very small – almost 0. Considering the current law for that node

$$I_1 + I_b = I_2 \qquad \textit{Eq 2}$$

and taking Ib to be zero, it comes out that in this case I1 and I2 are almost the same current. So, we can calculate the potential of the base in this case through a voltage divider:

$$V_b = \frac{4K7}{4K7 + 10K} \cdot 5V = 0.319 \cdot 5V = 1.6V \qquad \textit{Eq 3}$$

That means that the emitter base voltage of the transistor $Veb = Ve - Vb = 5V - Vb$ is in this case about 3.4V. As a transistor needs only about 0.7V to get turned on, this means that the transistor is turned on, and in addition it is

saturated. When a transistor is saturated, it emitter-collector voltage tends to go to zero, hence the output of the circuit – the collector potential which is then brought to Vdd – remains at around 5V, or algebraically

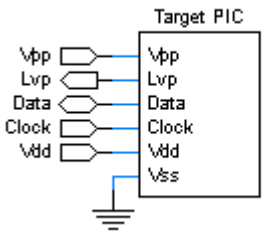
$$Vc = Ve - Vec \approx 5V - 0V \approx 5V \qquad \textit{Eq 4}$$

Hence, when the parallel port input is on a high electrical level, VDD microcontroller pin(s) is via Vc also set to a high electric level.

Similarly we can calculate the situation when the parallel port pin is low. Then, the inverter output Vi basically floats. Current I1 cannot continue as I2, and the only way it can go is towards the base of the transistor. As this is the opposite direction of the direction of a base current that opens a PNP transistor, this effectively shuts the transistor off. As the transistor gets off, I1 tends towards zero, and so does the voltage on the 10K resistor – which is also the emitter-base voltage. As the emitter-base voltage goes towards 0, the transistor is even further pushed towards switching off. With the transistor off, Vdd is only connected to ground via the 10K resistor, and hence it is set to 0V.

The exact same analysis holds for the VPP circuit, except that there we have 13V power source instead of 5V. Let's also notice that for the simple inverter circuits (data read and write, and clock lines) the microcontroller value is inverted from the parallel port pin value, whereas for the inverter plus transistor circuits (VDD and VPP), there is no inversion. This has to be taken account for in the programming software – most of the given software programming platforms support such tweaking. This also allows usage of a 7407, which has buffers instead of inverters, the layout is otherwise the same instead of a 7406 – the circuit operation will change in relation to inversion, but this can be accounted for in the programming software.

This concludes the basic analysis for the electronic design of the hardware programmer - let us just mention that Microchip considers programmers that cannot test the read/write operation at different VDD levels, to be programmers of "prototype" or "development" quality, but not of "production quality". Once we have obtained the conditioned



signals for VDD, VPP, data and clock, they are brought to the corresponding microcontroller pins, as displayed on this part of the schematic.

That is actually a good metaphor for the construction of the programmer – namely, all electronics can be on one PCB, whereas the microcontroller can be on another PCB, and they can be connected through a say, 6 wire cable. Hence, the same programmer can be used with other Microchip PICs as well, with a different type of pinout. A one sided PCB layout was drawn in ExpressPCB (a free PCB layout software), which followed this two piece design:

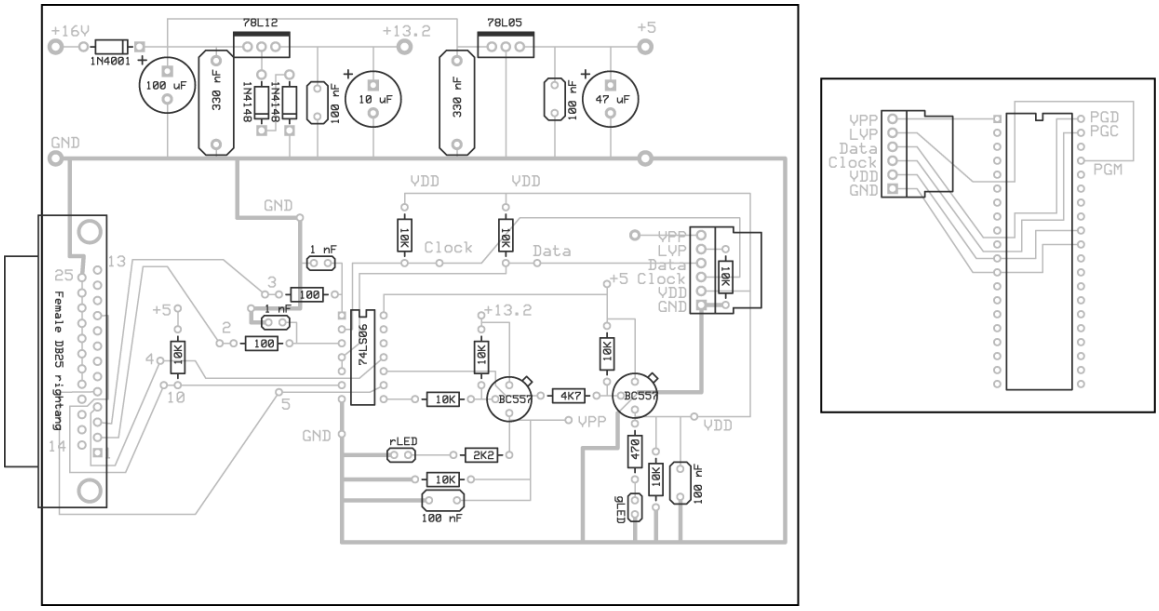


Figure 30. PCB layout of the hardware programmer, with components

For the connection between the two PCBs, a cable was made out of 6 wires and 6 pin Molex connectors. There was a design mistake that placed the PGM (LVP) pin on pin 37 instead of pin 36 where it belongs. Even a worse design mistake was to use very thin tracks while designing the PCB. This meant that the acid etched into the tracks, making microscopic cracks that broke the track connections.

Eventually, it was possible to find all the cracks, and to correct them by soldering them over.

Figure 31. The back side of the hardware programmer developed in this project, indicating a broken track problem

The small PCB track problem was rather difficult; yet it was possible to troubleshoot it mostly due to the kind assistance of the members of the Electro Tech Online forum on the Internet [63]. They provided a very good method to do a low-level hardware test of this programmer: "

1. Do not connect to PC and do not place PIC into socket.
2. Power up and confirm +5V and +13V healthy
3. Prepare a 100 ohm resistor and connect one end of it to +5V
4. Connect voltmeter to corresponding PIC socket pin and touch the other end of the resistor to the programmer's parallel port input pin. Watch if the corresponding PIC socket pin change from low to high or vise versa. Log the result.
5. repeat the test using one end of the resistor connects to 0V. [63]"

In retrospect, although the particular production of the hardware programmer in this project was very fragile – barely reaching the prototype level, it was possible to fix it, and it did manage to perform the function well – that is, to program the microcontroller, in conjunction with the IC-Prog programmer software. A slightly

better PCB layout design, would provide a much more stable and reliable programmer hardware.

Once the programmer is working, it is possible to observe the different signals that are produced using an oscilloscope. Let us note that a basic analogue oscilloscope, with only the possibility to trigger a signal retrace on zero crossings, is not of much use in this context: only the clock signal can be observed with it. Since the data signal essentially pulls out 0s and 1s at "random", we will not be able to fix a proper zero crossing for the data signal on an analogue oscilloscope, and we could never obtain a clear picture. In this case, it is of essence to have access to an oscilloscope with memory. There were several such oscilloscopes available during development of this project, which besides remembering a certain trace, could also print it out, which made the troubleshooting process easier.

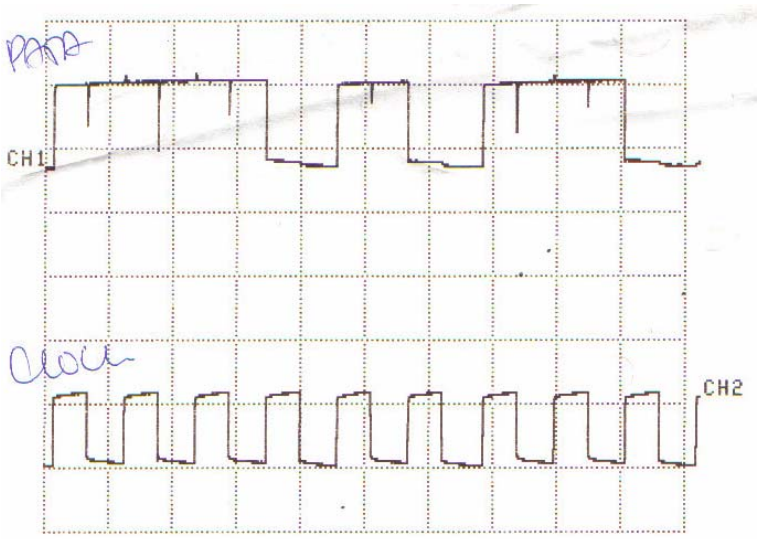


Figure 32. Actual oscilloscope trace from data and clock lines of the programmer during programming

The waveforms displayed on Figure 32 are produced by the software programmer, effectuated on the parallel port pins and thereafter processed by the programmer as described. This image is very easy to relate to the time diagram rendered in Figure 23 –We should mention here that in spite of the fact that we are using the parallel port, what we have discussed is actually a real world hardware example of synchronous serial communication. Namely, in serial communication, each bit of information is allocated certain time frame. Both the transmitter and the receiver need to be aware of it, in order to properly transfer the information, and the

way this is addressed, makes the primary distinction in serial communication between synchronous and asynchronous modes. In an asynchronous mode, the transmitter and receiver need to be "set" to a given communication speed before the communication begins – in that sense, they do not need to communicate the time frame information between each other, which means that one channel of information can be sent through one wire (plus the ground required to complete the circuit). Note that the RS232 PC serial communication, that we are going to use in the DAQ prototype, is an asynchronous mode of communication; we shall take a closer look at it in a while.

On the other hand, when using synchronous mode, a clock signal is separately sent, so one needs at least two wires – one for data and one for clock – to transfer information (plus the ground); yet the receiver does not need to know the communication speed beforehand, it can just synchronize with the received clock: "The principle difference between the synchronous and asynchronous modes of transmission is that in the synchronous case, the receiver uses a clock which is synchronised to the transmitter clock [64]". That is precisely the case that we have implemented in the hardware programmer, and is fundamentally different from the asynchronous serial communication we are going to implement as part of the data acquisition hardware.

4.5 Microcontroller software – the data acquisition program

Now that we have provided hardware and software means to code and program the PIC 16F74, we can proceed and discuss the actual program residing on it, which determines the function as a data acquisition hardware.

In essence, the situation is that the 16F74 has eight analog inputs. What we want to do is obtain their individual digitized values, format those values in some way, and then send them through serial. This operation will take some finite, yet short time, so the functionality of the device as DAQ hardware would be simply to endlessly repeat this process. Provided that the operation time is short enough, the received values will be refreshed with that rate, and we would have obtained a real-time acquisition of the sensor data. The code listing is provided in Appendix B; here we are going to discuss some of the algorithm in more detail.

Before we start with coding, we should be aware that we need to communicate with some special registers when we want to perform A/D conversion. Registers can be seen as special function variables of the microcontroller (although they have hardware specifics as well), by assigning a certain number to them, we can set a particular setting of the microcontroller. Full description of the registers is given in the microcontroller datasheet. First, we should be aware that groups of I/O pins of the PIC are grouped into ports (which can be identified from the pinout as well). As the I/O pins can be directional, there are registers that control their directionality, for instance: "PORTA is a 6-bit wide, bi-directional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= '1') will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a Hi-Impedance mode). Clearing a TRISA bit (= '0') will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin) [42]". For instance, assigning the decimal value 56 to TRISA, which is 111000 in binary, well set the pins 4, 5 and 6 of port A (pins 5, 6 and 7 of the PIC) as inputs, and the rest of port pins as outputs. Similarly, the TRISB register controls the directionality of port B. In addition, there are three registers that control the analog-to-digital conversion operation: "The A/D module has three registers. These registers are:

- A/D Result Register ((ADRES)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 ((ADCON1)

The ADCON0 register, controls the operation of the A/D module. The ADCON1 register, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be a voltage reference), or as digital I/O. [42]".

We can also use the BASIC commands High and Low to individually set a bit within a register, or control an individual port pin voltage; the syntax goes in this case [command] portName.pinName, where the pin names are internally defined for this version of BASIC (from the Oshonsoft IDE). A pin (bit) number can also be used instead of a name.

The process of analog to digital conversion is entirely implemented through usage of the ADCON0 register. Basically, we set the desired conversion channel, initiate the conversion clock, initiate the process and wait for indication that the process has finished, all by reading and writing bits of the ADCON0 register, which are:

bit 7-6 ADCS1:ADCS0: A/D Conversion Clock Select bits
bit 5-3 CHS2:CHS0: Analog Channel Select bits
bit 2 GO/DONE: A/D Conversion Status bit
bit 1 Unimplemented: Read as '0'
bit 0 ADON: A/D On bit

When the analog to digital conversion process has finished, the result is available in the ADRES register (A/D Result Register). So, to do a mutlichannel A/D operation, we simply have to perform analog-to-digital conversion to all available analog input pins one by one, saving the contents of the ADRES register as we go along (else it will always get replaced) in an array. Once this operation is finished, the array will contain the last available sensor values for all channels.

Based on this array we can format a string of characters, which will be sent via serial. Let us note here that although the microcontroller has a hardware UART for serial communication, which utilizes pins RC6 and RC7 on the PIC, this program is implemented through a software serial simulation, which is a part of the Oshonsoft BASIC interpreter, which is the command Serout. In this way, any output digital pin can be used to generate a serial code, and here it was arbitrarily chosen to be RB4. It can be suspected that better results might be obtained by using the hardware UART, however, not much time was available for the software development, so it pretty much stopped as soon as a decodable serial stream was obtained, and Serout did the job.

So, in essence, we can conceptualize the data acquisition program through the following pseudocode, with the functionality performed in the endless process loop:

```
Initialize PIC

loop forever

      erase out array
      read ADC 1
      store ADC 1 value in out array
      read ADC 2
      store ADC 2 value in out array
      ...
      read ADC 8
      store ADC 8 value in out array

      format string based on out array
      send string based on out array

end loop
```

The initialization process is performed with the following commands:

```
Symbol ad_action = ADCON0.GO_DONE  'set new name for A/D conversion start bit
Symbol display = PORTB  'set new name for PORTB used to display the conversion
result

Define SEROUT_DELAYUS = 500  'define serial delay
```

The serial delay is a delay between each character sent

```
Dim channel_id As Byte  'declare a variable
Dim channel_val As Byte  'declare a variable
Dim data(8) As Byte
```

The 8 byte long array `data` is out output array

```
TRISB = %00000000  'set PORTB pins as outputs
TRISA = %111111  'set PORTA pins as inputs

ADCON0 = 0xc0  'set A/D conversion clock to internal source AND SELECT CHANNEL 0
ADCON1 = 0  'set PORTA pins as analog inputs
High ADCON0.ADON  'turn on A/D converter module
```

After the initialization, there is an 4 second long alternating sequence, where the serial output pin is alternated between high and low levels each second. This was done in order to test some performance issues with the radio transmitters/receivers, and it was left in the prototype version, since it helped deal with an issue with the radio circuits.

```
 ' just a test sequence for start
' alternate high and low one second each
High PORTB.4
WaitMs 1000

Low PORTB.4
WaitMs 1000

High PORTB.4
WaitMs 1000

Low PORTB.4
WaitMs 1000
```

Thereafter we enter the main routine, which repeats forever. First we perform the analog to digital conversion of all channels, and storage in output array, in a for/next loop.

```
main:

    'acquire data, collect into packet data array

    For channel_id = 0 To 7 Step 1  'for-next loop

        ADCON0 = ShiftLeft(channel_id, 3)    'SELECT CHANNEL channel_id and
```

Let us remember that we express which channel to process by writing that number in bits 5-3 (Analog Channel Select bits) of ADCON0. Le'ts say we want to set channel 5 for processing, the binary representation of 5 is 101, and it takes exactly 3 bits. This value needs to be stored in bit positions 3 to 5 in ADCON0, that means something like this XX101XXX. For that purpose, we use the binary ShiftLeft operation. Shifting 101 (which is 5) by 3 places will give 101000 (which is 40 = $5*2^3$). Of course, storing that value in ADCON0 will imply 00101000, meaning all the other bits will be erased.

```
        ADCON0 = ADCON0 Or 0xc0      'set A/D conversion clock to internal source
```

For us it is important to keep the values of the A/D conversion clock (bit 7-6 A/D Conversion Clock Select bits), hence by using the Or command with the number c0 (11000000) we set those bits without disturbing the value we previously set in the Analog Channel Select bits.

```
        High ADCON0.ADON  'turn on A/D converter module, shift does not preserve the
last bit!
```

Now we also turn the ADC converter on by setting bit 0 of ADCON0 (ADON: A/D On bit) to 1 with the High command, and we go to the getadresult subroutine.

```
        Gosub getadresult  'go to conversion routine

        getadresult:  'A/D conversion routine
```

The subroutine simply starts the A/D conversion process, by setting bit 2 of ADCON0 (GO/DONE: A/D Conversion Status bit) to high. While the A/D conversion process is running, this value will remain logical one, so we simply enter an empty loop, which exits when this value is logical zero; and in this way, we wait for the A/D conversion process to finish.

```
            High ad_action  'start the conversion
            While ad_action  'wait until conversion is completed
            Wend
        Return
```

After the conversion process is finished, the subroutine returns, and the ADC conversion result is available in the 8-bit ADRES register. We simply save that value, storing it at the index of the array, which corresponds to the analog input that was sampled.

```
        data(channel_id) = ADRES  'save the result of conversion in the given pos.
in data array

    Next channel_id
```

When the for/next loop is finished, the data array is populated with the latest values of the analog input channels. Now we proceed with sending the data values using the Serout command, formatted in a word packet. Note that the Serout is a

software communication routine (which essentially performs bit banging), and the hardware UART on the PIC is not activated.

First we send the value 85 (which ASCII code for the character "U") 8 times as a preamble – this was done in order to troubleshoot certain issues with the radio devices.

```
    Serout PORTB.4, 4800, 85, 85, 85, 85, 85, 85, 85, 85
```

Then we send the values of the output array one by one as numbers, delimited by two characters, the character "@" and the character that corresponds to the chanel number. Obviously, the channel data will not always result with textual characters when the serial code is viewed as text. Finally, the packet is closed with the ";" character, followed by the CR (carriage return) and LF (line feed) characters (ASCII codes 10 and 13), which will result in a new line when the serial code is viewed as text.

```
    'direct the entire packet in one line
    Serout PORTB.4, 4800, "@1", data(0), "@2", data(1), "@3", data(2), "@4",
data(3), "@5", data(4), "@6", data(5), "@7", data(6), "@8", data(7), ";", CrLf
```

When we are done, we repeat the process again, and we loop forever …

```
    Goto main  'repeat forever
End
```

This simple program all that we need to define the operation of the microcontroller as a data acquisition hardware. Eventually, we end up with a 35 byte long word packet that contains the latest sensor data, which looks something like this

```
0           1           2           3
01234567890123456789012345678901234    5

UUUUUUUU@0³@1«@2Þ@3´@4°@5ô@6è@7ã;<CR> <LF>
```

0-7	8-9	10	11-12	13	14-15	16	17-18	19	20-21	22	23-24	25	26-27	28	29-30	31	32	33-34
'U' (85)	'@0'	Ch0 value	'@1'	Ch1 value	'@2'	Ch2 value	'@3'	Ch3 value	'@4'	Ch4 value	'@5'	Ch5 value	'@6'	Ch6 value	'@7'	Ch7 value	';'	CR + LF

This is the packet format that we need to have in mind when we write a decoder for Max/MSP. Obviously, the actual sensor information is only embodied in

characters with positions (starting from 1) 11, 14, 17, 20, 23, 26, 29 and 32 within the word. Basically, we need a minimum of 8 bytes, yet we have 35, which is a lot of redundant information. Still, some of that redundancy was aimed at troubleshooting the operation of the radio devices, and in any case, it can be used for a simple form of error detection, which would drop the received packet, if one of the redundant characters is not received correctly.

As the program loops, new word packets are being sent out, and a sequence of packets is received:

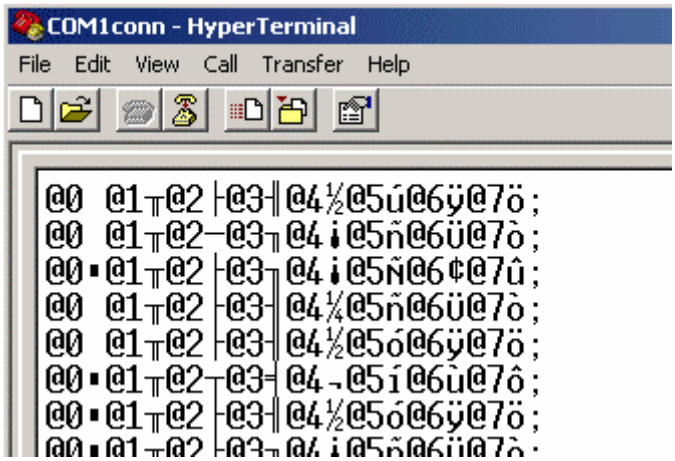


Figure 33. Received packets as reported by Windows HyperTerminal with an earlier format of the packet

Each word packet allows that the latest sensor values are extracted from it, hence the time it takes to generate and send one packet will determine the refresh rate of the sensor data acquisition. This time can be obtained by simulating the program in PIC Simulator IDE. However, before it is simulated, it needs to be compiled first. As a result of the compilation, we obtain several files from PIC Simulator IDE. One of them has an extension .asm, and it contains the assembler translation of the basic program. Its contents include the commented original basic lines, as well as their assembler translations, like:

```
; 12: TRISB = %00000000 'set PORTB pins as outputs
 BSF STATUS,RP0
 CLRF 0x06
 BCF STATUS,RP0

; 13: TRISA = %111111 'set PORTA pins as inputs
 BSF STATUS,RP0
 MOVLW 0x3F
 MOVWF 0x05
```

```
BCF STATUS,RP0
```

Assembler is already on the machine language level, but written in human readable mnemonics. In addition to that, we obtain the HEX file as well, which contains the same machine level data:

```
:100000008A110A120528000009008316860183124E
:1000100083163F3085008312C0309F0083169F01F6
:1000200083121F14B0013008073C031C2F2830082E
```

only formatted so it's optimized for use with software programmer software and hardware. The contents are obviously not human readable.

