# MIDDLE EAST TECHNICAL UNIVERCITY

## Department of Computer Engineering

# CENG 491

## Initial Design Report

## ComFuture Technology

**Orhan Tuncer 1250851**

**Ugur Turan 1348028**

**Guven Orkun Tanik 1347947**

**Sebnem Sonmezler 1298231**

**Hakan Okten 1250562**

# INDEX:

# 1. Introduction

## 1.1 Purpose of This Document

This document is prepared to summarize the efforts spent on the early design phases of our project, ComFuture Bluetooth Controller. It basically covers the initial design activities and establishes a basis for the detailed design phase. In this document design concepts will be undertaken in a general sense. The detailed design will ascertain the concepts in this report and so this report will form a basis for the final design report.

## 1.2 Definition and Scope of the Project

Our project is to design and implement a general-purpose controller device which will allow any suitably configured Bluetooth device to control the intended device which contains our controller module. Thus our controller will be modular, easily installed, compliant with the current standards in the area. We will design and implement developer libraries to use the controller and thus the end product will depend on the specific implementation. We are going to produce a detailed and easily understandable API and a broad spectrum of library functions.

The main speciality of the product will be its generic modules and compliance with other hardware units. By this way Bluetooth won't be a close-packed technology for other developers since by this product they can easily integrate Bluetooth property to their products with ease and a very little background.

## 1.3 Overview of the Project Properties

Since the project has some predefined concerns it is useful to explain these concerns in order to give a more precise understanding on the overall design. They are fundamental to understand the design goals which are explained in the next chapter (1.4).

### 1.3.1 Portable

Since we are designing a general purpose Bluetooth controller, the product has to be portable for a variety of devices.

If this functional requirement would not be satisfied, it will contradict with the main idea of a “general” purpose Bluetooth controller. In order to reduce the probability of the occurrence of this type of contradiction, we will consider a wide variety of devices that can be controlled by our product and try to meet their standards during the design phase.

### 1.3.2 Secure

Although there would be some applications for the Bluetooth controller which does not require security, we assume that our controller system should provide a level of security which can be defined by four fundamental elements: Availability, access, integrity, and confidentiality. If we assure that the connection between the controller and the device is secure under these conditions, no other third party device can interfere with the connection or interrupt, while the device and the controller is communicating without losing integrity and confidentiality.

As it is known by most of the cellular phone users interested in Bluetooth, any Bluetooth device can be realized by a phone when searched for devices. Since Bluetooth technology can communicate out of sight, this can be a critical problem in security issues. In today's cellular phones there is an authentication protocol for giving permission to connect to a Bluetooth device. This must be implemented in our product since the proof of concept can be used in critical places for security.

Also we want to add portability and generic issues to our system. As a result we must implement all the same protocols for the Bluetooth connection about security and in our libraries there must be additional security issues for the hardware which are wanted to be kept more private that a cellular phone.

### 1.3.3 Reusable

Our hardware will be easily pluggable and unpluggable. As long as the number of pins are enough it can be reprogrammed and used in an other context.

### 1.3.4 Easily Programmable

The generic component on the board will be the client application which is located on the PIC. It can be easily reprogrammed to accommodate any user needs. So it can be easily reprogrammed to be used in any context or situation. Our client side library will offer the necessary tools to develop an client application fastly and easily.

On the server side our Application Server will offer the necessary container to keep and process the server side applications. Our server side library will offer the necessary tools to develop a server application fastly and easily.

## 1.4 Design Goals

Our design goals are driven by the facts of simplification necessities of any hardware by the means of design costs and production costs, simplification necessities of software systems by the means of design costs and maintainability as well as the concerns about the satisfaction and demands of the end users.

### 1.4.1 Hardware Design

We divided our hardware design process into two parts consisting of selecting the suitable PIC, which has a high importance level in our designs including the software design which is highly connected to the PIC selection, and intermediate circuit design which is necessary to include the PIC to the current hardware, the Bluetooth development environment.