Once the compilation process is competed with success, the program can be simulated. To find out the effective refresh rate, we can set a simulation breakpoint at the BASIC program line

```
Serout PORTB.4, 4800, 85, 85, 85, 85, 85, 85, 85, 85
```

Actually, in PIC Simulator IDE, we can only set breakpoints at assembler commands, so we can set the breakpoint at the first assembler command that makes the BASIC command, here it would be the `BSF STATUS,RP0` command:

```
0139    0031           ; 60: Serout PORTB.4, 4800, 85, 85, 85, 85, 85, 85, 85, 85
0140    0031    1683        BSF STATUS,RP0
```

We can thereafter start the simulation process. The simulator indicates how does the assembler execute, how much time has passed since the start of the simultation, and the values of the registers.

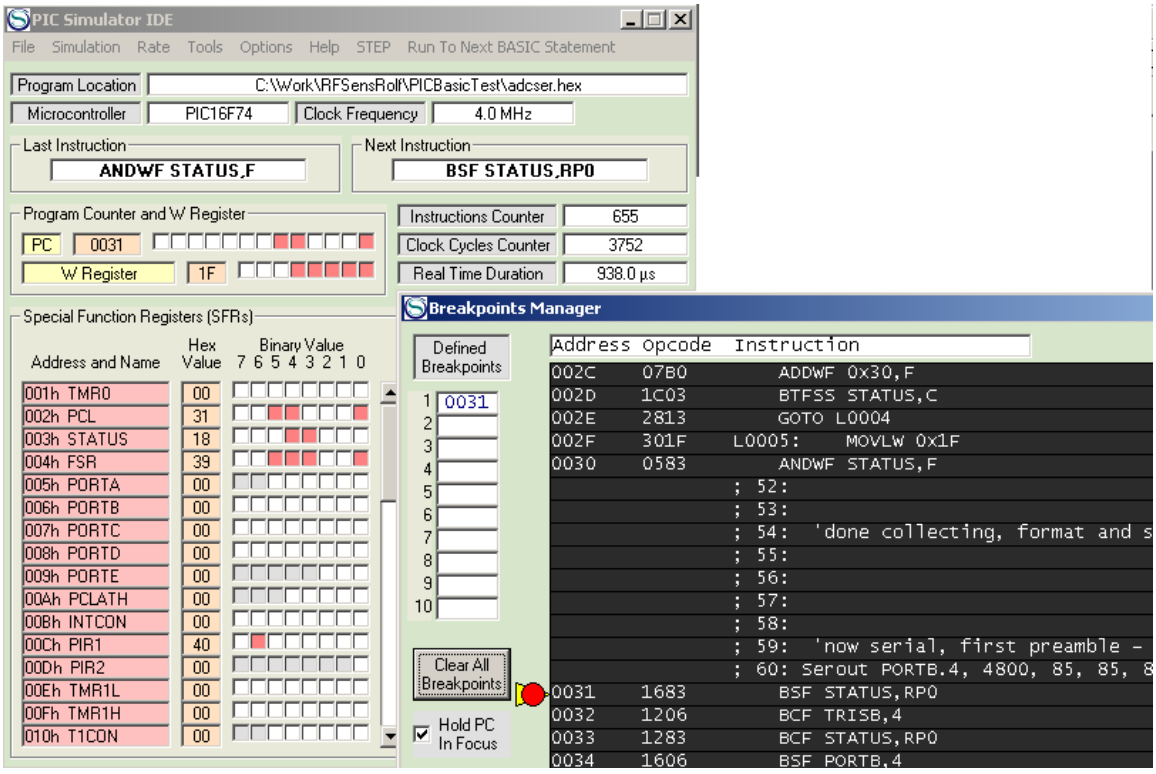


Figure 34. PIC Simulator IDE in processing a breakpoint in a simulation

We can note down the time expired when the breakpoint is encountered for the first time, and when the breakpoint is encountered for the second time. The time difference between these events will give us a measure for the effective refresh rate. Such measurements resulted with the following:

First breakpoint at 938 µs,
next at 85604 µs,
difference = 84666 µs = 84.6 ms

This means that the time for creating and sending of one packet is 84.6 ms, and that is also the effective refresh rate of a sensor channel update with this program. Although several things could be undertaken to improve the performance of the software, the important thing is that by having the possibility to write and compile this program, and program it into the 16F74 PIC, we have implemented the data acquisition functionality of the microcontroller.

The next step is thus, to implement a test data acquisition circuit, which will enable the microcontroller to execute its program and to communicate the results to a PC; and eventually, to implement a wireless version of the data acquisition circuit. If all goes well, the only remaining thing to do would be to write a decoder for Max/MSP.

5 Serial communication and RS-232

5.1 Introduction to serial communication

Since we are going to utilize serial communication with a PC, we shall use this section to quickly reiterate some of its basic concepts. There is a wealth of information and tutorials regarding serial communication on the Internet; some of those will be used in the following discussion.

Before we start discussing serial communication, let us state that we are talking about communication between electric devices. In its most elementary form, we can illustrate communication through the elementary electric circuit.

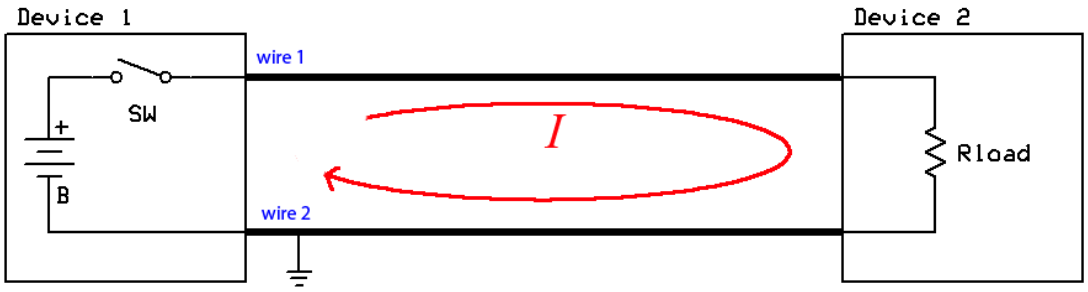


Figure 35. Elementary circuit as a simplex communication channel

In essence, the elementary circuit is consisted of a generator (the battery B) and a load (the resistor R), as well as the wires that connect them. We have an elementary circuit when the connection between generator, load and wires forms a closed path though which electric current can flow.

We can implement a switch in this circuit, and formally conceptualize the switch and the generator as one device, and the load as another device – as shown on the figure. Such conceptualization implies that such devices must have two terminals (connecting points to the outside world) each – allowing that a closed path can be formed by connecting the terminals with two wires.

When the switch is closed, a closed path is formed, and current I starts flowing. In this case, the generator provides the energy, and the load consumes it – which can be determined by the fact that we can measure voltage at the ends of the resistor. There is exchange of energy, and in that sense, also exchange of information. We can also say that a signal is sent via wire 1, whereas wire 2 represents the return path for a signal. When the switch is closed, no closed path exists, and so current cannot flow – hence, no voltage would be developed at the end of the load. We could emphasize the different roles of the devices in terms of power – that is, the generator provides energy, the receiver consumes it. In that sense, we can initiate information exchange only through the generator, so in this context, device 1 would be transmitter, and device 2 receiver of information. Thus, the communication we can establish with an elementary circuit, is inherently one-way, makes a distinction between transmitter and receiver, and requires two wires for one channel of communication (as a "a pathway over which information can be conveyed[65]"). Since the direction of message transmission cannot change, this is also referred to as a simplex channel of communication.

Hence, such a basic system lends itself for use in communication – provided that the devices are separated with some physical distance, and connected with a two-wire cable, they can be used for information transfer. For instance, the load could be a light bulb; hence a human sender by turning the switch on and off could transmit a message to a human receiver, who would interpret the flashes of the light bulb; the same principles are the basis for telegraphy as well. In those early days of communication, it was discovered that the Earth ground can be used as a return line, hence enabling communication through only one wire: "By 1844, Morse's line between Baltimore and Washington consisted of just two wires, one carrying the electrical current for signaling, and the other acting as a return line, to make a complete circuit. However, it turned out that even that could be simplified, and the return wire eliminated, if the sending line was 'grounded', i.e. physically connected to a plate buried in the earth. The ability to eliminate the return wire was something of a mystery at the time, and the phenomenon became known under the misnomer of the 'ground return', since it was incorrectly thought that the return electrical current was somehow flowing through the ground all the way back to the sending location[66]". Thus, although two wires are required for one communication channel, only one is enough to implement it, provided there is a return path for the signal that closes the circuit.

That becomes quite obvious if we try to model a simple system based on the elementary circuit that would allow bi-directional communication:

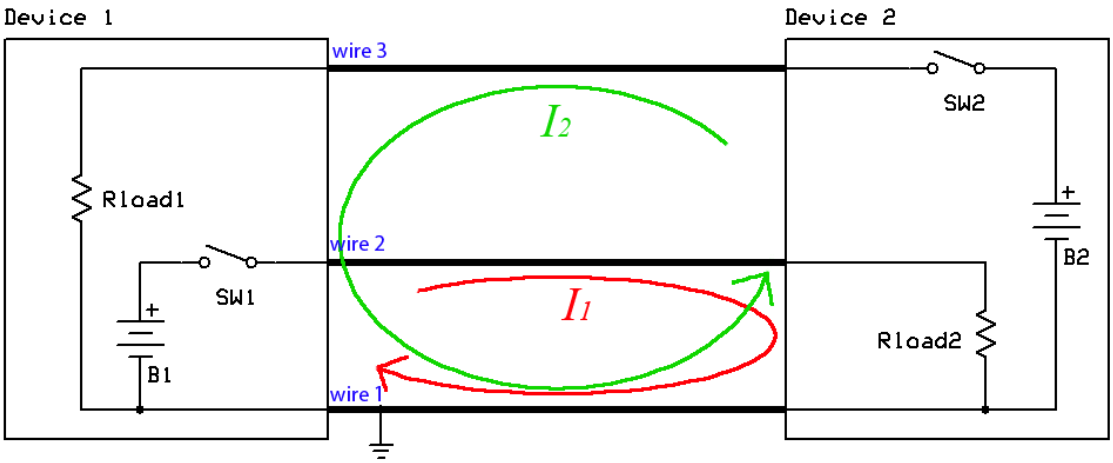


Figure 36. Two elementary circuits (simplex) as a full-duplex communication channel

Each of the human actors in the communication needs to have the facility both to receive messages (observe the light bulb) and initiate sending of messages (use the switch). Hence each communication device now contains both a transmitter and a receiver (in relation to the elementary circuit), so two independent circuits can be established. Literally following the elementary circuit, we would need four wires for such a connection; yet it is not necessary. Actually, the return path can be shared so it can be a single wire.

Here is where the metaphor of a channel and return path is useful, since now we can consider each channel of communication as one wire, provided there is an additional return path wire. Using this logic we can deduce that minimum number of three wires is required for a bi-directional communication as rendered on the figure. If we consider that there are two communication devices (and two human actors), we can perceive the three-wire connection as a single channel of communication that allows “simultaneous message exchange in both directions[65]”, also known as full-duplex (an analogy for full duplex would be ”a telephone conversation when both persons are talking at the same time[67]”). It is obvious from the figures that a full duplex is composed of two simplex systems. Finally, it is possible to use two wires to establish bi-directional communication, provided there are capabilities to change the direction of transmission; such a system is known as half duplex (the analogy of which

would be “the use of a CB-radio where two-way communication is possible but only one person can speak at a time [67]“). This distinction of communication channels is rendered on the figure:

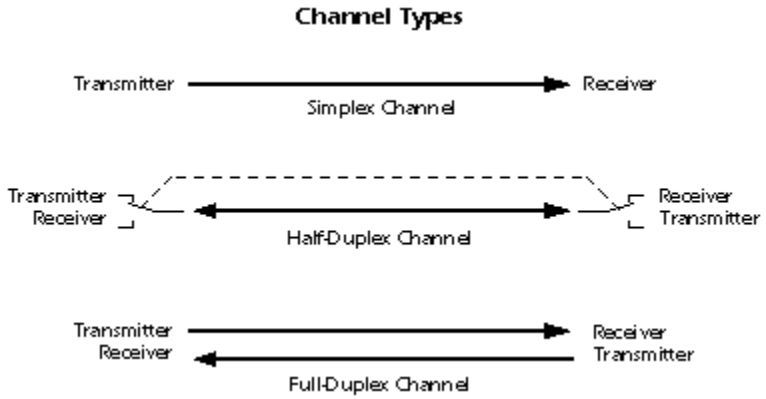


Figure 37. Communication chanel types (Ref. [65])

Let’s now consider a case where the communication devices involved are closer to our daily experience. For example, we can consider the communication between a computer and a printer, where the computer communicates the character ‘a’ to the printer. We are aware that in the digital domain characters are identified by ASCII codes, so the code for ‘a’ is 97, or 01100001 as an 8-bit binary number. Provided that we can communicate a binary 1 with high voltage and binary 0 with low voltage, we basically need to communicate 8 different voltage values from the computer to the printer, in order to transmit the character ‘a’. The most obvious approach is to use 8 wires (transmission lines) for each bit value, plus the implied return path:

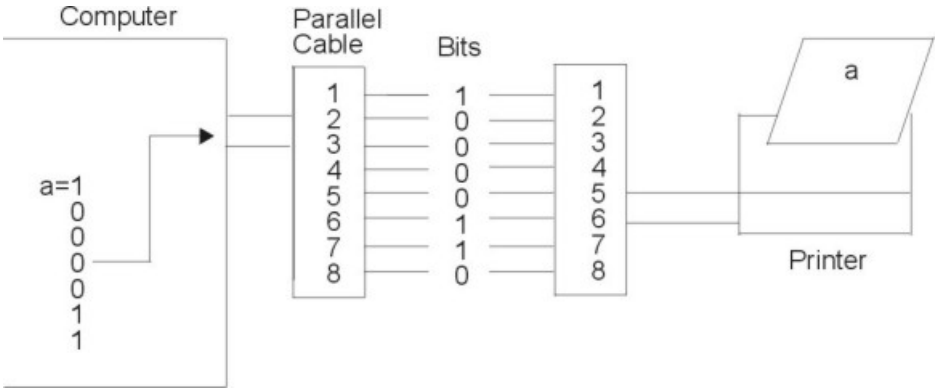


Figure 38. Parallel communication between computer and a printer (Ref.[67])

This mode of communication is known as parallel communication, since independent transmission lines imply that we can transfer multiple bit values simultaneously. Considering that 8 bits form a byte, this means that in this way we can transfer one byte a time. Obviously, this is a quick method of communication, but it demands a lot of wires.

We can transfer the same information using only one transmission line (plus the implied return path) if we send the bits sequentially – in series: "the data is converted from a parallel form (sent by the computer), to a sequential form, where bits are organized one after the other in a series. The data is then transmitted to the device with the least significant bit (or zero-bit) sent first. Once received by the remote device, the data is converted back into parallel form[67]".

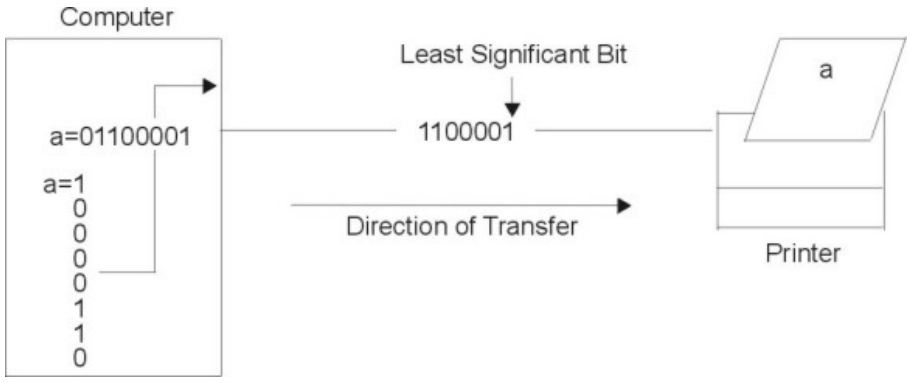


Figure 39. Serial communication between computer and a printer (Ref.[67])

This represents the essence of serial communication; to send multiple bits of information through one transmission line, one bit at a time. It is obvious then that in this case, all that distinguishes them from one another is their position in time – and this also implies that the time duration of all bits must be the same. As we relate a bit of information as a certain voltage state, what we would observe on a transmission line in case of a serial communication is simply some voltage changing in time, seemingly hopping between low and high voltage levels at random. To distinguish bits of information in such a context, we basically need to know two things: when did the communication start, and what is the time duration of one bit (how much time does each bit take). Actually, deducing the time duration of a bit just by looking at the changing signal is quite difficult, since in essence, values might alternate at random, and hence we would get sequences of 0s or 1s that have different durations: "The receiving system does not know where one character ends and the other begins[67]"

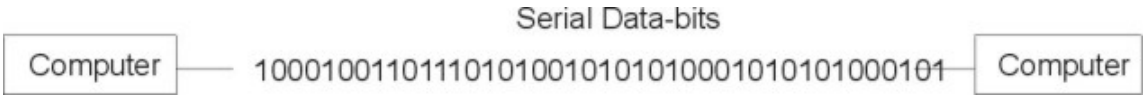


Figure 40. Serial transmittion of a long, seemingly random binary sequence (Ref.[67])

Thus, it is imperative that both sides in the communication agree about these parameters (bit duration [speed] and start) of the serial communication. Hence, the problem of timing and synchronization arises as a primary issue in serial communication. Namely, timing signals in electronic devices are obtained through oscillators – the oscillator frequency is basically the rate of the clock that oscillator represents – and can be used to define a bit duration. In essence, we can say the transmitter and the receiver agree on the bit duration (transfer speed) by synchronizing their internal clocks. However, the frequency of an oscillator is electrically influenced by external parameters, such as temperature – so, it is easy to imagine a situation where two devices, with oscillators otherwise set to the same frequency, might end up oscillating at different frequencies due to various influences, generating different clock rates for the bit duration (which makes serial communications impossible). Hence, the issue of synchronization is central in serial communication, and it distinguishes between two common modes of serial communication: synchronous and asynchronous.

In synchronous serial communication, both devices must synchronize their clocks according to some master signal: "a transmission procedure by which the bit and character stream are slaved to accurately synchronized clocks, both at the receiving and sending end [68]", "type of transmission in which the transmission and reception of all data is synchronized by a common clock and the data is usually transmitted in blocks rather than individual characters [69]", "with synchronous communications, the two devices initially synchronize themselves to each other, and then continually send characters to stay in sync [70]". What this synchronization issue means in practice is basically that we need two separate transmission lines – one to carry the data, and one for the clock: "Synchronous serial communication uses two wires, one for the data and one called the clock that controls the rate at which the data is sent. One device acts as the master and sends a timing pulse on the clock wire every time it is ready to send or receive a bit of data. The term shifting is used for synchronous serial communication, because with each clock pulse you shift over next bit of data [71]". Hence, we recognize that the process of hardware programming of

the microcontroller which was discussed previously, is a real-world example of synchronous communication between the PC (through the parallel port and the programmer) and the microcontroller – as both data and clock lines are implemented, whose signals are shown on the oscilloscope trace in Figure 32. Hence, in this context, we need at least three wires (clock, data and return path/ground) for implementation of a synchronous serial communication channel. Note also that synchronous serial communication implies that data can only be sent in sync with the clock.

5.2 Asynchronous serial communication – PC serial

The other mode of serial communication is the asynchronous mode, and this is the mode used in regular serial communication on PCs. We shall provide some details about it here, as this is the communication method that is implemented in the serial communication between the microcontroller as a DAQ hardware and the PC in this project. Opposite to the synchronous mode, here we do not have a clock line, and hence no synchronization – in effect, this truly enables simplex communication through one wire (plus the implied return path). Hence, we need only one transmission line for one channel of asynchronous serial communication. However, since we do not have a separate clock line, we would now have a problem identifying the bit duration (communication speed), and distinguishing between the individual bits in general. Hence, in asynchronous serial communication, both devices need to be set to the same clock rate separately (in essence, the human users set them) – and in order to synchronize the communication process, the only other parameter needed is the start of the communication. Since there is no clock, the transmitter can initiate the start of communication at any time – thus we need a well-defined protocol for initiating communication in this case.

When discussing this protocol, we observe a simplex communication channel between a transmitter and a receiver, and we begin by recognizing two states on the asynchronous communication line: logic 1 and logic 0 (let's note that at this time they do not necessarily mean low or high voltages). We also recognize that the bit duration is determined by the communication speed: "The baud rate refers to the signalling rate at which data is sent through a channel and is measured in electrical transitions per second. In the EIA232 serial interface standard, one signal transition, at most, occurs per bit, and the baud rate and bit rate are identical. In this case, a rate of 9600 baud

corresponds to a transfer of 9,600 data bits per second with a bit period of 104 microseconds (1/9600 sec.)[65]". The bit duration is made known to both actors in advance: "the transmitter and receiver must be preset in advance to an agreed-upon baud rate[65]".

The analysis starts with the line being idle (nothing is being sent); this is known as a 'mark' state and has the value of logic 1, to distinguish from a disconnected line. The transmitter says that it wants to send a character by changing the 'mark' state to a so-called 'space' state, which corresponds to logic 0: "when the mark state is interrupted by (a binary 0), the receiving system knows that data characters are going to follow[67]". This is a signal to the receiver that a transmission of data is about to begin, and it represents the start bit of the packet frame: "The start bit triggers an internal timer in the receiver that generates the needed clock pulses. Following the start bit, eight bits of message data are sent bit by bit at the agreed upon baud rate. The packet is concluded with a parity bit and stop bit [65]".

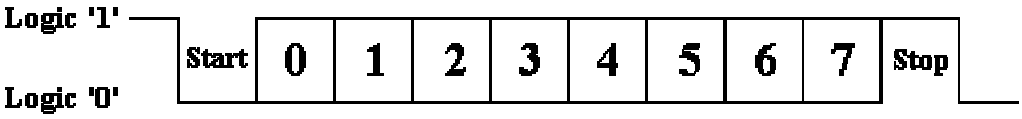


Figure 41. Asynchronous serial packet (Ref. [73])

Thus, by using start bits and stop bits, we are able to provide some sort of time synchronization: "each word is synchronized using it's start bit, and an internal clock on each side, keeps tabs on the timing [73]". The usage of start and stop bits is known as framing: "the data sent using this method, is said to be *framed*. That is the data is *framed* between a Start and Stop Bit[73]". Notice also on Figure 41, "the diagram, shows the next bit after the Stop Bit to be Logic 0. This must mean another word is following, and this is it's Start Bit. If there is no more data coming then the receive line will stay in it's idle state (logic 1) [73]".

We could say that such organization is the "reason that the start bit, which precedes the data character, is always a space bit (binary 0) and that the stop bit, which signals the end of a character, is always a mark bit (binary 1) [67]". Thus, by using these framing rules about the start bit and the stop bit, we can delimit a long asynchronous bit sequence into packets, and read the data from it:

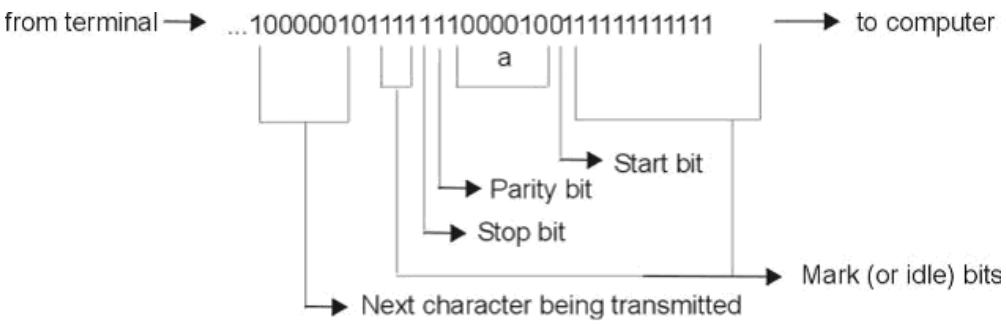


Figure 42. An asynchronous bit sequence delimited by frame (Ref.[67])

The reception process consists of sampling the incoming voltage at time intervals at the agreed baud rate, which starts when the start bit is received: "a very accurate local oscillator within the receiver will then generate an internal clock signal that is equal to the transmitter's within a fraction of a percent [65]":

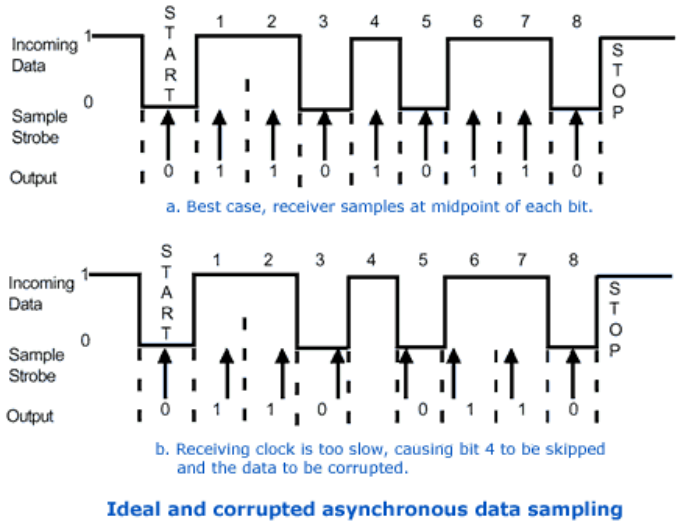


Figure 43. Receiving asynchronous serial data packet – sampling (Ref.[72])

It should be obvious from Figure 43 that if the internal clock rate of the transmitter and the receiver is not the same, we will not decode the data bits correctly. Hence we need to do some sort of error checking. For one, we know that the start and stop bits are always logic 1 and logic 0, respectively. So, we can check what is the value of the received stop bit: "should the Stop Bit be received as a Logic 0, then a framing error will occur [73]". A framing error would indicate that the data decoded previously was probably wrong, and the last character should be dropped. This is, of course, not the only method to perform error checking; a parity bit could be added to the packet,

and parity checking could be performed. Hence, a more general structure of the asynchronous serial data packet would look like this:

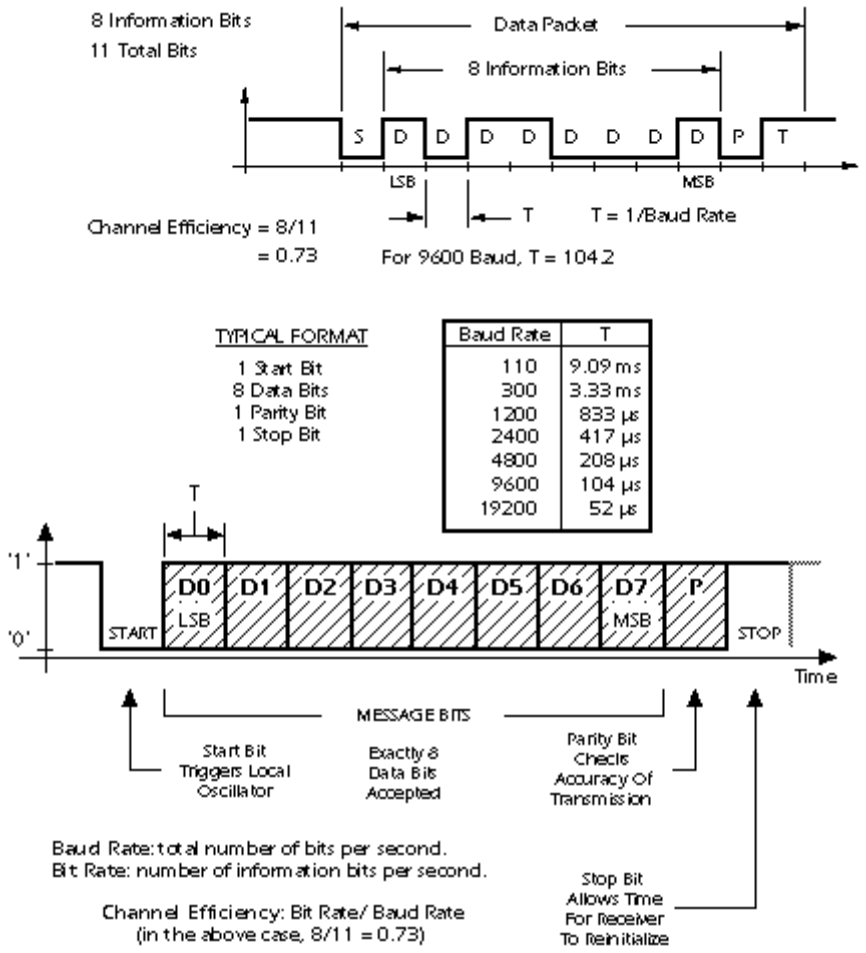


Figure 44. General asynchronous serial data packet format (Ref. [65])

We can see that the packet consists of a start bit (S), data bits (D), parity bit (P) and stop bit (T). In general, the parity bit can be implemented, but it is not necesarrry to do so; so in this sense, the format of the packet frame, also becomes a parameter of communication that must be agreed between transmitter and receiver. In practice, most of the time the parameters of a given serial communication are specified by providing (baud rate)–(data bits)–(parity bits)–(stop bits). For instance, communication at 9600 8N1 means baud rate of 9600 bps, 8 data bits, no parity bits and 1 stop bit per character frame (data packet). The format 8N1 is the most common format used, although other formats are possible, since standard values for data bits could be 5, 7 or 8; standard values for parity bit choices are even parity, odd parity,

mark parity, space parity or none; and the stop bit could occupy 1, 1.5 or 2 bit durations.

Let's also mention that in software, the software interface will usually transparently take care of all the framing issues. For instance, the Serout BASIC command that we use in the microcontroller program is a rather simple software interface: we simply write `Serout [output pin], [baud rate], [characters to send]` – the compiled code then automatically takes care of generating a start bit and the rest of the framing so it follows the 8N1 format which in the case of Serout command is preset by default. Let's also repeat that the Serout command is a software serial emulation (bit-banging) routine, which does not use the hadware UART on the microcontroller, and can target any pin. The corresponding command in the Oshonsoft BASIC interpreter that employs the hardware UART is `HSerout [baud rate], [characters to send]` – and it targets the UART TX pin, which is pin 25 (RC6) on the PIC16F74 microcontroller.

5.3 RS-232 serial - voltage levels, connectors and issues

The previous discussion represents the an introduction of the essential of asynchronous serial communication. At this level, we can establish a relationship between actual electrical signal levels that are used and the logic 0 and logic 1 levels used in the discussion so far. When we are talking about the microcontroller, we should be aware that it operates with TTL logic levels. In essence, TTL logic defines logic 0 to be low voltage (around 0V), and logic 1 to be high voltage (around 5V), within a given range of tolerance:

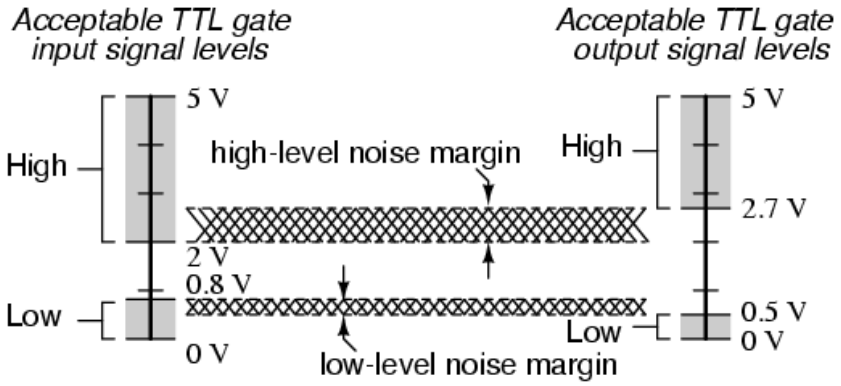


Figure 45. Acceptable TTL voltage levels for logic high and logic low (Ref. [74])

Hence, all the diagrams of serial packets presented so far, would also be valid as voltage diagrams for TTL implementation – since here logic high corresponds to high voltage and vice versa. That is, however not the case with the asynchronous serial communication with a PC through the serial port.

Communication through the serial port of a PC adheres to the RS-232 standard: "RS-232 (ANSI/EIA-232 Standard) is the serial connection found on IBM-compatible PCs. RS-232 is limited to point-to-point connections between PC serial ports and devices. RS-232 hardware can be used for serial communication up to distances of 50 feet [75]". "RS-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. The serial ports on most computers use a subset of the RS-232C standard[70]". Originally the standard was developed in the late 1960s: "The EIA 232D standard was developed in 1969 to specify the connections between a

computer and a modem. The term itself is an acronym which can be read as follows: Electronics Industry Association (EIA) accepted standard, ID number 232 revision D [72]". The standard specifies baud rates, as well as electrical specifications like line capacitance etc. Among the more important specifications are: "

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.
3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!) [73]"

We can notice that in the case of RS-232, voltage levels are inverted in respect to logic levels; that is, low logic level is represented with a high voltage level and vice versa. So, a voltage diagram of a RS-232 serial packet would look like this:

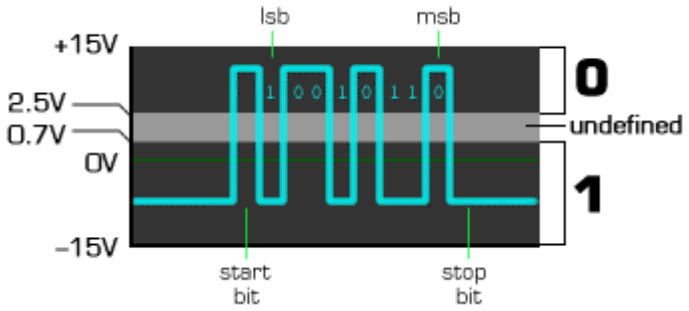
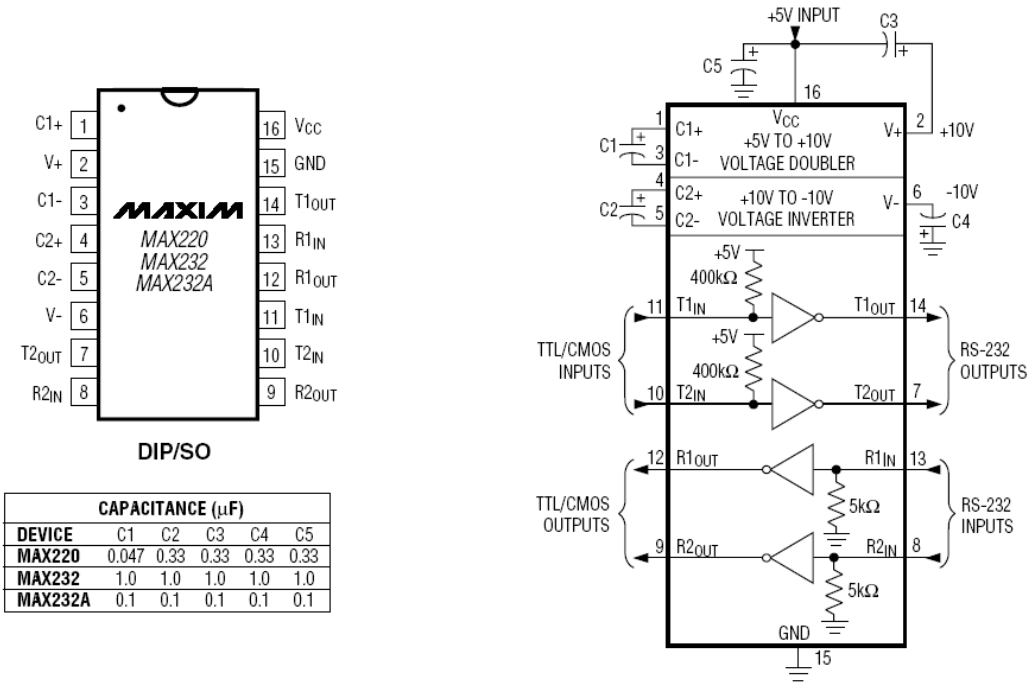


Figure 46. Voltage diagram of an asynchronous RS-232 packet (Ref. [76])

Here we can notice a significant problem: namely, our microcontroller will produce a serial code that conforms to TTL logic; however, the PC needs to receive a signal that conforms to RS-232 logic. This is however a standard hardware communication problem, and it can be addressed by using level converters that can transform signals back and forth between TTL and RS-232 levels. Those are usually manufactured as single ICs. Some models require a symmetric power supply to generate the negative voltages for RS-232, which can be a circuit design problem. However, this issue is well addressed by an IC device, originally produced by Maxim (Dallas Semiconductor), known as MAX232 [77], marketed as "+5V-Powered, Multichannel RS-232 Drivers/Receivers". This device "includes a Charge Pump, which

generates +10V and -10V from a single 5v supply. This I.C. also includes two receivers and two transmitters in the same package [73]". By using a single 5V power supply and some external capacitors, this device can translate two independent signals from TTL to RS-232, and additional two signals from RS-232 to TTL, allowing for conversion of two full duplex communication lines at the same time:



CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 47. MAX-232 pinout (left) and typical basic MAX-232 circuit (right) (Ref. [77])

Once RS-232 levels are available, we can make a connection with the PC; in general, "if you wanted to do a general RS-232 connection, you could take a bunch of long wires and solder them directly to the electronic circuits of the equipment you are using, but this tends to make a big mess and often those solder connections tend to break and other problems can develop. To deal with these issues, and to make it easier to setup or take down equipment, some standard connectors have been developed that is commonly found on most equipment using the RS-232 standards [78]". For instance, the 25 pin connector known as DB-25, which is usually used as a parallel port on a desktop PC, is one of the standard connectors for RS-232.

However, the connector which is most commonly used for asynchronous serial communication, and is known as the serial port, is the 9 pin connector, known as DB-9:

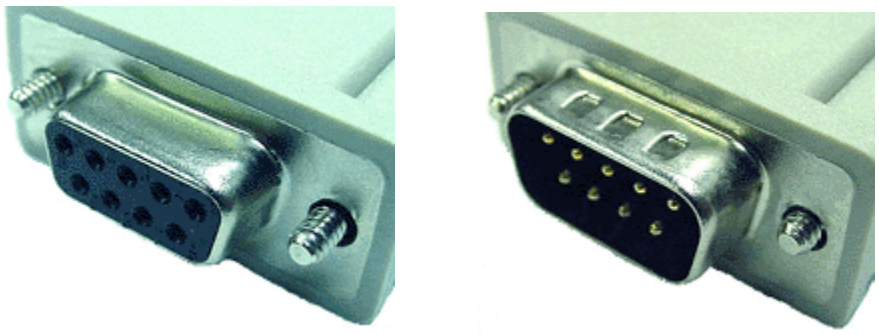


Figure 48. Female (properly known as DE9F) and male (properly known as DE9M) DB-9 connectors (Ref [78])

This connector interface allows that asynchronous, full duplex communication can be implemented by using the minimum of three wires, in the spirit of Figure 36 – and hence a simplex communication is possible by only using one transmission line (plus the implied ground return path). Let us then consider the pinout of this connector:

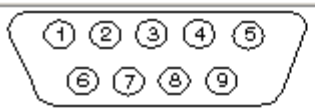
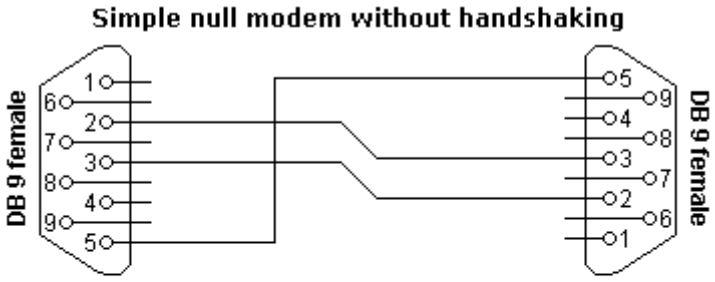
	9 Pin Connector on a DTE device (PC connection)
Male RS232 DB9	
Pin Number	**Direction of signal:**
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from a modem

Figure 49. Pinout of a (male) DB-9 connector (Ref. [70])

For us, the most important pins are SG (pin 5, signal ground), RD (or RX, pin 2, incoming data) and TD (or TX, pin 3). We can connect two serial devices for full duplex asynchronous communication by using only these three pins for connection, which results with only three wires being used:



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

Figure 50. Serial full duplex using only three wires (Simple null modem cable without handshaking, Ref. [79])

From Figure 50 it is obvious that if we want a simplex asynchronous serial communication from transmitter to receiver, all we need is the signal ground wire, and a wire that goes from TX pin of the transmitter to RX pin of the sender – hence we have serial communication through only one transmission line. In other words, in the context of the project, this simply means that whatever we generate as a valid RS-232 formatted serial voltage signal, we need to bring it to RX pin of the receiver (PC serial port) which implies one transmission line, granted that the signal ground is accounted for.

The rest of the pins of a DB-9 connector are used for handshaking (through them the transmitter/receiver can communicate whether they are ready to send/receive data, which helps with synchronization of asynchronous serial), and although they are available, obviously they are not required for establishing serial communication – so they will not be discussed further. Let us also mention that the standard makes a distinction between a DTE (Data Terminal Equipment) and DCE (Data Communications Equipment). DTE would correspond to a desktop PC, and DCE to a device that is remotely controlled by that PC, however note that: "in practice the distinction between the two pieces of equipment is really a matter of function

rather than any real difference [78]". In addition, let's note that serial communication can be seen through the paradigm of the OSI model: "While serial data communication is not strictly a network communication protocol, it is still important to understand the layered communications model when dealing with any sort of communications protocols … Often serial data communication does not implement all of these different layers [of the OSI model], and even more often these different layers are combined in the same module or even the very same function. This model was originally developed by the International Organization for Standards (ISO) in 1984 to help give a good idea of where different networking structures could be separated and intermingled. The point here is to know that you can separate different parts of communications sub-systems to help with the debugging process, and to move structures from one sub-system to another [78]" (see [78] for an alternative layer model aimed for describing serial communication).