#### 1.4.1.1 PIC

We will be working with a PIC16F877A model PIC. Its relatively high internal capacity and embedded analog to digital converter will be used to simplify the hardware part considerably.

#### 1.4.1.2 Intermediate Circuit

We will use an extremely simple intermediate circuit which will consist of noise dampening bypass capacitors and voltage adjustment subcircuit to connect PIC and serial port of the Bluetooth controller.

### 1.4.2 Library Design

There will be two different libraries located at two sides of the architecture; server side and client side.  This division is required because the server side applications and the client side applications will be working on two different systems which also separate them by the means of programming languages. The server side library, as well as the application server itself, will be constructed on the Java Technology, on the other hand client side library will be developed with C language for PIC programming, which also defines the client side programming language. The details of client side programming and server side programming will be explained in chapter 4 “Library Design”.

#### 1.4.2.1 Client side Library

It will be used at the compile time of the client side application. It will contain the necessary tools to communicate with the Bluetooth controller. We tried to develop a compact but powerful library to be used at the client side that will perform the hardware related operations in order to give the application programmer a more logical view of the system.

#### 1.4.2.2 Sever side Library

It will be used to communicate with the client application. We tried to develop a compact but powerful library by not incorporating any redundant or interlaced functions. It will give the application programmer the ability of controlling system events and develop a better business context.

### 1.4.3 Application Server

Application Server is the main improvement we have added to the design. It will be the main container of the server side applications with ability to offer a better control to the system designer using our Bluetooth controller. It will be constructed on the Java Technology.  This improvement will give a more flexible working environment for both the application and system designers as well as the client application itself.

### 1.4.4 Server Side API

It will be used to coordinate the interaction with the application server. To improve the consistency and the security of the system, an application programmer can define some rules to be forced on the application by the server. It is essential if the server side application has various components and if a well defined business logic is required. Our server side API will give the designer these abilities by offering the complementary tools effectively using the abilities that the Application Server offers.