Finally, while on the topic of serial communication, let's again mention the issue of the UART. The UART (universal asynchronous receiver/transmitter) is a device that can transfer between parallel data and asynchronous serial and vise versa. As this parallel to serial conversion is performed on a hardware level in this device, using only an oscillator clock, the timing of the produced serial bits should be more precise. However, for microcontrollers that do not have an UART it is still possible to generate asynchronous signal. In essence, by narrowing down the transmission possibilities, we understand that simplex communication can be achieved by basically pulsing a single pin (which will drive a single wire, granted a signal return path) with properly timed high and low voltages. The timing of these signals can be decided using a microcontroller software routine in which case we talk about bit-banging: "*Bit-banging* means generating serial output in software as opposed to with a hardware UART. Timing accuracy of bit-bang output is only as good as the program that generates it, the accuracy of the system clock, and in interrupt-equipped systems, the degree to which other interrupts affect timing. Some compilers generate truly bad bit-bang serial timing, based on the assumption that the receiver is always going to be a PC (with its generous error tolerance) [76]" In this case, opposed to the case of using a hardware UART, the timing of the signals is decided by a software microcontroller routine, and may be affected by more factors than a hardware UART. Yet, in both cases, we are dependent on a crystal oscillator timing for generating the serial bit duration, as even those devices are susceptible to jitter:

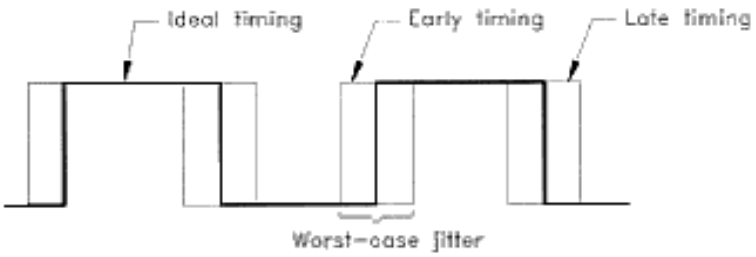


Figure 51. Illustration of jitter (Ref. [80])

Jitter is "the uncertainty or short-term variations of a digital waveform timing from their ideal positions in time. The waveform transition could be too early or too late compared to the ideal waveform timing [80]". If the central microcontroller clock behaves like this, obviously that will be mirrored in the timing of the asynchronous serial signal as well; regardless of whether we use a UART or a software bit-banging routine – both sync to the master crystal clock, only in different ways.

This concludes the discussion on serial communication, as all the necessary aspects that were considered in this project are introduced.

6 System Design and Implementation

The system design is guided by the assumptions provided in the analysis in this report. First, we need to implement a demonstration of a wired DAQ system. We have already discussed how to program the PIC16F74 microcontroller to behave as a mulitchannel DAQ hardware – to sample all of its analog inputs, and create and send asynchronous serial code based on the sampled results. Once that is done, all we need to do is to design a circuit that can execute this microcontroller program – that is, a circuit that will facilitate power and connectivity with a RS-232 PC serial port. In that sense, we would have reconstructed the function of the Teleo in the footstep controller:

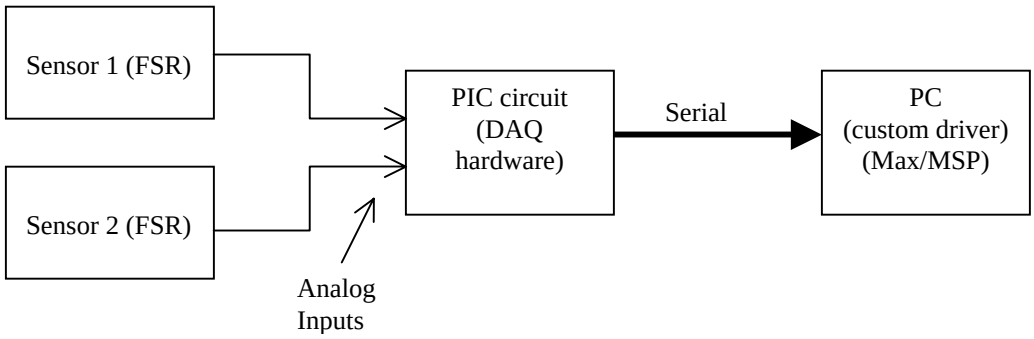


Figure 52. A conceptual block diagram of a custom wired DAQ system in the context of the footstep controller

We would implement simplex asynchronous serial communication, conducted through only one transmission line – we have already discussed the prerequisites for achieving it. A single transmission line implies a single voltage change in time – which can be used to modulate a parameter of a radio wave. We assume that using a pair of commercially available pair of transmitter/receiver, coupled RF devices, can be used to directly apply the serial voltage signal as a modulation signal for a radio wave; additionally, we assume that the demodulation process is automatically performed by the receiver, resulting in the reconstruction of the modulating serial voltage signal in its original form. The received reconstructed serial voltage signal can thereafter be transferred in the same manner to a PC. Simply speaking, we have cut the transmission wire, and closed its ends with coupled transmitter/receiver devices,

which implement a wireless link – thereby we have replaced the wired transmission with a wireless one, as is illustrated on the figure:

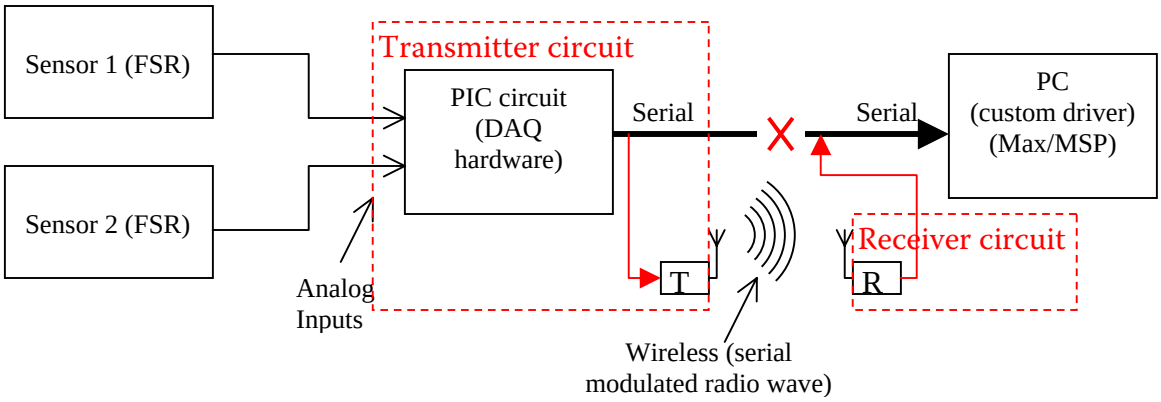


Figure 53. Conceptual block diagram of the replacement of a wired transmission with a wireless one – which implements a custom wireless DAQ system in the context of the footstep controller

The "cutting" process results with a creation of two circuits, a transmitter and a receiver, which implement the wireless link, as is illustrated on Figure 53; eventually this represents the model demanded in the problem formulation and illustrated on Figure 5. As we aim to send the exact same serial stream in both the wired and the wireless case, the corresponding Max driver will be the same for both of them.

6.1 An example of a wired multichannel DAQ system

So, we have written a multichannel DAQ program, we have compiled it, we have burned this compiled program onto the PIC16F74 using the software and hardware programmer, and we have removed the PIC microcontroller from the programmer. All we need to do now is place it in a circuit where it can work. As discussed before, this involves bringing power and timing signal to the microcontroller. A basic circuit that can make the PIC 16F74 microcontroller run is shown on the following schematic:

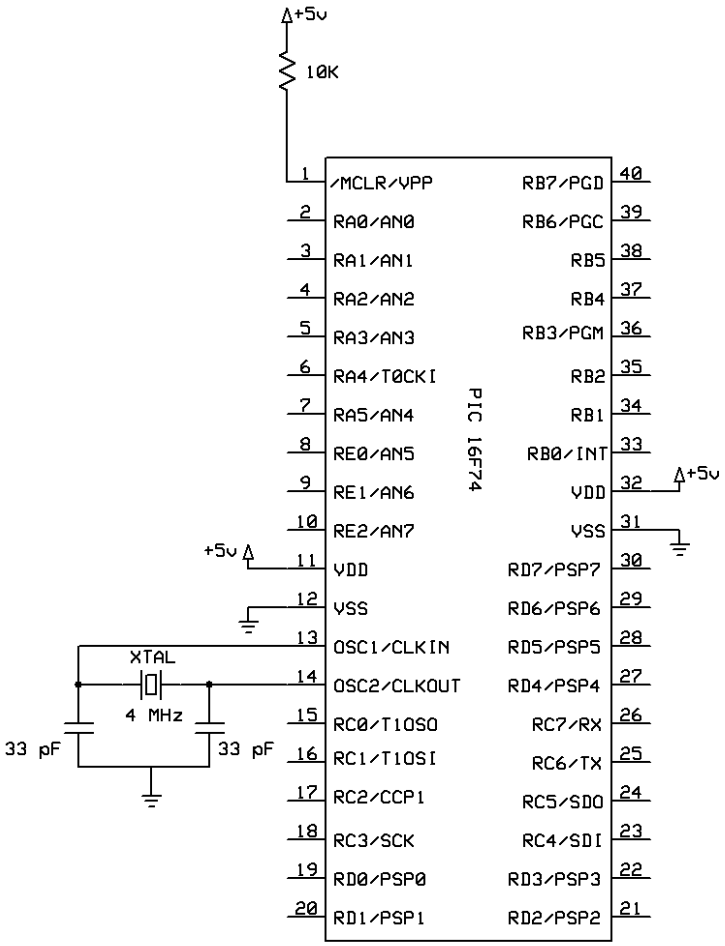


Figure 54. Basic circuit for the 16F74

Let us mention that the basic circuit, for some reason, is not given in this simple form in the datasheet for the PIC 16F74 microcontroller; in case of Microchip such information could be discovered for instance, in some application notes like "Care and Feeding of the PIC16C74 and Its Peripherals" [81]. In the case of this project, the basic circuit was deduced based on different hobbyist projects found on the Internet, like [82].

We can observe that the timing of the microcontroller is implemented by wiring a 4 MHz crystal between OSC1 and OSC2 pins (pin 13 and 14), and 33 pF capacitors from these pins to ground. Let's note that this construction can be also found in a single three terminal device known as resonator (with built in capacitors) – such a device is rendered for instance on Figure 15. In any case, here we use discrete capacitors and a discrete 4 MHz crystal. In addition to that we also need to bring 5V power to VDD (pins 11 and 32) and MCLR (pin 1) pins, and a ground reference to VSS pins (pins 12 and 31). This circuit is all that is needed to make a microcontroller run its program when the circuit is powered. In our case, the PIC would start sampling the analogue inputs (which are not connected to anything, so they are floating – means they might provide random voltage values) and generating a serial code – alternating voltage on pin RB4 (pin 37).

We need to take a look at a power supply section first. As already discussed, we are going to implement the same voltage regulator design as in the hardware programmer case, discussed in 4.4 Hardware programmer circuit.

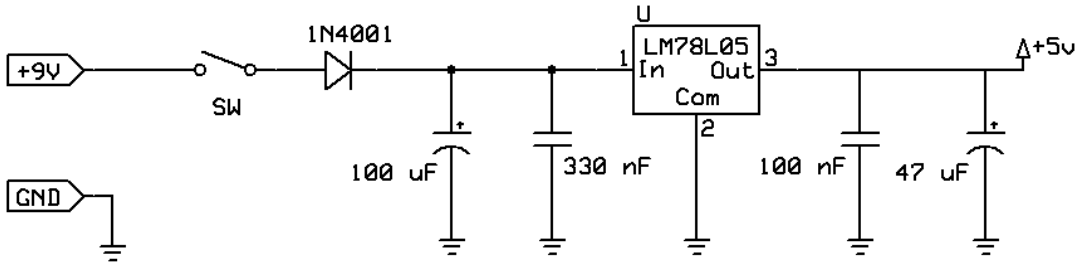


Figure 55. 5V Power supply section

A switch is added before the diode, which can act as an on/off switch for the power section, and thus for the whole circuit. The input power is set to be 9V – in essence, the 7805 can be fed with any voltage in the given range of 6.7V – 35V;

however, by specifying 9V we can implement 9V battery connectors in the hardware circuit. This would allow the circuit to be fed both by a battery, or a standard AC/DC adapter (a lot of these adapters feature several types of connectors, including a 9V battery connector). It can be expected that a general purpose AC/DC adapter will also have some 50 Hz leakage from the AC power supply – however, that is accounted for by the input bypass capacitors (100 µF and 330 nF). Hence, this power supply would cover two common usage situations: a battery and a standard adapter, connected via 9V battery connector.

With the power supply section added to the basic PIC circuit, all we need is a 9V battery and a flip of the on/off switch to make the microcontroller run and generate a serial code on pin RB4. We are aware from the previous discussion that this serial code will conform to the asynchronous serial packet frame; however, we are also aware that the microcontroller generated serial code will conform to TTL logic levels (logic low – 0V, logic high – +5V). We would need RS-232 logic levels to communicate with a PC, hence as discussed, we need to implement a level converter (line driver/receiver) like MAX232. The basic circuit for the MAX232 is the same which is given in the datasheet:

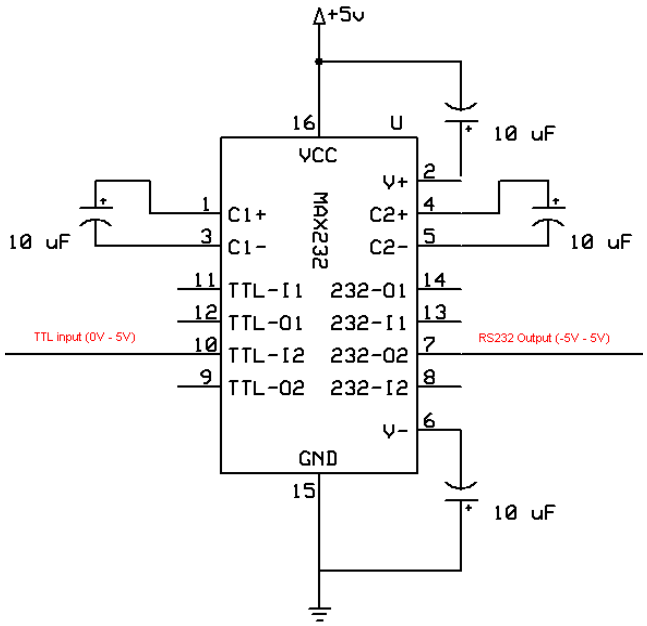


Figure 56. Max232 Line Driver/Receiver basic circuit

We have one TTL formatted serial code generated by the PIC, so it needs to be brought to one of the TTL input pins of the level converter. Here we have decided to use TTL input 2, or pin 10 of the MAX232 IC. The corresponding output to this input is RS-232 output 2, or pin 7 of the MAX232 IC – and this pair of TTL input and RS-232 output defines one "driver" or "RS-232 transmitter" unit of the MAX232. As this is the only transmission conversion we need to perform, the other pins of the MAX232 need not be used.

This RS-232 output 2 (or pin 7) of the MAX232 IC will now provide a proper RS-232 formatted signal (logic low – +5V, logic high – -5V). As such, it can be now directly brought to the receiving pin of the PC serial port (pin RX or pin 2 of the DB-9 connector). Of course, it is understood that the ground reference of the PC serial ports needs to be connected to the ground reference of the rest of the circuit as well. Hence, a complete schematic of the example circuit for a wired DAQ system that can communicate with a PC through the serial port includes the power supply, the PIC with a timing crystal, a MAX232 level converter and a DB-9 connector for connection with a PC; it is presented below:

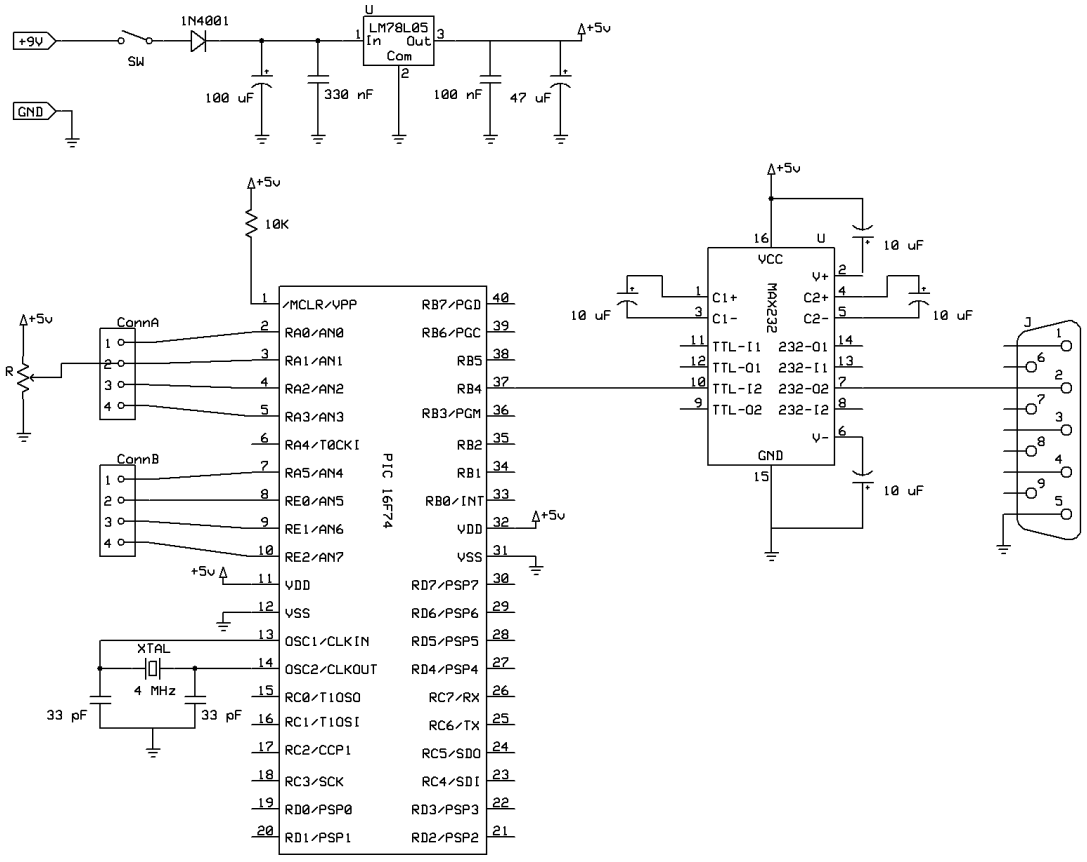


Figure 57. Schematic of the test circuit for a wired data acquisition system

Connectors are provided on the schematic for easier identification of the 8 analog inputs, which correspond to the 8 channels of data acquisition that we perform. An example for wiring a single potentiometer (used say as a rotation sensor) to say channel 2 (analog input 1) is also shown; the potentiometer in that configuration can provide the a voltage swing between 0 and 5V – which is precisely the analog input voltage range of the analog inputs on the PIC. Hence a potentiometer in a configuration like on the schematic could be used to quickly test the operation of the circuit. As the ADC on this PIC allows for 8 bit resolution, we are aware that the input voltage range between 0 and 5V can be represented with 256 levels, where a digital numeric value of 0 would represent a sampled input voltage of 0V, and a digital numeric value of 255 would represent a sampled input voltage of 5V. Hence the results of the A/D conversion take up one character or one byte per channel. Note also that the schematic allows a visual perception of a linear "wire" connection that carries the serial signal.

The PCB layout for the hardware implementation of the given schematic for a test circuit (for a wired data acquisition system) is presented below:

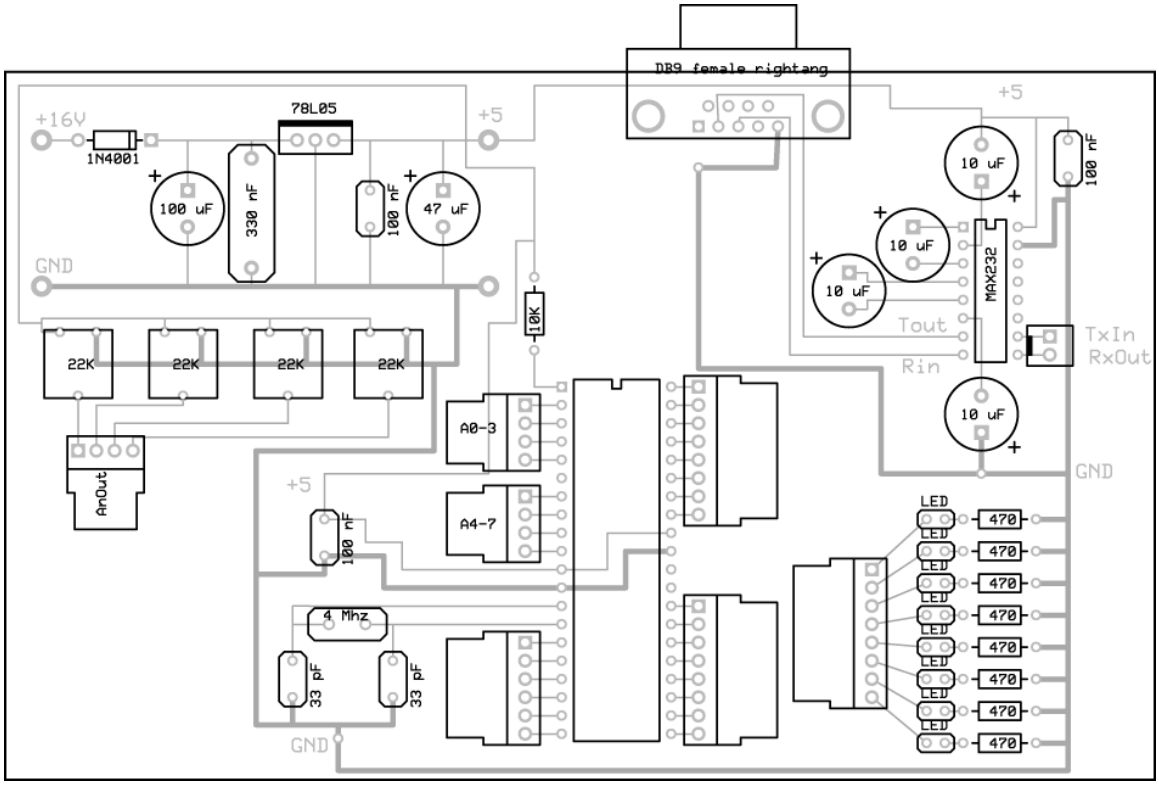


Figure 58. PCB Layout of the test circuit for a wired data acquisition system

The approach taken in the design of the circuit board, is that the power connection are implemented as PCB copper tracks. The I/O pins of the microcontroller are connected with Molex connectors. An additional section of four 22 KΩ trimmers is added, to act as analog sensor signal generators, and are also connected with a Molex connector. Molex connectors are also implemented at the input pins of the MAX232 IC, so in a sense, by using these Molex connectors, we have isolated each functionality module. Using patcher wires, which would have Molex connectors on the end, connections could then be made – the corresponding pins of each module can be connected directly, through the connectors and the patcher wire. A separate module, also provided with Molex connectors, was implemented using LED diodes and resistors – to allow for more versatile PIC program testing. The ground line is implemented as a single copper PCB trace, and all of the ICs have a reference to it – hence there is a return path for all transmission lines implemented. In addition, the ground is shared with the serial port of the PC, since it is brought to pin 5 of the DB-9 connector.