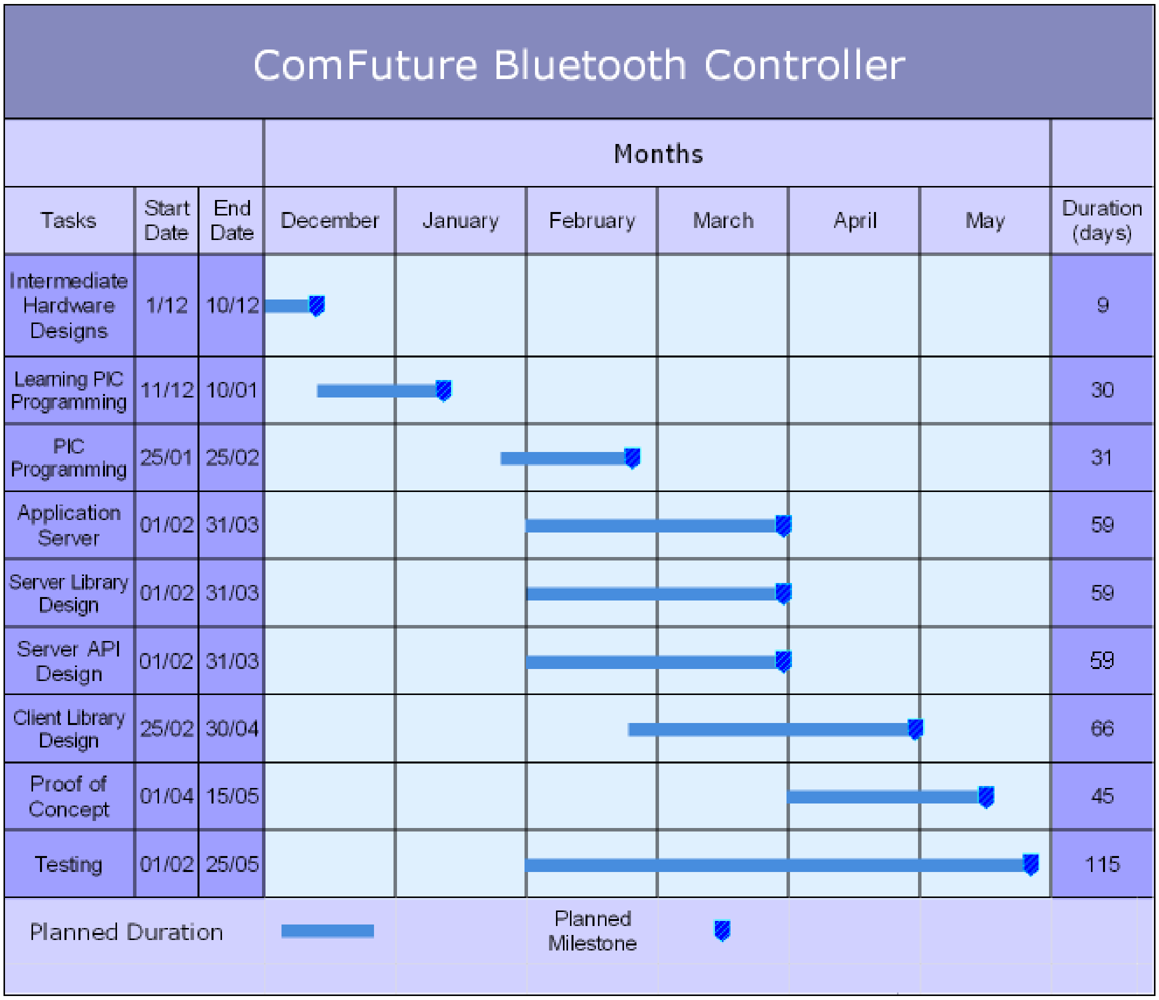
### 1.4.5 Proof of Concept

Our proof of concept will be a modified remote-controlled vehicle, which we will buy as a toy, but going to modify to meet our project needs. Since it will also be used as a testing and evolution tool of our design, hardware and software it needs to be simple but improvable. Some possibilities have already been started to evaluated and they will be explained later.

## 2. Project Schedule

We will start with intermediate hardware designs which will enable us to implement and try our PICs as we program.

We supplied our PIC programming software and hardware and also we purchased necessary bread boards and circuit components. This will allow us to concentrate on building our project rather than trying to decide on which part to use. As such we will proceed as learning and programming in order. We will do the peripheral programming first than concurrently start application server and Client programming. Also library design can not be separated from any of these. As we approach to the final phase of development we will concentrate on the proof of concept. Testing will be held during all of the development stages.



## ComFuture Bluetooth Controller

| | | | Months | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Tasks | Start Date | End Date | December | January | February | March | April | May | Duration (days) |
| Intermediate Hardware Designs | 1/12 | 10/12 | ▬▨ | | | | | | 9 |
| Learning PIC Programming | 11/12 | 10/01 | ▬ | ▬▨ | | | | | 30 |
| PIC Programming | 25/01 | 25/02 | | ▬ | ▬▨ | | | | 31 |
| Application Server | 01/02 | 31/03 | | | ▬ | ▬▨ | | | 59 |
| Server Library Design | 01/02 | 31/03 | | | ▬ | ▬▨ | | | 59 |
| Server API Design | 01/02 | 31/03 | | | ▬ | ▬▨ | | | 59 |
| Client Library Design | 25/02 | 30/04 | | | ▬ | ▬ | ▬▨ | | 66 |
| Proof of Concept | 01/04 | 15/05 | | | | | ▬ | ▬▨ | 45 |
| Testing | 01/02 | 25/05 | | | ▬ | ▬ | ▬ | ▬▨ | 115 |

Planned Duration ▬ Planned Milestone ▨

# 3. Architectural Overview

Our architecture has two main aspects. First one is Application Server and the second one is ComFuture Bluetooth Controller.

The Server Application in our previous model is replaced by an application server in our current model. The reason is to provide users a better intermediate tool between client application and server application. In this approach user can define multiple classes that can work together and also the client application can choose the class to work with.

In this scheme such a layout will be formed:

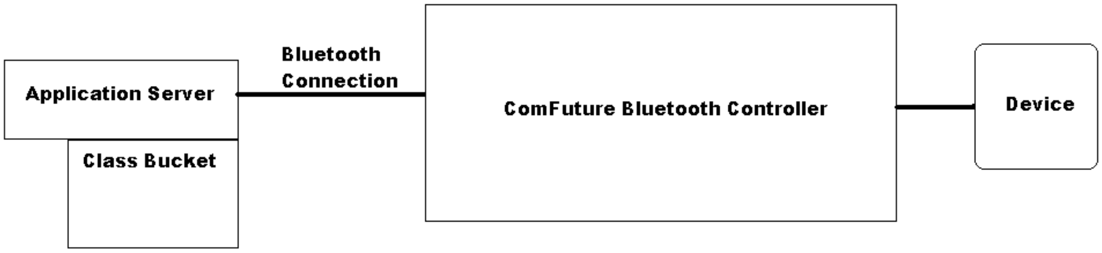


Figure - 3.1 Architectural Overview

In our Controller besides to Bluetooth communication circuit given to us we will implement the following circuits:

- Intermediate hardware to connect the serial port of the Bluetooth communication circuit. This circuit will generally condition the input to the PIC in our desired format.
- PIC16F877A as a container for the client application and intermediate processes.
- Intermediate hardware to distribute and format the output according to our specifications.
- Output Pins.

Figure 3.2 depicts this layout:

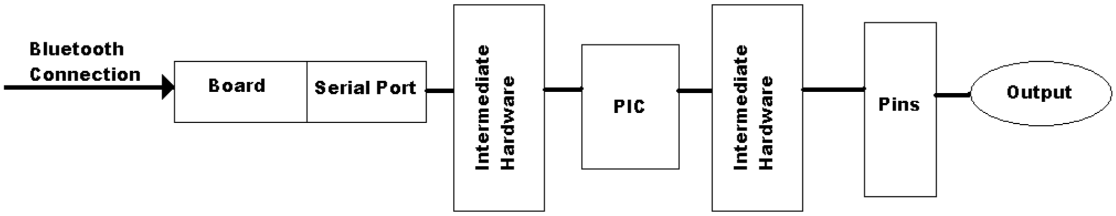


Figure - 3.2 ComFuture Bluetooth Controller Details

# 4. Library Design

There are two libraries in our design separated by the means of location, usage and programming language: client side and server side. The motivation of separating them is explained before so here we will be focusing on details of their designs.

## 4.1 Client-Side Library

Client Side Library is designed to fulfill the needs of application programmer. It consists of necessary tools which will hide the hardware connection details from the programmer. This library will be implemented in C language for PIC and the client side applications are restricted to be implemented in C language.

Client application will have the ability of choosing the server that it wants to work with and also the server application to handle the data that it sends. These powerful abilities will give the client application a more flexible working environment and the system designer more easily designable and maintainable environment.