So, to test the circuit, we simply use patcher wire to connect an analog voltage source (generated by the trimmers) to an analog input of the PIC, and also, to connect the serial output pin (RB4) of the PIC to the TTL input 2 pin (pin 10) of the MAX232. We use a corresponding serial cable to establish a connection between the DB-9 connector on the circuit and the DB-9 serial port of a PC. With this setup, by turning on the circuit, the PIC starts generating serial code, of settings 4800 8N1 which is transferred to the PC. This serial code can be received in the PC using a generic application like the Windows default Hyperterminal, which then responds with the output like the one rendered on Figure 33. Of course, using the custom driver for Max/MSP is also possible.

The circuit operated in this manner without any serious problems already at first power up. As such, it was determined that this circuit fulfills the purpose of demonstrating a single microcontroller, wired data acquisition system, and hence all the prerequisites for pursuing a wireless version of this design were in place – we can thus proceed with development of the wireless data acquisition system.

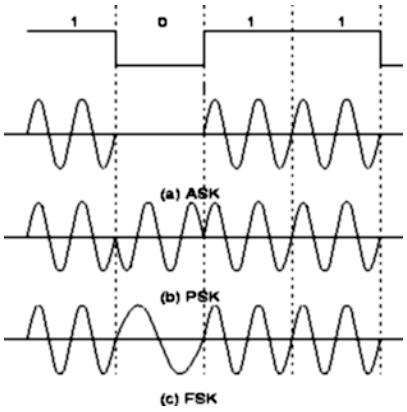
6.2 An example of wireless multichannel DAQ system

As demonstrated on Figure 53, what we need to do now is simply "cut the wire" and terminate the cut ends with coupled radio transmitter/receiver devices, which would implement a wireless link. Before we do that, let's take a look at the properties of such devices that can be found on the market, and some basic principles behind their operation.

6.2.1 RF devices - introduction and issues

We have insisted on providing a single simplex communication channel, since in that case we have a single changing voltage that represents a communication signal (in this case an asynchronous serial signal). In that case, it is easy to identify the possibility to apply this voltage as a modulation signal to some parameter of a radio frequency signal. Using an antenna, the modulated RF signal can be radiated into space as a radio wave, hence allowing for wireless communication. On the receiving side, a demodulation procedure must be applied to the received RF signal, and the original modulation signal (in this case an asynchronous serial signal) can be reconstructed.

Thus we enter the complex theoretical subject of modulation, and in particular, digital modulation (encoding) schemes. Without going too deep into the subject (consider [83] as introduction), we can say that the three basic parameters of an alternating signal (wave) that can be modulated are amplitude, frequency and phase. When talking about digital modulation, we use the term "shift keying", so we have amplitude, frequency and phase shift keying (ASK, FSK and PSK).



As this figure suggests, modulation implies that an alternating periodic signal to be modulated exists; in this case it is a simple sinusoid. When we have a raw binary signal, transmitted by changing constant voltage between a high and low level (as in the implemented serial communication) we use the term (NRZ) pulse code modulation – PCM [85].

Figure 59. ASK, FSK, and PSK modulation scheme (Ref [84])

It is obvious from Figure 59 that a single bit stream, like the synchronous serial stream that we are generating with the PIC, can be applied as a modulation signal to a single frequency unmodulated signal. On the electronic hardware level, this simple mapping means that we are faced with construction of a high frequency oscillator, which would produce the unmodulated radio frequency AC signal; construction of a modulator, which would accept the PIC generated serial as a modulation signal and apply it to the raw AC signal; corresponding high frequency amplifiers; and construction of an antenna system that is an efficient radiator for the given frequency. This involves quite complicated issues, since size, frequency and efficiency of the antenna are related; the smaller size antenna we want, the higher the frequency needed. However, as we go into higher frequencies, parasitic characteristics of the components begin to appear – hence the design of a radio device at high frequency from discrete components is a very difficult manner; as is the actual implementation as well – at high frequencies, the very length and shape of the PCB tracks begins to have influence on the eventual circuit operation. In the context of this project, attempting to build such a custom transmitter/receiver system, with the modulation and demodulation capabilities was practically impossible.

On the other hand, in recent times, miniature integrated circuits that implement such functionality on a single chips are readily available. In spite of the great number of producers, big and small, of such chips, compact introductory information into this subject is difficult to track down; most of the time, only articles or tutorials from the manufacturers themselves is available (consider for instance, the 'RF Topics' online collection of articles from Aerocomm company [86]). The

terminology used is different, from 'RF modules' to 'RF chipsets', and the functionality can differ. A lot of times, the engineering considerations described in such articles deals with advantages and disadvantages in relation to different wireless technologies, such as Bluetooth, or wireless LAN; there are IC radio chips that address those technologies too. However, we need something far simpler – a pair of devices that implement both the simplest theoretical notion of modulation of a RF signal with an external voltage input in the transmitter, and the notion of reception and demodulation of the signal in the receiver – in this way, most of the difficult engineering issues involved in RF design can be safely avoided, while still keeping the perspective on signal transmission. Hence, instead of providing an introduction into these devices from a theoretical perspective, we shall simply describe the devices that were used in this project, and the properties that made them desirable for use.

6.2.1.1 RF Solutions RTFQ1 and RRFQ1 – transmitter and receiver modules

To begin with, the investigation into these devices started off with a wrong assumption that the IHK component store did not have such devices in stock. It comes out at that it does: the devices in question are TX3A and RX3A (datasheet [87]) by the British company Radiometrix. These ICs provide the exact same functionality as the devices used in this project, with slightly better performance, according to the datasheet, and are marketed as "UHF FM Data Transmitter and Receiver Modules".

The devices chosen for this project are known as RTFQ1 (transmitter) and RRFQ1 (receiver), produced by the British company RF Solutions (datasheet [88]). These devices are marketed as "FM transmitter & receiver hybrid modules" and have the following features:

Transmitter

- 3-12V supply voltage
- SIL or DIL package

Receiver

- PLL XTAL design
- CMOS/TTL output
- RSSI Output
- Standby mode (max 100nA)
- 5V supply voltage

- Range up to 250 metres
- FM radio transmitter and receiver
- Data Rate upto 9.6Kbps
- Miniature packages
- No adjustable components
- Very stable operating frequency
- Available as 315, 433 or 868 MHz
- Temp range -20 to +85C

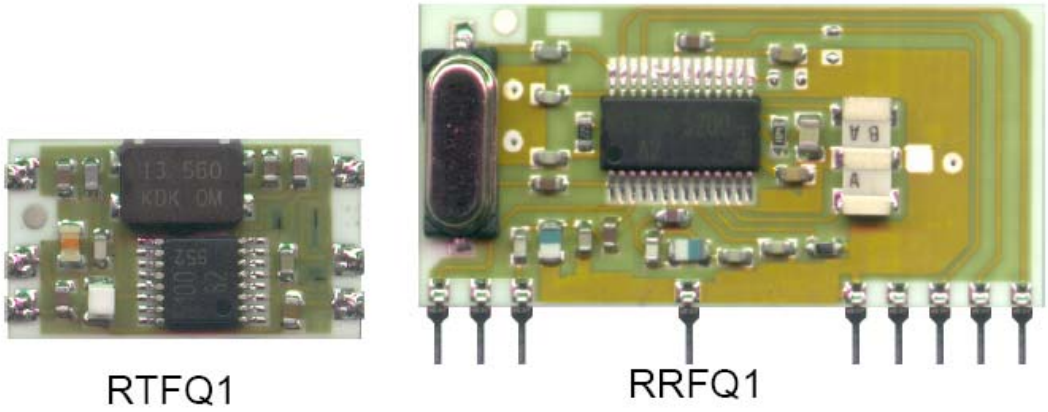
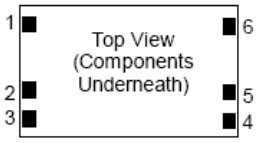


Figure 60. Transmiter (RTFQ1) and receiver (RRFQ1) RF IC modules used in this project (Ref. [88])

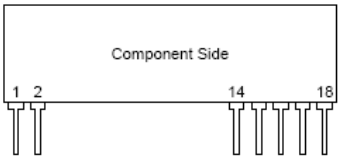
As we can see from the images, these are complex devices, which nonetheless provide their functionality through a simple interface of only few pins each.

Images on Figure 60 do not do justice to the real size of these devices: the transmitter is about 20 x 11 mm, and the receiver about 38 x 18 mm in size (excluding pins). We realize that they are adjusted to TTL logic levels, and to transmission of binary signals – as there is a guaranteed data rate of 9.6 Kbps. The datasheet confirms they suitable for use in the project: "This transmitter and receiver pair enables the simple implementation of a data link at distances upto 75 metres in-building and 250 metres open ground ... Because of their small size and low power requirements, both modules are ideal for use in portable, battery powered applications such as hand-held terminals[88]" As these devices seemed to suit the needs of the project development, and were available for ordering, a transmitter and receiver pair of these devices was acquired. These devices are coupled – they come in pairs that operate at a single frequency. There are three frequencies of operation made available by the manufacturer; in this case only the 868 MHz transmitter/receiver pair was available, so that is the one used in this project.

We can look at the pinout of these devices to find out more about the functionality they provide:



RTFQ1	Name	Description
1	En	Enable (active high)
2	IN	Data input
3	GND	Ground, Connect to RF earth return path
4	Vcc	Supply Voltage
5	GND	Ground, Connect to RF earth return path
6	EA	External Antenna



RRFQ1	Pin Description
1	+Vcc
2, 7,11	GND
3	Data In (Antenna)
12	NC
13	Received Signal Strength Output
N/A	AF Output
14	Data Out
15	Power Down 0V = Standby 5V = Operating

Figure 61. Pinout of the transmitter (RTFQ1 - top) and receiver (RRFQ1 - bottom) module (From ref. [88])

From the pinout description, and the typical application schematic provided in the datasheet, we can deduce the basic principle of operation of these devices: when the transmitter is powered (pin 4) and enabled (pin 2), a high frequency signal is generated, centered at 868 MHz, which is then frequency modulated by the voltage brought to the input pin (pin 2). By attaching an external antenna (to pin 6), this RF signal is radiated into space as a radio wave.

The receiver, when it is powered (pin 1) and enabled (pin 15 high - Operating), listens for a radio wave centered at 868 MHz through an external antenna (attached to pin 3 – Data In Antenna). The power of the detected radio signal at that frequency is reported as DC voltage (in datasheet, given range is from 1.2V to 2.75V). At the same time, the received RF signal is demodulated back into the original modulating voltage, and provided on the Data Out pin (pin 14).

This electrical interface looks simple enough, and it seems to successfully cover all the issues that are otherwise involved with the task of modulating and demodulating a radio wave (a quick look at the block diagrams for the devices in the datasheet reveals oscillatory circuits, PLL synthesizers, power amplifiers, filters and demodulators – each of which is a challenging engineering undertaking, had we

attempted to implement such a system from scratch with discrete components). This, however, does not mean that we do not have any RF issues at all, simply by choosing to use these devices. In any case, we must remember that a high frequency signal sees the elements of the circuit differently, due to which the geometry of the circuit starts playing a role. This is, in one way or another, the background behind all issues that are radio/HF specific in regard to hardware layout; the problems encountered in the development process can also be traced back to this origin. The experience from the development shows that these issues, if not handled properly, eventually effectuate themselves as errors in the decoded stream.

Thus, it is good to take a look for basic recommendations when having to deal with such problems. The datasheet includes the following prototyping hints: "

1. It is essential when building any Low Power Radio System that you have a 'clean' DC power source. Typically the ripple voltage should be less than 10mV Peak to Peak. Normally a 470uF decoupling capacitor is sufficient de-coupling for an AC derived DC power source.
2. Never place a Transmitter or Receiver directly into Vero-Board or any similar prototyping board. This will severely restrict the range. Rather, use small lengths of wire from the prototyping board to the pins of the Transmitter or Receiver.
3. A useful antenna, for testing purposes, for both the Transmitter and Receiver on 433MHz is to use a piece of wire 17.3cm long (23.8cm at 315MHz) soldered directly to the antenna pin. [88]"

The third tip was initially understood as a promise that the, otherwise complicated, problem of antennas could be easily avoided. Let's note that these lengths represent a quarter wavelength for a wave at the respective frequency, using the wavelength formula:

$$\lambda = \frac{c}{f} \qquad \textit{Eq 5}$$

where λ is the wavelength, c is speed of light, and f is the frequency of the wave. Using this formula, a quarter wavelength for 868 MHz is about 8,6 cm – eventually a piece of wire of about 9 cm was used, so the advice was followed literally. This however is not

enough as we shall see further on – and this was one major issue in this part of the development.

The second advice stems again from the effects of the geometry of the PCB tracks at high frequencies, and vero-boards typically feature grids of conductors, which can exhibit capacitive properties at high frequencies. The first advice involves the decoupling of any ripple that may be left from the 50 Hz mains AC – which is performed in the design of the voltage regulator power supply that we have used in circuits so far.

Other recommendation can be found on the RF Solutions online forums, which are quite useful place to look for information. Here among others we find tips like: "

- Tracks connected to the antenna (RF input / output) pin of transmitter and receiver modules should be as short as possible. Any conductor connected to this track will act as an antenna, so it will lengthen and detune the actual antenna.
- Design PCBs with an adequate ground plane. A ground plane is an area of conductive PCB connected to the circuits ground. It helps with RF propogation and is generally better when placed perpendicular to the antenna. It should be placed on the pcb in a large area, however it should not be placed directly underneath the RF module.
- Use decoupling capacitors on the power supply circuitry to prevent rf interference passing through the power lines. [92]"

Here we actually find advice that corresponds to the two major issues encountered in this project development: the issue of bypass/decoupling capacitors (stable power source) and the issue of antenna construction (ground plane). Before these issues were addressed, the reception resulted with a completely erroneous signal; as they got resolved, the limitations in the operation of the devices emerged. Thus, before we discuss these two hardware issues, let us look at the signal reception and reconstruction perspective of the wireless serial link implemented by RTFQ1 and RRFQ1. Some of the issues in this discussion resulted with specific interventions in the discussed microcontroller program.

6.2.1.2 Mode of operation of the wireless link

Let's note that the datasheets for these devices actually do not have a lot of information about expected operation of the wireless link established by the devices. For example, the datasheet does not specify what sort of values are expected on bit level during normal operation of the wireless link, that is, if a low voltage is applied as modulating input at the transmitter, what is the corresponding decoded value in the receiver? The typical application circuit in the datasheet uses other digital devices from RF Solutions as encoders and decoders of switch values; however, the actual signals that are generated to drive the transmitter are not reported.

During the development it was concluded that, within the operation quality achieved, the link implemented by the devices behaved like this: when a high voltage was sent (used as modulation input at the transmitter), high voltage level was reported as received data; however, when low voltage was sent, the receiver would report a low voltage only for a certain amount of time, after which random sequences of high and low voltage levels occurs.

Thanks to the availability of an oscilloscope with snapshot memory and printing capabilities, we can observe that effect on the following figure:

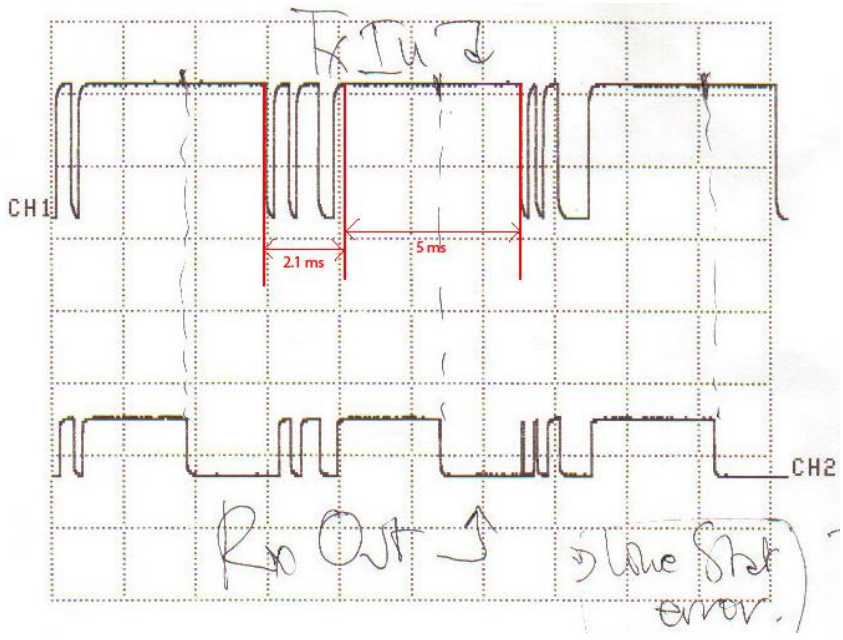


Figure 62. Simultaneous oscilloscope trace of the sent input signal (top) and received output signal (bottom)

Initially, the very first pair of transmitter and receiver circuits produced, resulted with a completely random digital signal being decoded for a low voltage level sent from the transmitter; the high level was reported properly. This of course resulted with complete lack of synchronization, and blockage of most PC programs that read the serial port. Several measures were taken to troubleshoot this problem. Communication was attempted at the beginning at the maximum rate offered by the devices – 9.6 Kbps. This was then dropped to 4.8 Kbps (as it stayed in the final prototype version) by software. As it didn't help, an unusually long delay of 5 ms length was coded between each serial character sent. Consider that a bit duration at 4.8 Kbps is 208 μs, meaning that a 8N1 packet that is 10 bits long takes about 2.1 ms to be transferred; thus the delay between each packet (character) is almost twice the duration of the packet. Also, an alternating sequence, of high and low levels, 1 second in duration each, was implemented in the program before actual transfer of serial data starts. This was intended to help with troubleshooting the wireless, and it did help – one could observe one second of stable received level when a high level was sent, and one second of random noise when a low level was sent.

After solving the issues with the decoupling capacitors and the antenna ground plane, discussed further on, the operation of the wireless link changed and the trace displayed on Figure 62 was obtained. It was deduced that meaning of this trace is that for a high modulating voltage on the transmitter (a logic 1 being sent) the correct, high value could be decoded and generated by the receiver only for the first 2.5 ms after that state had been entered (after a low to high transition has been made) – thereafter the receiver flips its state and stays on low voltage. The slow alternating sequence at the start of communication also confirmed this – a stable low level is observed during a one second input low level – when the input is set to a high level, the received output transits from a low to a high level, and then it quickly transits back to a low level. This already represents a reliable operation of the wireless link, since all we need to do is change the PIC program and remove the unusually long delay – as long as each state is not longer than 2.5 ms (and let's again note that at 4800 bps one character packet takes 2.1 ms, so in one packet we must have at least one start and one stop bit – that's at least two transitions, even if the data bits are all one or all zero) it will be properly decoded by the receiving side. So, the delay was removed and the wireless link started working properly, with characters being decoded by the PC.

This behavior is most likely a design feature of these particular circuits – it seems that the receiver relies on transitions between low and high levels to establish a sync with the incoming radio wave, and it seems that such synchronization cannot be maintained for a longer period of time, when a high voltage level is being sent. As such information is not covered in the datasheets, here the RF Solutions online forums are again quite useful place to look for information. Within this forum, we can find that "it is not possible to simply connect a logic high and expect it to be repeated at the other end etc. instead you must use encoding to filter out noise and gain a reliable data communications path [90]" which implies that this is the observed mode of operation of the wireless link: it guarantees only that a state transition will be correctly reported, but not that the state itself will be correctly outputted for its entire duration as long as it may be.