Functions of this library and their explanations are given below:

| **FUNCTIONS** | **EXPLANATION** |
|---|---|
| bcnt blu_Connect (char* serverID, char* ClientID) | Opens connection to Blucon server. Returns Blucon connection object which defines the path of server application which is decided by the server. |
| int blu_classPref(bcnt* con, string class_name ) | Sends the request of working with a specific class located at the server. |
| int blu_Disconnect (bcnt* con) | Closes connection to Blucon server. |
| int blu_sendLayout (bcnt* con) | Sends the pin layout to the server in Blucon pins object format. |
| bpin blu_readLayout (void) | Reads the values of the pins and returns Blucon pins object. |
| int blu_writeLayout (bpin pin) | Sets the values of the pins. |
| void blu_onLayout (bfn* fnc) | The function to be called when new layout arrives. The function must be in bfn format. |
| void blu_onPinUpdate (bfn* fnc) | The function to be called when the values of the pins changes. The function must be in bfn format. |
| void blu_clientLoop (bmain* main) | Client side main processing loop. The function must be in bmain format. |

### Defined Function Formats

| |
|---|
| void bfn (bcnt* cnt, bpin pin) |
| int bmain (void) |
| void bcon (bcnt* cnt) |

## 4.2 Server-Side Library

Server Side Library is designed to fulfill the needs of application programmer by the means of defining events and give the programmer control over these events. It is main tool that enables the programmer to communicate with the client application. Since a server must only serve we did not provided any tools which will interfere the client application.

Server side applications will be java classes resting in the container section of the application server which uses our library functions to interact with the client.

Functions of this library and their explanations are given below:

| **FUNCTIONS** | **EXPLANATION** |
|---|---|
| int blu_sendLayout (bcnt* cnt, bpin pin) | Sends the pin layout to the client. |
| void blu_onConnect (bcon* con) | The function to be called when a new connection is requested. The function must be in bcon format. |
| void blu_onDisconnect (bcon* con) | The function to be called when a connection is closed. The function must be in bcon format. |
| void blu_onLayoutServe (bfn* fun) | The function to be called when new layout arrives. The function must be in bfn format. |
| void blu_serverLoop (bmain* main) | Server side main processing loop. The function must be in bmain format. |

**Defined Function Formats**

| |
|---|
| void bfn (bcnt* cnt, bpin pin) |
| int bmain (void) |
| void bcon (bcnt* cnt) |

# 5. Application Server

Application Server is the most powerful aspect of our design. It will be constructed on the top of the Java Technology and it will offer the necessary tools to develop a well defined system using our Bluetooth controller.

The backbone of our application server will be the Java's dynamic class loading ability.  The server side applications will be resting in the container section of our application server. After a client is connected to the server it will choose the class that it wants to work with. At this moment application server dynamically loads the requested java class and maintains the necessary linkage with the client application and server application. After a connection dies the linkage is halted and the server application is killed.

It is also possible for different clients to work with the same class. In this case each client works with a different copy of the java class and handled independently.

Since the design of our application server is in its very early stages there is not much documentation prepared for it yet.

# 6. Server API

Server API is designed to be a middleware between server application and the application server. It is used to define the working conditions of the server applications and tell the application server how to run the server application. It offers the necessary tools to maintain the consistency and the security of the designed system.