Here is actually one potential problem with the asynchronous serial code as we are using it – namely occasionally long sequences of 0s or 1s can occur – as this is non return to zero PCM modulation, a sequence of 1's looks like a long constant high voltage (which is visible on Figure 62 as well). Thus devices that rely on bit transitions for synchronization would have a bit of trouble with the asynchronous signal as we use it; however, let's not forget that the framing with start and stop bits at least guarantees transitions at the start and end of a packet; that is why we can actually use the wireless link with raw serial signal as we do. We can find confirmation for this elsewhere on the RF solution forums: "The basics of getting a good reliable data communication channel using radio modules is to ensure that you have a good preamble 16 bits or so, data formatting to ensure that you do not get a consecutive sequence of 1's or 0's and checksum on the end to verify the data is right and what you had expected. The Manchester encoding system is one such system that does all of the above and is very good [91]". If we notice that Manchester encoding is such that say a logic 0 is encoding as a low to high transition, and a logic 1 is encoded as high to low transition, we can safely assume that the conclusion that the radio link implemented by these devices actually demodulates bit transitions, and guarantees a (high) level only for a certain amount of time after a transition, can be seen as correct; and that it is the proper operation of the device. This was taken into account by implementing the 8 bytes of the number 85 (which is represented as alternated 0's and 1's - 1010101), which is the ASCII code for the character 'U', to be sent at the beginning of each word packet from the microcontroller.

Again, the initial transmission issues of random levels being decoded was remedied by implementation of bypass capacitors and an antenna ground plane. With this, the wireless link could be trusted to correctly report the level of the sent signal within 2.5 ms (after which an eventual high level would collapse to a low one). Yet, even though this mode of operation allows that asynchronous serial code can be directly used as modulation signal and transferred, the receiver still generates a random binary signal, if the transmitter happens to be turned off at the time – and this again causes serial framing errors and locks serial PC applications. Hence, at the current state of the prototype, care must be taken that transmitter is always turned on before the receiver – to prevent generation of random serial data, which would cause framing errors in PC serial applications.

We can now proceed with a brief discussion of the two major RF design issues in this project. These were also noted on the RF Solutions forum [93]in more detail.

6.2.1.3 Decoupling capacitors on the power supply

Let us now briefly discuss the issue of decoupling capacitors. As the typical application circuit in the datasheet noted a 3.3V power supply being used for the transmitter, a corresponding voltage regulator was obtained, with the intent to produce that voltage (as the max voltage that can be fed to a RTFQ1 is 4V). At the first testing attempt, it was hooked to feed from the 5V power supply directly, and its 3.3V output was directly fed to the supply voltage pin (VCC, pin 4) of the RTFQ1. At that point, the wireless link still reported a completely erroneous signal.

Following the advice found on the RF solutions forum about RF interference and power line, the output pin of the 3.3V regulator was inspected with an oscilloscope, and surprisingly enough, a radio frequency ripple was observed, superimposed to the constant 3.3V. As the advice was to use decoupling capacitors (which was also discussed in the case of the power supply for the hardware programmer), several values were tried out, by soldering capacitors directly on the PCB tracks, between the output pin and the ground pin of the 3.3V voltage regulator.

The values that cleared out the ripple are noted in the schematic for the transmitter circuit.

6.2.1.4 Antenna ground plane

As mentioned before, the advice to use a wire of certain length as an antenna was followed literally. In this case a bit too literally, as in the very first version, the wire was the only part implemented as an antenna system. However, that is not enough – a ground plane is necessary as well, as also the forum advices given previously recommend.

Let's start by looking at a simple radio communication system:

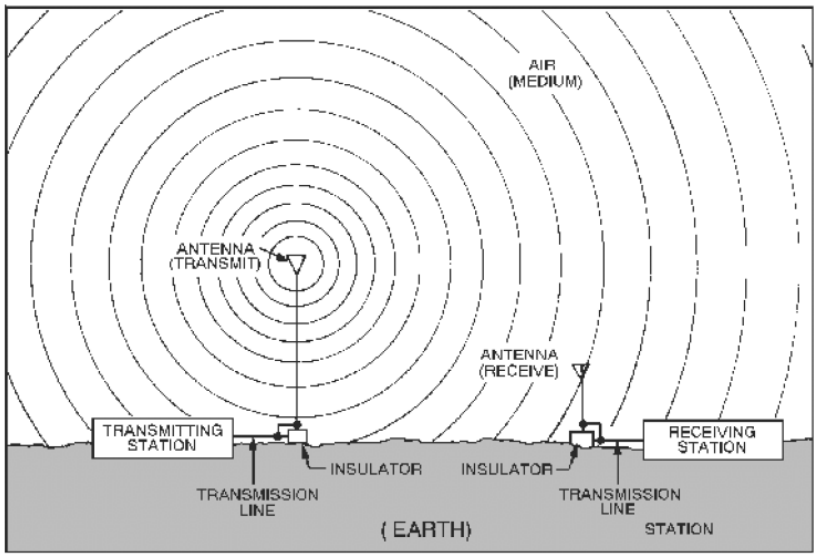


Figure 63. Simple radio communication system (Ref.[94])

The operation of this system can be described as follows: "In the illustration, the transmitter is an electronic device that generates radio-frequency energy. The energy travels through a transmission line to an antenna. The antenna converts the energy into radio waves that radiate into space from the antenna at the speed of light. The radio waves travel through the atmosphere or space until they are either reflected by an object or absorbed. If another antenna is placed in the path of the radio waves, it absorbs part of the waves and converts them to energy. This energy travels through

another transmission line and is fed to a receiver … An antenna is a conductor or a set of conductors used either to radiate electromagnetic energy into space or to collect this energy from space [94]" However, one thing that we must keep in mind is that "The half-wave dipole antenna is the fundamental element normally used as a starting point of reference in any discussion concerning the radiation of electromagnetic energy into space ... The antenna acts as if it were a capacitor [94]". It is this principle of the antenna operation – that it acts as a capacitor, that demands implementation of a ground plane.

Let us then briefly repeat the principle of operation in a capacitor.

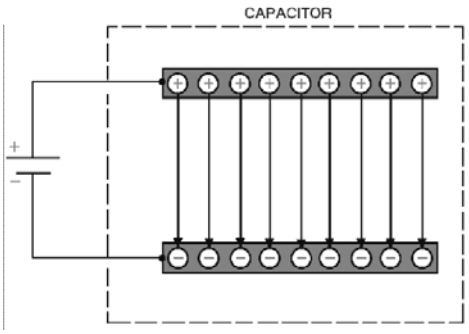


Figure 64. Electric field between metal plates of a capacitor (Ref.[94])

A capacitor is a system of two metal plates separated by an insulator. We can assume that at the beginning of the analysis, both plates are electrically neutral, thus at the same electric potential. When the generator starts acting, its electromotive force starts taking electrons away from one plate and pushing them onto the other. As net transport of charge has occurred, we observe that current is flowing through the conductors. However, as the pushed charge has nowhere to go, it accumulates on the plate, resulting with one plate having excess charge – similarly the other plate has lack of electrons. This means that both plates are now at different electric potential, which means that force acts on the plates, and which also means that an electrostatic field is being established in the space between the plates. As the electric field lines are always normal to a conductor, most of the electric field is contained in the space between the plates.

Let us now try to stretch out the plates of the capacitor. Even though the plates are at different angles, the electric field lines are still required to be normal to the conductors.

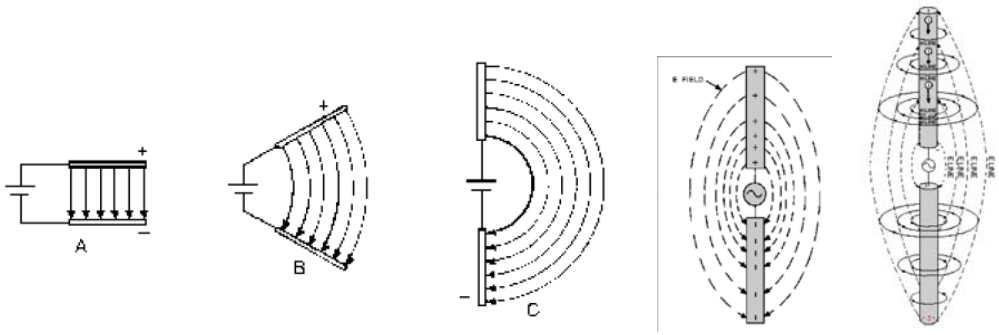


Figure 65. A, B, C: Electric fields between plates at different angle; electric field lines between antenna elements; relationship between magnetic and electric fields and current flow (Ref.[94])

When the capacitor plates are stretched at an angle of 180 degrees we can notice that most of the field lines occupy free space, due to the requirement that they are normal to the conductor surface. This extreme case is actually the definition of a dipole antenna – the capacitor plates are replaced with wire antenna elements.

We can recognize here that the process of charging of the capacitor plates, involves current flow through the generator – which also involves creation of a changing magnetic field that is coupled to the changing electrostatic one. Hence an alternating process of charging and discharging the plates will result with a continuous alternating flow of current, which will result with continuously alternating electromagnetic fields. Such fields propagate into free space as an electromagnetic wave. We can visualize the evolution of the fields during one period of alternation in the following diagram:

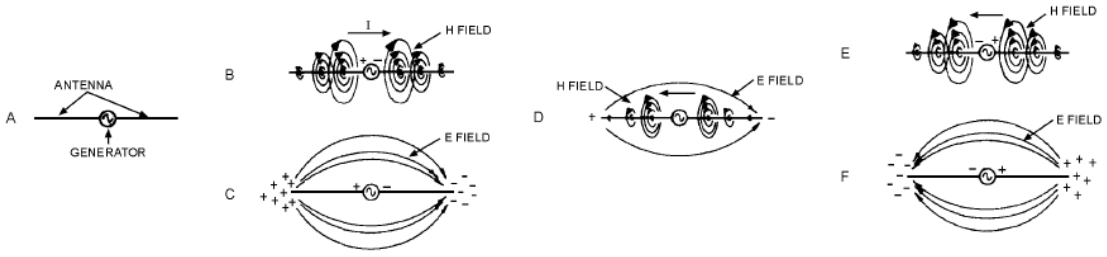


Figure 66. Induction field about an antenna (Ref.[94])

Obviously, this is not a trivial process. Yet, without going too deep into the theoretical details, we understand that having a continuously changing voltage if we count on propagation of electromagnetic waves – hence the need for generation of an unmodulated RF carrier wave, and modulating it afterwards with the digital signal. The geometry of the dipole antenna as a stretched capacitor is also of great importance here, since we can simply understand that most of the fields are in free space anyway (they are not contained within as in the capacitor case) – hence an antenna can radiate electromagnetic waves into free space efficiently. Thus, it is also important that there is a free space shortest path between the two conductors – that the electric field lines would follow, to close at right angle on the surface of the conductors.

And here is where the problem with the ground plane is found. Namely, if we simply place a piece of wire in the circuit as an antenna pin, the rest of the circuit tries to continuously push or pull charge from it – so implicitly, it becomes one of the element conductors of a dipole antenna. The other element is implicitly the ground conductor, which has the geometry of a PCB track. Obviously, lines that would close at right angle from a wire antenna to a PCB track at right angle, would not nearly as efficiently fill out free space as is the case in the basic dipole antenna. Hence, a separate conductor is needed specifically for that purpose – to establish the free space shortest path reference to the antenna conductor – and this is the ground plane.

As noted before, in the very first version, the ground plane was missing. After considering the above discussion, a ground plane was implemented on the same transmitter and receiver circuits, as luckily, there was some available space around the antenna on the PCB which was not occupied by components. Pieces of an unprinted PCB board were then cut and drilled at the center, so the antenna can go through; and placed on the PCB board, with the antenna coming through the drilled hole, with the copper side facing towards the antenna. A simple piece of wire was soldered on the ground plane, and was used to establish the connection to the ground conductor of the respective circuit.

That was implemented for both transmitter and receiver circuits, resulting with a solution looking like this:

Figure 67. Additional ground plane added to the original transmitter/receiver circuits, made of a drilled piece of single sided PCB board

With this implementation in place, and the mentioned decoupling capacitors, the wireless link implemented by the transmitter and receiver modules begun to reliably operate in the previously described mode of operation. These adaptations are included in the transmitter and receiver circuits presented here.

6.2.2 Transmitter circuit

The transmitter circuit is a direct implementation of the cutting idea displayed in Figure 53, applied to the schematic of the example circuit for the wired DAQ system. The serial transmission line from pin RB4 of the PIC to the MAX232 level converter is “cut”, and brought to the Data In pin of RTFQ1 transmitter module. The power supply generates stable 5V and 3.3V voltages, used to power the PIC and the RTFQ1 respectively. The schematic of the transmitter circuit is presented below:

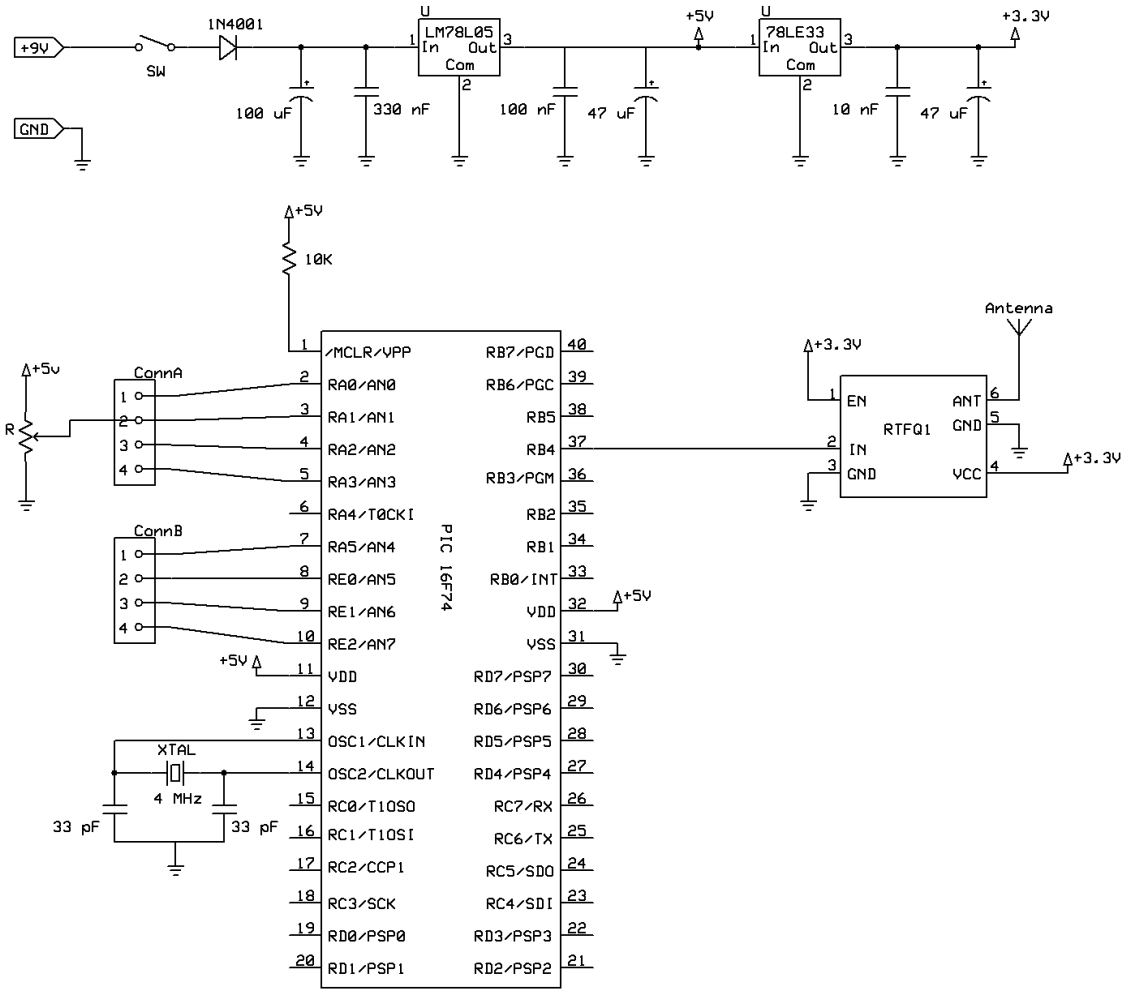


Figure 68. Schematic of the transmitter circuit of the wireless DAQ system

The final PCB layout, presented on the right, is 88 x 94 mm in size. It features wire connectors with mounting screws for the analog inputs, 5V power and ground – in the same fashion as the Teleo, thereby facilitating easy and solderless connection of wire and ready made sensors. Area is allocated on the layout for a ground plane.

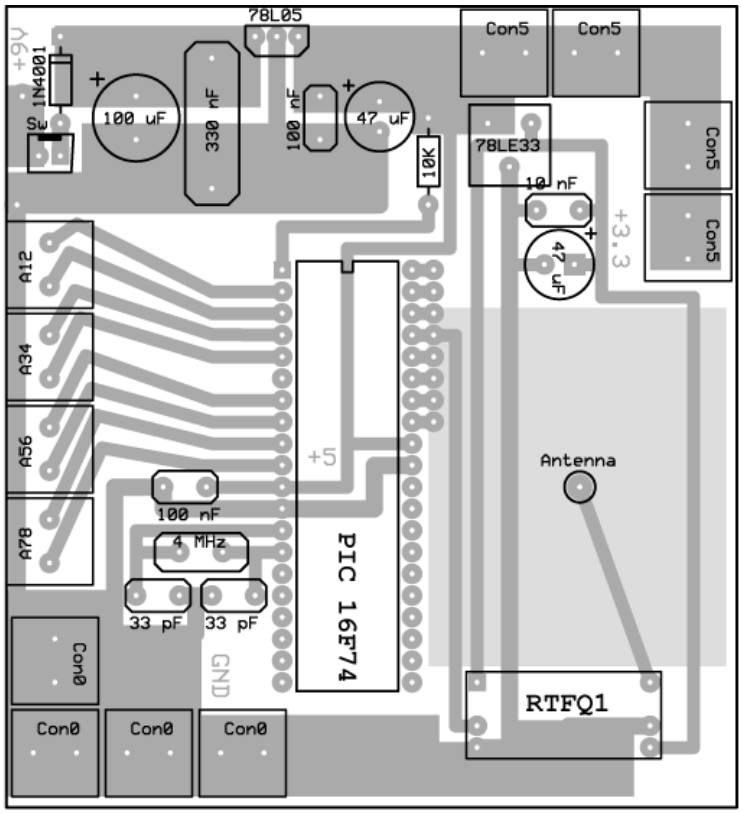
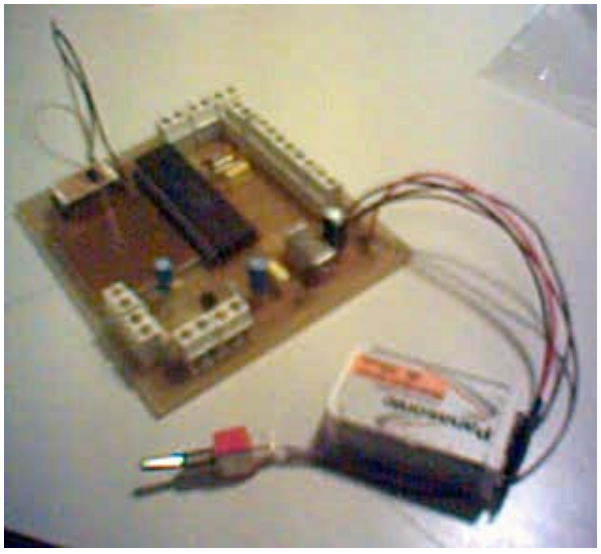


Figure 69. PCB Layout of the transmitter circuit of the wireless DAQ system



The image on the left shows the actual form of the transmitter circuit, with the PIC, antenna, ground plane, and connectors, as well as a switch and a battery connection. It is portable and can be worn in a pocket. The antenna can be skewed in that case, as long as it does not touch the ground plane.

Figure 70. Final form of the transmitter circuit of the wireless DAQ system

6.2.3 Receiver circuit

The receiver circuit is the other part of the direct implementation of the cutting idea displayed in Figure 53, applied to the schematic of the example circuit for the wired DAQ system. The serial line from the PIC to the MAX232 that was cut, is here taken from the Data Out pin of the receiver module RRFQ1. As it conforms to TTL voltage levels, and we saw that indeed, it can represent the sent asynchronous code bit by bit, it is directly hooked to the chosen TTL input of the MAX232. The remaining connection toward the DB-9 serial connector remains the same as in the case of the wired example circuit.

There is an additional yellow diode added to the power section, to visually indicate power up, as well as an NPN transistor with a LED diode, which is fed by the measured received signal strength pin of the RRFQ1 – to indicate visually whether a working transmitter is registered. The schematic is presented below:

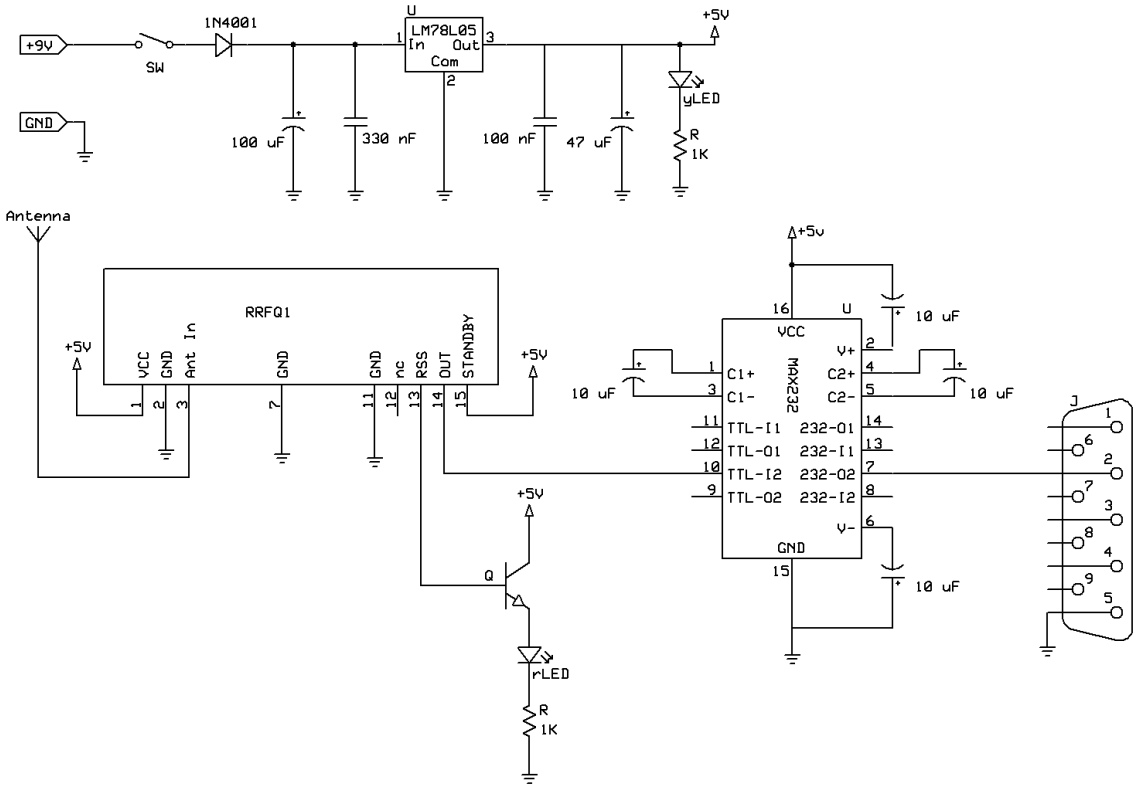


Figure 71. Schematic of the receiver circuit of the wireless DAQ system

The final PCB layout, presented on the right, is 94 x 64 mm in size. It features several jumpers that were placed to facilitate easier troubleshooting if needed. Also, since the transistor is used in that manner, so it turns on when the received signal level is high (that is a transmitter has been turned on), it was thought that the same very transistor could be used to enable the chip as well, thus turning it on only when the transmitter is online, and avoiding the problem of generating random binary data when the transmitter is offline. That is however impossible, since the chip needs to be enabled for the signal measuring function to work.

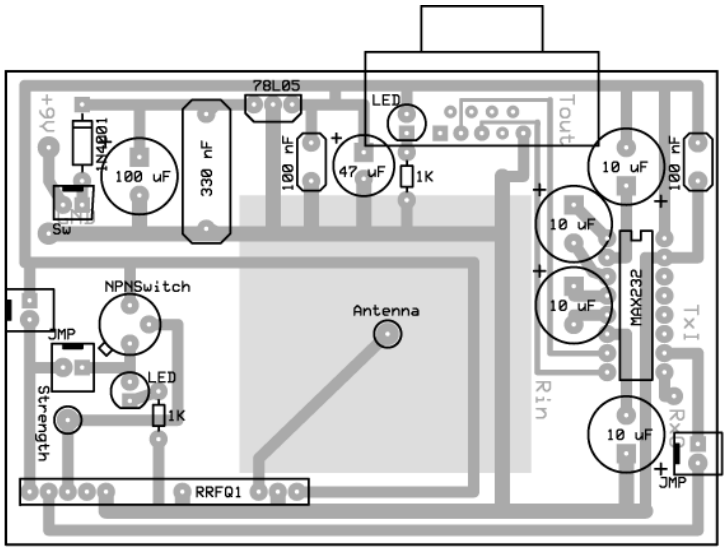


Figure 72. PCB Layout of the receiver circuit of the wireless DAQ system

The image on the left shows the actual form of the receiver circuit, with the RRFQ1, antenna, ground plane, and connectors, as well as a switch. It is powered by a standard AC/DC adapter set to 9V. It indicates whether it is on (yellow LED) and whether a transmitter is on (red LED), and is connected to a PC using a serial cable.

Figure 73. Left: Final form of the receiver circuit of the wireless DAQ system, right: receiver circuit hooked to a PC

6.3 Consuming the received digitized values

Eventually, what we receive from the wireless DAQ system is a serial character stream. What we need, however, is the current (or latest) numerical value of each of the analog channels. Thus, to consume, or utilize the values provided by the system, we need to write a decoder for the serial stream, which will extract the needed values from the character stream and make them available as numbers in the programming platform where it is needed. In this case, that platform is Max/MSP.

6.3.1 Writing a serial decoder for Max/MSP

As noted before, the serial character data stream that is generated by the PIC is organized into 35 byte words, which follow this format:

0-7	8-9	10	11-12	13	14-15	16	17-18	19	20-21	22	23-24	25	26-27	28	29-30	31	32	33-34
'U' (85)	'@0'	Ch0 value	'@1'	Ch1 value	'@2'	Ch2 value	'@3'	Ch3 value	'@4'	Ch4 value	'@5'	Ch5 value	'@6'	Ch6 value	'@7'	Ch7 value	';'	CR + LF

Most of the data in the packet is redundant, the only information we need is the numeric sampled value of each analog channel, which is stored in character positions (starting from 1) 11, 14, 17, 20, 23, 26, 29 and 32. These we need to make available as numbers in Max/MSP.

To begin with, Max has its own native object that deals with serial communication, called serial. This object outputs the latest data received through the serial port each time it receives a bang. The communication is set to 4.8 Kbps through the argument, 8N1 is implied by default. The received character stream is sent to the subpatcher dec_direct that represents the serial decoder.

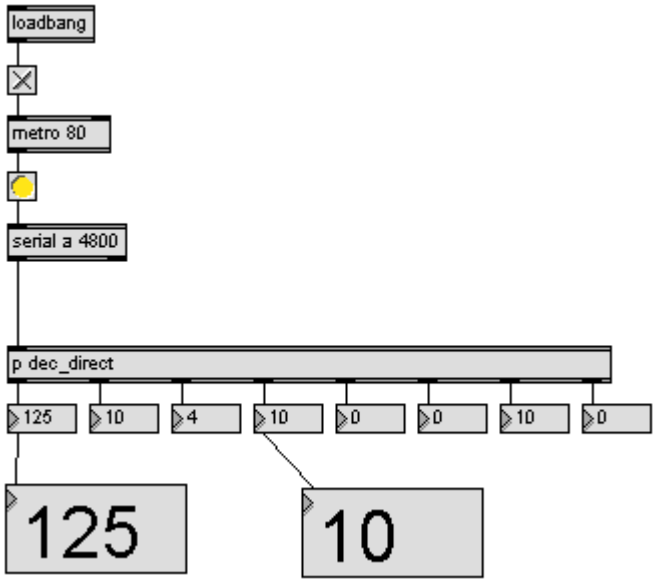


Figure 74. Max sample patch that utilizes the decoder (dec_direct) and provides the numeric sensor values

There are eight outputs from the decoder, each of which carry the numerical value of corresponding analog channel. In order to discuss the decoder, we should realize that the serial object output all characters received since the last bang, hence it must be banged continuously. The metro object performs that function with a period of 80 ms. Then, the received characters are output as integers ranging from 0-255 (meaning, as an ASCII code), one at a time in a sequence. Such sequence of integers enters the dec_direct subpatcher input. The contents of the subpatch are presented below:

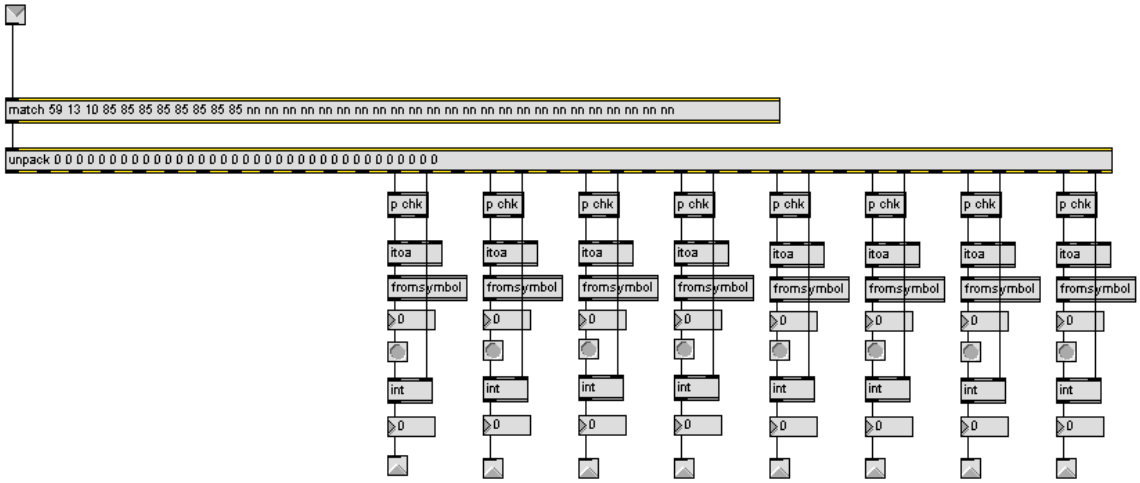


Figure 75. Contents of the serial decoder subpatcher

The sequence of integers starts filling the match object. The match object can be seen as a buffer, with as much allocated space as there are arguments. Here we have 35 arguments, corresponding to the 35 bytes we expect in a word. As each integer comes in it fills the next position available; if there are no positions available, the integer is pushed onto the stack, and the last character (received 35 bytes ago) is popped out. Thus the stream is windowed. When the contents of the window match the arguments of the match object box, the entire buffer is sent on the output of the match object as a list, and the buffer is thereafter cleared. The arguments of the match object start from the ; character, and then contain the CR LF, and the 8 bytes of 85. So the match arguments are the ASCII codes for this sequence: 59 13 10 85 85 85 85 85 85 85 85 85 followed by nn, which means “don’t care”. Thus when this sequence is encountered, list representation is sent on the output and the buffer is cleared. This is also one form of error checking, since if one of these characters is received wrongly, the entire word will be dropped.

The list of integers thus sent, is received by an unpack object, which unpacks it into 35 separate numbers, of same values as the ASCII codes. This is actually the final ste of the decoding process – all we need to do is to extract the numbers at positions 11, 14, 17, 20, 23, 26, 29 and 32 away from the first 85 byte ('U'), and pass them through an output. Here however, they are passed to the right outlet of an int object box, which acts as a memory for this value; the other inlet of this box acts as a trigger – when a bang is received, the remembered value is output. Here we have another form of error correction, which is not strictly speaking necessary – we know that one character before the channel values, the channel number is sent. Since it can only be a numeric character from '0' to '7', the ASCII code will fall between 48 and 56. The subpatcher chk checks if this is true, and in that case sends a bang:

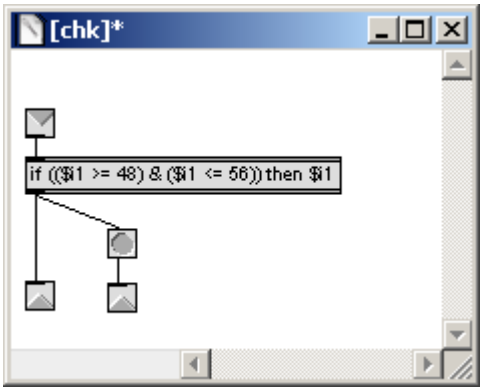


Figure 76. Contents of chk subpatcher

If that is the case, then the boxes itoa and from symbol serve to demonstrate how do we get the say number 2, from the character '2' (or rather, its corresponding ASCII code) and they serve no other purpose; in addition the bang is a signal that the channel value remembered in the int box, is probably OK and should be sent via its outlet. Let's remember here, that the time it takes for an entire word to be sent is about 84.6 ms. At best, we can expect that all the characters in the stream will be decoded correctly. Even in this case, the algorithm is such first we wait for the match buffer to be filled, and then if all is ok, we provide all the decoded values simultaneously. This means that in best case, the minimum time between two refreshes of the decoded values is bound by the 84.6 ms, which is the time required to send one 35 byte word. Additional delays could be added to this time, according to the scheduling situation that Max has to handle.

Let us mention that this metaphor of a decoder object, which provides an independent integer outlet for each data acquisition channel, is exactly the same approach implemented by the driver object for the Teleo, which is called t.intro.ain.

The decoder represents a complete implementation in a graphic programming environment that Max/MSP is. As such, it provides a metaphor on the aspect of data decoding, which is a bit more easier and more illustrative than the traditional C methods. However, we should note that such implementation would also lack in performance, since the Max scheduler dictates the execution of the object box functions, and there is no guarantee that the timing will suffice. Although no significant dropping of packets was observed, it must be mentioned that the Max serial object itself is very sensitive, especially to the kind of uncertain and jittering signals that this system produces. Once it experiences what is probably a framing error, the Max serial object does not recover, and Max needs to be restarted. All this being said, the utilization of the data acquisition signal in Max/MSP would probably benefit more from a dedicated serial driver and decoder, implemented as a Max external object and coded in C++, hopefully allowing a slightly better priority schedule for the serial decoding routine, and greater tolerance to framing errors.

7 Usage and performance

As mentioned previously, the development of a multichannel wireless data acquisition system was initiated by the VR research project. As a working prototype got implemented, so arrived the opportunity to use it in the context of the research project.

The context of the VR research project was an investigation into the responses of test subjects in a visual and auditory virtual reality environment. The test subject was equipped with a head mounted display, which rendered the visual VR feedback. An 8-channel speaker system was set up that rendered the auditory VR feedback. The test subject was also equipped with a magnetic tracker sensor, which was attached to the head mounted display. Both the tracker and the head mounted display were wired to a PC Linux workstation, running a VR++ application, which accepted the magnetic tracker information and rendered photorealistic 3D VR visuals (this platform is known as Benogo). The test subject was also equipped with the wireless footstep controller, which consisted of a pair of sandals with attached FSRs, which were connected to the prototype transmitter circuit, described previously. The transmitter circuit was placed in a pouch, facilitating a possibility to quickly equip the test subject with both the sandals and the transmitter pouch. The prototype receiver was connected to a Windows XP PC, running the Max/MSP patch that was rendering the auditory feedback. At the same time, TCP/IP network connection was established between Benogo on the Linux workstation, and the Max/MSP patch on the Windows PC – and the magnetic tracker data was sent from Benogo to Max/MSP using this connection. The purpose of the Max/MSP patch was to acquire the footstep sensor data, calculate whether a step has been made, in which case a footstep sound was trigerred. This footstep sound was processed according to the positional magnetic tracker data, received from the Benogo PC via TCP/IP, and 8 independent channels of audio were rendered as a result, which provided an experience of a spatial sound.

The research was conducted during five days, during which the prototype was used constantly, so this was a good testing opportunity for the performance of the developed system. To begin with, we can say that the devices were working for at least 6 hours each day. The transmitter, which is battery operated, stopped working on the

fourth day after which the battery needed to be replaced. Thus, a rough estimate is that the transmitter design allows for about 24 hours of continuous operation on a single 9V battery, which is not all that much after all (but still beats the 10 hours offered by the Kroonde). We have already discussed that the inherent refresh rate of the sensor data for the prototype system is about 84.6 ms, which is significantly more than commercial devices used for the same purpose. Yet, as an initial observation, people did not seem to be bothered or distracted by that delay – possibly because the information acquired was in any case a triggering one, rather than a continuous one. The test subjects had a freedom of movement of not more than a 4-meter radius around the receiver. Hence, the wireless link was not tested for length, as it was not necessary for the research project; within those short limits, no problems were directly experienced related to the range.

The severe problems were mostly actually due to the instability of the decoded serial code. Probably because of this, framing errors occurred often, and it comes out that the Max/MSP serial object is quite sensitive to them, and once such error occurs, the serial object cannot recover and Max needs to be restarted. Even worse is that the Max/MSP patch experienced severe crashes seemingly at random, which often times suddenly interrupted a running test and rendered it unusable. From this perspective, there could be many factors that attributed to the crashes – the fact that the patch had to thread between both TCP/IP and serial communication, as well as the fact that using the communicated parameters, eight independent channels of CD-quality audio were to be rendered in realtime. On the other hand, there were periods where the entire system would work without a crash or a problem, continuously for almost an entire hour. It is yet not clear whether any electromagnetic intereference may have made the wireless communication more unreliable.

It was also observed that if an analog input was left to float (unconnected), then a change acquired on the channel right next to it, would also be effectuated on the floating channel. This was solved by using odd channels (like 1 and 3) as analog input for sensor connection - and the even channels between them were then connected to ground using the connectors and wire. Interestingly enough, this effect was more obvious when using an FSR voltage divider; when using a simple potentiometer the data acquisition appeared otherwise quite stable, the few times it was tried. Rumours were also heard that the Kroonde behaves in the same manner with FSR voltage dividers, but that was not investigated.

We should finally mention that the similar organization with the Teleo in terms of hardware interfacing (via connectors) and software interfacing (in terms of the Max/MSP serial decoder), should make it easy for anyone that has used a sensor with a Teleo in an interaction context, to simply port the solution to the prototype hardware wireless platform – hence the usage context pretty much covers all areas where the Teleo could be applied as well.

8 Conclusion

The prototype for a wireless multichannel data acquisition system, implemented in a transmitter and a receiver circuit, which were developed in this project, can be evaluated on several levels.

As far as the problem formulation of this project, as outlined earlier in this report, is concerned, the developed prototype circuits answer the problem formulation – a working prototype is developed, using the minimum number of components and with minimum additional hardware expenses. Using only components from the school store was not possible on two occasions – once due to the author's own ignorance, which increased the cost for about 40 US$ per device; and several components had to be purchased externally, which further increased the cost for 4 US$ per device; still, even this cost represents about 3% of the cost of purchasing an additional Kroonde. Given the limited time allocated for development, in these terms the development is a success.

As far as the original demands of the VR research are concerned, they are answered partially. A working platform is provided which implements a wireless footstep controller. In the sense of the interaction required (the test subject should have the freedom of movement), that is satisfied – the transmitter can be worn on a belt pouch, there is an implied wire connection with the FSRs on the sandals, however, the transmitter, the sensors and the sandals translate as the test subject translates, so there is freedom of movement, as there isn't a wired connection between the transmitter and the platform which processes the data acquisition signals. However, the prototype grade quality of the developed hardware, which in essence rendered them highly unreliable, resulted with frequent crashes of the API (Max/MSP) that

processed the interaction input – which made the actual research much more difficult. We may also add that the large refresh period of about 84.6 ms is also a parameter that is undesirable from the perspective of the VR researchers.

Here we can recognize an essential difference of scope, when approaching HCI problems. A designer from the traditional perspective of HCI, starts with the human actions and needs first, and accounts for the usability issues - then provides a high level software solutions which implements the design. That is however, based on existing hardware, which is both reliable and available. In this sense, the engineer has the machine perspective – the engineer needs to take a lot of physical aspects, and provide first and foremost, a working solution – and then also a reliable solution. As this report shows, the scope of theories involved with providing a hardware solution are quite different than the scope of theories a HCI designer would use to evaluate a usability of a given interaction process. In essence, that can be rephrased by the motivation for this projects. The VR designers evaluate the usability of an interaction solution, and demand wireless options to meet their demands. On a hardware level, however, the question of implementing this wireless option in the given context, then delves into a broad scope of engineering theories, required just to make a concept of a system that is possible to implement and that would work.

We could speculate on the reasons for such development. On one hand, only a couple of decades back, the technology we have today was simply not available. Any sort of machine design, and with that, the interaction design for its operation, demanded a lot of resources – which were obviously, first and foremost invested into making a given machine work, and thereafter efforts are invested into making the operation reliable. As this report also shows, that involves many different consideration – and those regarding power, for instance, tend to take precedence when considering the reliability of a given circuit. That being said, we may speculate that historically, it simply was not economical to put more effort into interaction design, considering the efforts needed just to make the machine work reliably.

As the technology develops, we arrive at a situation where suddenly a lot more technology is available, with standardized interaction facilities. Here we primarily have the expansion of the PC with its standard interaction peripherals like keyboard monitor and mouse. The technology not only becomes broadly available, but is also more reliable; and hence it makes sense to completely abandon the technical issues

inherent in HCI processing and focus on the usability from the perspective of the human user. Added to that we have the Moore Law development – as we expect more processing power and smaller size of technology, it becomes easier to establish high level programming paradigms, where programmers do not necessarily have to bother with technical issues like moving blocks of memory or releasing resources.

It is this development, however, that also stimulates the continuous development of HCI. As more technology options become available, HCI designers attempt to break free from the constraints of the classic mouse and keyboard, and provide creative interactive solutions. However, as this project shows, they are still limited by the available technology that will implement such custom solutions. Thus, in a case of a HCI design that extends over the boundaries of traditional HCI, we recognize the need for a closer level of acquaintance with engineering concepts – in relation to understanding hardware that would implement a certain HCI design.

That is one of the findings of this report – basically, the project is directed towards a custom development, due to the failure to identify the data acquisition technology as the backbone hardware for interaction processing. The uncertainty that this theoretical issue brings about in the development, results with only an initial prototype quality, which may not adhere to all of the quality requirements of the research project – this means that ultimately it is not perfectly suited as a technology backbone for the HCI solution that the researchers designed. However, the result of the development also address the issue, in the sense of providing a platform that works, simple enough for implementation by students that may not necessarily have all the necessary engineering background. In that sense, the academic value of this prototype is probably greater than the value as an actual technology to use in a HCI solution.

In essence, it would be expected that extending the HCI designs into custom interaction solutions, implies that the HCI designer needs also to be acquainted with possible technologies that could be used for a custom implementation. On the other hand, as the development progresses into custom interaction, the need for a closer cooperation between engineers and HCI designers is obvious. We could postulate that a HCI designer, which is acquainted with the basic engineering problems involved with interaction technology on a hardware level, would be able to at least communicate better with engineers.

And it is here that we could declare an educational value for the developed prototype. Namely, this report is written in a tutorial form, trying to encompass the most important issues from the different theories that find place in a hardware development project. As it can be seen from this project, the scope of the theories used is broad; it is however not too deep. We use mostly basic, introductory terms in this report – and the mathematics used is on a level of linear equations. The intent is thus to show, that through a project that demands implementation of hardware whose specific intent is use in interaction, we have the opportunity to present the most important aspects of the theories involved, in a format that does not require formal engineering training.

In addition, the simplicity of the circuits involved, allows for almost direct establishment of a relationship between theoretical concepts and practical implementation – for instance, the waveform of a TTL synchronous serial signal completely corresponds to the theoretical definition of a serial frame, and such a waveform can be physically observed with an oscilloscope (though one with a memory, for more precise observations). Electrically, all functional parts were presented as independent modules and then connected to allow for the total functionality – which makes the process of schematic reading much easier. Building the programmer, being involved with the process of microcontroller programming, building the wired circuit and then building the wireless transmission illustrates a wide range of theories through implementation: issues discussed include power, signal interfacing and adjustment, as well as communication all the way down to the physical layer of the OSI model; physical difference between asynchronous and synchronous communication, relationship between oscillator clocks and program execution; signal modulation and RF propagation issues; as well as physical influences on the performance of the system.

In most engineering courses, these theories are introduced in detail, yet the relationship to actual hardware implementation is often obscured; the following quite, from an author discussing hardware serial communication maybe describes the situation best: “I do believe that there is much that could be taught about computer networking by students experimenting with serial data communication … Every layer of the OSI model could be demonstrated in a manner that students would learn from first-hand experiences why certain rules/systems have been implemented on the

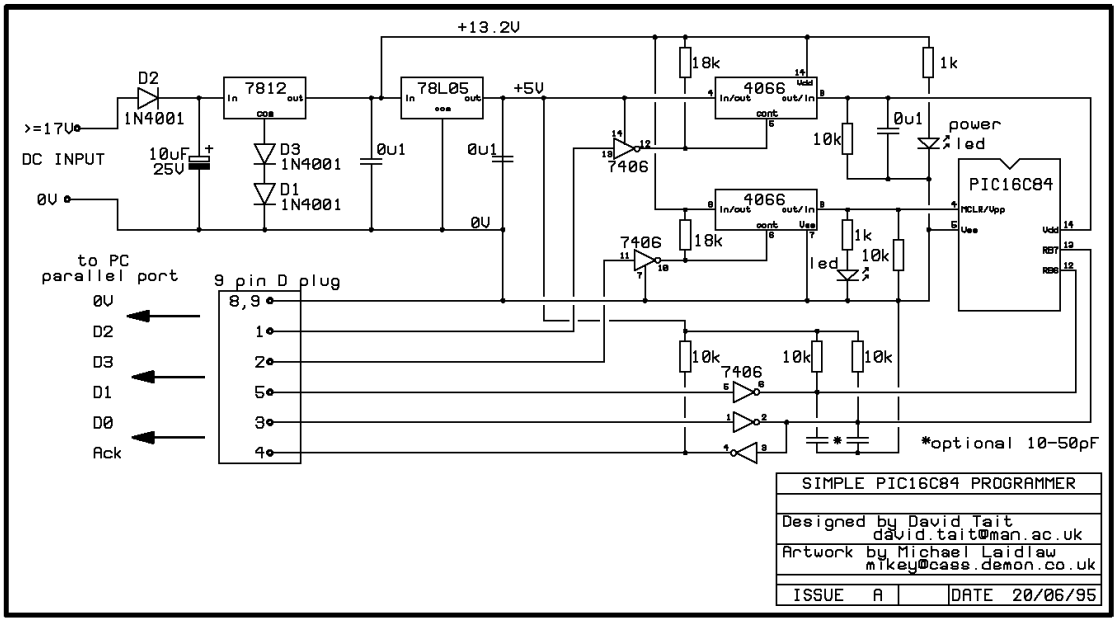
Internet, what standards documents mean, and perhaps even participate in creating standards documents … this is a topic that is seldom taught at a university, and usually only in passing when they are rushing through a whole bunch of other protocol suites. Software developers are usually introduced to this topic by having their supervisor dump a bunch of specification documents on their desk, a driver disk with API documentation, and perhaps a typically short deadline in order to get something working that should have been working sometime last year. Software developers who really understand serial data communication are worth gold, and often even these developers only learn just enough to get the immediate job done [78]". Having this in mind, maybe the greatest value of the developed prototype is its simplicity, its low cost and the fact that it can be made to work; as well as the fact that a lot of theories are introduced, which however remain at a basic level only. This makes it suitable for use in HCI education, say as an exercise which introduces all these theoretical perspectives through practice.

Considering the system itself, we are aware that the chief problem is the unreliability of the serial communication. Even at the level of discussion of this report there are several thing that could be undertaken. For instance, the communication speed can be increased to the maximum 9.6 Kbps, once we have obtain a certain mode of operation for the RF devices. The format of the word packet can be optimized and lowered from the current 35 bytes – for one, an 8 byte preamble for each word is probably not necessary anymore. Framing errors could be accounted for by trying to use two stop bits instead of one. A gate circuit could be implemented at the receiver, which lets the demodulated data out signal through only if there is a transmitter online, thus eliminating the random binary signal which is sent via serial when the transmitter is offline, and which regularly locks the PC. As jiterring is an issue, we could first start by using the hardware UART instead of a software bit-banging routine. The decoder for Max/MSP could be optimized, and indeed written as a separate object. Experimenting with different pairs of transmitter /receiver RF modules is also viable; consider that the mentioned Radiometrix devices claim a data link speed of 56 Kbps – which would certainly bring the prototype to a whole new level. Experimenting with microcontrollers with built in transmitters is also viable, as they will essentially contain the whole transmitter circuit (besides the power section) on one IC. However, going beyond that, into trying to develop a production quality device - for example through trying to design a more efficient wireless link by using better antennas - will require more serious engineering efforts.

In conclusion, we can say that the development in this project resulted with a hardware implementation of a wireless, distributed, multichannel data acquisition system that is relatively simple and inexpensive to build. In regards to the interaction context that originated the development, this system can be used to implement a wireless footstep controller, as it conforms to being portable and wearable; however, its current prototype grade quality make it generally unreliable, and undesirable for continued use. However, that can be compensated by the fact that this hardware implementation, requires introduction of a broad range of engineering theories, yet on an introductory level; coupled with the simplicity and low cost of the system, this makes it viable as an educational exercise for students of interaction, that would serve the purpose of introducing basic engineering concepts through practice; and in this way, the prototype may serve as one educational base in the field of HCI, which building on its traditional perspective, tries to overcome the bounds of standardized human computer interaction, with custom sensor-based interactive input designs. It can be imagined that such development will require closer cooperation between HCI designers and engineers; hopefully this projects shows one of the potentials for better coupling between these two professional fields.

9 Appendix

Appendix A. Schematic of the classic Tait hardware programmer



Originally on David Tait's PIC archive, mirrored on http://www.nomad.ee/PIC/pp.gif

Appendix B. BASIC code listing for microcontroller data acquisition program

```
Symbol ad_action = ADCON0.GO_DONE  'set new name for A/D conversion start bit
Symbol display = PORTB  'set new name for PORTB used to display the conversion
result

Define SEROUT_DELAYUS = 500  'define serial delay

Dim channel_id As Byte  'declare a variable
Dim channel_val As Byte  'declare a variable
Dim data(8) As Byte

TRISB = %00000000  'set PORTB pins as outputs
TRISA = %111111  'set PORTA pins as inputs

ADCON0 = 0xc0  'set A/D conversion clock to internal source AND SELECT CHANNEL 0
ADCON1 = 0  'set PORTA pins as analog inputs
High ADCON0.ADON  'turn on A/D converter module

' just a test sequence for start
```

```
' alternate high and low one second each
High PORTB.4
WaitMs 1000

Low PORTB.4
WaitMs 1000

High PORTB.4
WaitMs 1000

Low PORTB.4
WaitMs 1000

main:

    'acquire data, collect into packet data array

    For channel_id = 0 To 7 Step 1  'for-next loop

        ADCON0 = ShiftLeft(channel_id, 3)    'SELECT CHANNEL channel_id and
        ADCON0 = ADCON0 Or 0xc0              'set A/D conversion clock to internal
source
        High ADCON0.ADON  'turn on A/D converter module, shift does not preserve the
last bit!

        Gosub getadresult  'go to conversion routine
        'after conversion routine is back, result is in register ADRES for 16F74
        data(channel_id) = ADRES  'save the result of conversion in the given pos.
in data array

    Next channel_id

    'done collecting, format and send packet data array

    'serial, first preamble - send 85 01010101 7 times
    Serout PORTB.4, 4800, 85, 85, 85, 85, 85, 85, 85, 85

    'direct the entire packet in one line
    Serout PORTB.4, 4800, "@1", data(0), "@2", data(1), "@3", data(2), "@4",
data(3), "@5", data(4), "@6", data(5), "@7", data(6), "@8", data(7), ";", CrLf

    Goto main  'repeat forever
End

getadresult:  'A/D conversion routine
    High ad_action  'start the conversion
    While ad_action  'wait until conversion is completed
    Wend
Return
```

List of references

[1]. Nordahl, Rolf, "Auditory rendering of self-induced motion in virtual reality", M.Sc. project report, Dept. of Medialogy, Aalborg University Copenahgen, 2005

[2]. Making Things homepage, http://www.makingthings.com/

[3]. OSI model, Wikipedia, http://en.wikipedia.org/wiki/OSI_model

[4]. 7-layer OSI Model, NetworkClue, http://www.networkclue.com/routing/tcpip/osi-model.php

[5]. Teleo Starter Kit User Guide, http://www.makingthings.com/products/documentation/teleo_intro_user_guide/index.html

[6]. "Introduction to Data Acquisition", Tutorial, Development Library, National Instruments, http://zone.ni.com/devzone/conceptd.nsf/webmain/AE2A7B85BD4785D586256F620066EFF4

[7]. Ozkul, T., "Data acquisition and process control using personal computers", Marcel Dekker Inc., New York, 1996

[8]. James, K. "PC Interfacing and Data Acquisition: Techniques for Measurement, Instrumentation and Control", Newnes (Elsevier Science), Oxford, 2000

[9]. Deliwala, S., Introduction to Data Acquisition Systems and LabVIEW, http://www.seas.upenn.edu/ese/rca/software/Labview/daqlvOverview.html, Electrical and Systems Engineering undergraduate lab notes, University of Pennsylvania, 1999

[10]. Webopedia, word definition for "distributed processing", http://www.webopedia.com/TERM/D/distributed_processing.html

[11]. Georgetown University, Data Warehouse: Glossary, http://uis.georgetown.edu/departments/eets/dw/GLOSSARY0816.html

[12]. Wu, Jie, "Distributed System Design", CRC Press LLC, Boca Raton, 1998

[13]. Kent, A., Williams, J. G., Kent, R, "Encyclopedia of Microcomputers", Volume 4: Computer-Related Applications: Computational Linguistics to dBase, Marcel Dekker Inc., New York, 1990

[14]. "Instrument Control Technologies for Any Bus, Any Language", Tutorial, Development Library, National Instruments,

http://zone.ni.com/devzone/conceptd.nsf/webmain/56C6CE2C33E5EAA886256FFE00588EC0
[15]. NCTE (National Centre for Technology in Education) – Glossary, http://www.ncte.ie/ICTAdviceSupport/AdviceSheets/Glossary/
[16]. Naidu, P. S., "Sensor Array Signal Processing", CRC Press LLC, Boca Raton, 2001
[17]. Rees, W.G., "Physical principles of remote sensing", Cambridge University Press, 2001
[18]. The Kroonde Gamma webpage, Le Kitchen website, http://www.la-kitchen.fr/kitchenlab/kroonde-en.html
[19]. Wikipedia, UDP, http://en.wikipedia.org/wiki/User_Datagram_Protocol
[20]. RadioPod page, IPC Systems website, http://www.ipcsystems.ltd.uk/radiopod.html
[21]. Portable DAQ Solutions from Measurement Computing Corporation, webpage, http://www.measurementcomputing.com/portable_DAQ.html
[22]. HCI Resources: Online courses/lecture notes, Linköping University, http://www.ida.liu.se/~miker/hci/course.html
[23]. Putnam, W., Knapp, R. B., "Input/Data Acquisition System Design for Human Computer Interfacing", online course notes, Stanford University, 1996 http://ccrma.stanford.edu/CCRMA/Courses/252/sensors/sensors.html
[24]. Cook, P. R., "Real-Time Computing For Human Computer Interfacing", online lecture notes, Princeton University, http://soundlab.cs.princeton.edu/learning/tutorials/RealTime/realtime.html
[25]. Kilosamples or kiloHertz, online buletin board discussion, Council of Science Editors, http://www.councilscienceeditors.org/services/bboard1/thread.cfm?threadid=70&messages=5
[26]. Liptak, B. G., "Instrument Engineers' Handbook: Process Software and Digital Networks", CRC Press LLC, Boca Raton, 2002
[27]. “Microcontroller”, From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Microcontroller
[28]. “Integrated circuit”, From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Integrated_circuit
[29]. “Computer”, From Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Computer

[30]. “Universal asynchronous receiver transmitter”, From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/UART
[31]. MPLAB Integrated Development Environment webpage, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002
[32]. PICSTART Plus Development system webpage, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023547&part=DV003001
[33]. “Instruction set”, From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Instruction_set
[34]. "Machine code", From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Machine_language
[35]. Picc-Lite homepage, http://www.htsoft.com/products/PICClite.php
[36]. David Tait homepage, http://www.people.man.ac.uk/~mbhstdj/piclinks.html
[37]. http://akizukidenshi.com/images/org/pic16f74.jpg
[38]. "The Basic PIC 16F84 Circuit", http://www.sonofsofaman.com/hobbies/electronics/pic/16f84/circuits/basic.asp
[39]. Microchip rfPIC12C509AF home page, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010382
[40]. Infineon SAB 80C535, http://www.infineon.com/cgi-bin/ifx/portal/ep/programView.do?BV_SessionID=@@@@2109288059.1130838028@@@@&BV_EngineID=ccckaddgdgdikhjcflgcegndfifdfoh.0&channelId=-64446&programId=33153&programPage=/ep/program/document.jsp&pageTypeId=17099
[41]. PIC 16F74 page, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010220
[42]. PIC 16F74 datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/30325b.pdf
[43]. PICC Lite page, http://www.htsoft.com/products/PICClite.php
[44]. Parallel Port PIC Programmer page, http://www.oshonsoft.com/picprog.html
[45]. EPROMs amd Memory Chip Fundamentals, Andromeda Research Labs website, http://www.arlabs.com/help.htm
[46]. PIC Simulator IDE homepage, http://www.oshonsoft.com/pic.html
[47]. Intel HEX-record Format, http://www.cs.net/lucid/intel.htm

[48]. WinPicProg 1.95c, Nigel Goodwin, http://www.winpicprog.co.uk/
[49]. WIN PIC Programmer Nov-2005, DL4YHF's PIC Programmer for Windows, Wolfgang Buescher, http://www.qsl.net/dl4yhf/winpicpr.html
[50]. PICALL Programmer v0.16, Bojan Dubaj, http://www.picallw.com/
[51]. IC-Prog 1.05D, IC-Prog Prototype Programmer, Bonny Gijzen, http://www.ic-prog.com/index1.htm
[52]. PIC16F7X FLASH Memory Programming Specification, http://ww1.microchip.com/downloads/en/devicedoc/30324b.pdf
[53]. Silicon Graphics Fuel Visual Workstation Hardware User's Guide, http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=bks&srch=&fname=/SGI_EndUser/Fuel_UG/sgi_html/apa.html
[54]. DB-25 parallel connector pinout, http://www.bbdsoft.com/parallel.html
[55]. Interfacing to the IBM-PC Parallel Printer Port, http://www.doc.ic.ac.uk/~ih/doc/par/
[56]. Parallel Direct Cable Connection, http://www.firewall.cx/cabling_dcc-parallel.php
[57]. Zhahai S., Interfacing the IBM PC Parallel Printer Port FAQ, http://www.infonewsindia.com/pinout/ibmlpt.txt
[58]. Engdahl, T., Parallel port interfacing made easy: Simple circuits and programs to show how to use PC parallel port output capabilities, http://www.epanorama.net/circuits/parallel_output.html
[59]. Erlich Industrial Development, Corp., Voltage regulator, http://www.eidusa.com/Electronics_Voltage_Regulator.htm
[60]. Ross, K., Basic Circuits - Bypass Capacitors, http://www.seattlerobotics.org/encoder/jun97/basics.html
[61]. Capacitor Design Data, and Decoupling Placement, How-To, http://www.interfacebus.com/Design_Capacitors.html
[62]. ExpressPCB page, http://www.expresspcb.com/
[63]. Electro Tech Online discussion, http://www.electro-tech-online.com/viewtopic.php?p=116331
[64]. Fairhurst G., Asynchronous Communication, University of Aberdeen, EG2069 course notes, http://www.erg.abdn.ac.uk/users/gorry/eg2069/async.html
[65]. Strangio, C., "Data Communications Basics", CAMI Research Inc., http://www.camiresearch.com/Data_Com_Basics/data_com_tutorial.html

[66]. White, T., “The Electric Telegraph (1860-1914)”, United States early radio history online articles, http://earlyradiohistory.us/sec002.htm
[67]. "Serial Communication", Asynchronous Communications Guide online book, IBM, http://publib16.boulder.ibm.com/pseries/en_US/aixasync/asycomgd/Serial_com_AsynComm.htm
[68]. www.wgcu.org/watch/hdtv_glossaryofterms.html
[69]. www.3gnewsroom.com/html/glossary/s.shtml
[70]. Introduction to serial communications, Taltech Software, http://www.taltech.com/TALtech_web/resources/intro-sc.html
[71]. O'Sullivan, D., Igoe, T., "Physical Computing: Sensing and Controlling the Physical World with Computers", Thomson Course Technology, Boston, 2004
[72]. Asynchronous serial communication overview, Quatech Technical Support, http://www.quatech.com/support/comm-over-asyncserial.php
[73]. Interfacing the Serial / RS232 Port, Beyond Logic online tutorial, http://www.beyondlogic.org/serial/serial1.htm
[74]. Logic signal voltage levels, http://www.allaboutcircuits.com/vol_4/chpt_3/11.html
[75]. Serial Communication General Concepts, http://digital.ni.com/public.nsf/allkb/2ad81b9060162e708625678c006dfc62?OpenDocument
[76]. Practical Tips on Serial Communications, http://www.seetron.com/ser_an1.htm
[77]. Max232 (MAX220-49) Datasheet, http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf
[78]. Programming:Serial Data Communications, From Wikibooks, the open-content textbooks collection, http://en.wikibooks.org/wiki/Programming:Serial_Data_Communications
[79]. Bies L., RS232 serial null modem cable wiring and tutorial, http://www.lammertbies.nl/comm/info/RS-232_null_modem.html
[80]. Quartz Crystal Application Notes, Abracon Corporation, http://www.beckwithelectronics.com/abracon/quartzan.htm
[81]. Angelo, R., "Care and Feeding of the PIC16C74 and Its Peripherals", FACT003 application note, http://ww1.microchip.com/downloads/en/AppNotes/00840a.pdf
[82]. Jacobs A. J., Alarm Clock (16F74 project webpage), http://www.obelisk.demon.co.uk/electronics/alarm.html

[83]. Modulation - From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Modulation
[84]. Saunders G., Data Comm Basics webpage, http://www.people.vcu.edu/~gasaunde/data_comm_basics.htm
[85]. Pulse-code modulation - From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Pulse-code_modulation
[86]. RF help: the basics of radio frequency design & integration, online collection of articles, Aerocomm company, http://www.aerocomm.com/RF/RF_basics.htm
[87]. TX3A RX3A datasheet, Radiometrix, http://www.radiometrix.co.uk/dsheets/tx3arx3a.pdf
[88]. RRFQ1 RTFQ1 datasheet, RF Solutions, http://www.rfsolutions.co.uk/acatalog/DS069-7.pdf
[89]. Vcc shoots up with antenna connected, RF solutions topic post, http://www.rfsolutions.co.uk/cgi-bin/forum/YaBB.pl?board=Modules;action=display;num=1105794318
[90]. RXQ1-noise on data output, RF solutions topic post, http://www.rfsolutions.co.uk/cgi-bin/forum/YaBB.pl?board=Modules;action=display;num=1112174688
[91]. Interfacing PIC and RFSullutions, RF solutions topic post, http://www.rfsolutions.co.uk/cgi-bin/forum/YaBB.pl?board=RFDesign;action=display;num=1114763095
[92]. RF Design Tips, RF solutions topic post, http://www.rfsolutions.co.uk/cgi-bin/forum/YaBB.pl?board=RFDesign;action=display;num=1110747126
[93]. RRFQ1/RTFQ1 @ 868 MHz - problems with serial?, RF solutions topic post, http://www.rfsolutions.co.uk/cgi-bin/forum/YaBB.pl?board=Modules;action=display;num=1132734208
[94]. Navy Electricity and Electronics Training Series, Module 10—Introduction to Wave Propagation, Transmission Lines, and Antennas, US Navy 1998, https://www.cs.tcd.ie/Stephen.Farrell/ipn/background/US-Navy-NEETS/Module10-14182.pdf
[95].
[96].
[97]. df