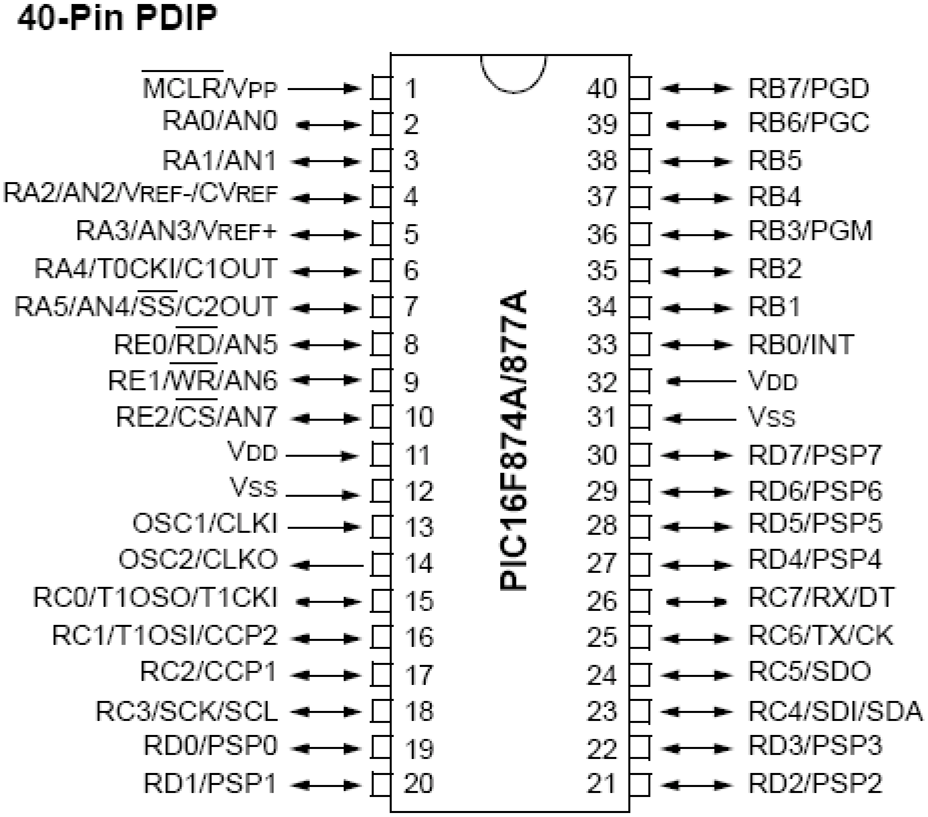
Functions of the Server API and their explanations are given below:

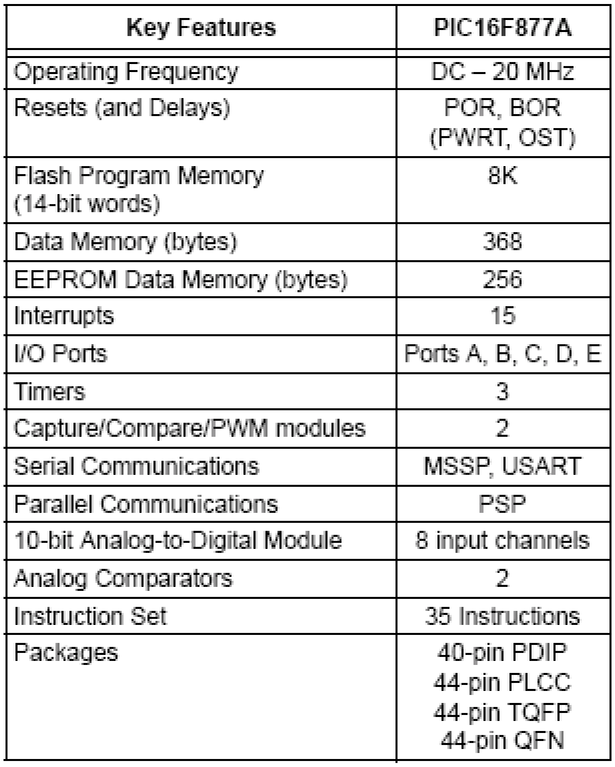
| **FUNCTIONS** | **EXPLANATION** |
|---|---|
| int setMaxIdleTime(int time) | Sets the max allowed idle time for the server application. If no interrupt comes from client for the defined amount of time, the application is killed. |
| int activateLogging (string file_name) | Keep the log of transferred pin values between server and client. Data will be written to a file with the given name. |
| int deactivateLogging (void) | Deactivate logging. |
| int setMaxClient(int number) | Set the number of maximum clients that can use that class at a time. |
| int activateSecurity(void) | Activates the restrictions on the class to increase security. |
| int deactivateSecurity(void) | Deactivate security mode. |
| Int setAllowedDevices(string* deviceList) | Sets the allowed device identities that can use the class. No other client is allowed to work with that class. |

# 7. Hardware Design

## 7.1 PIC

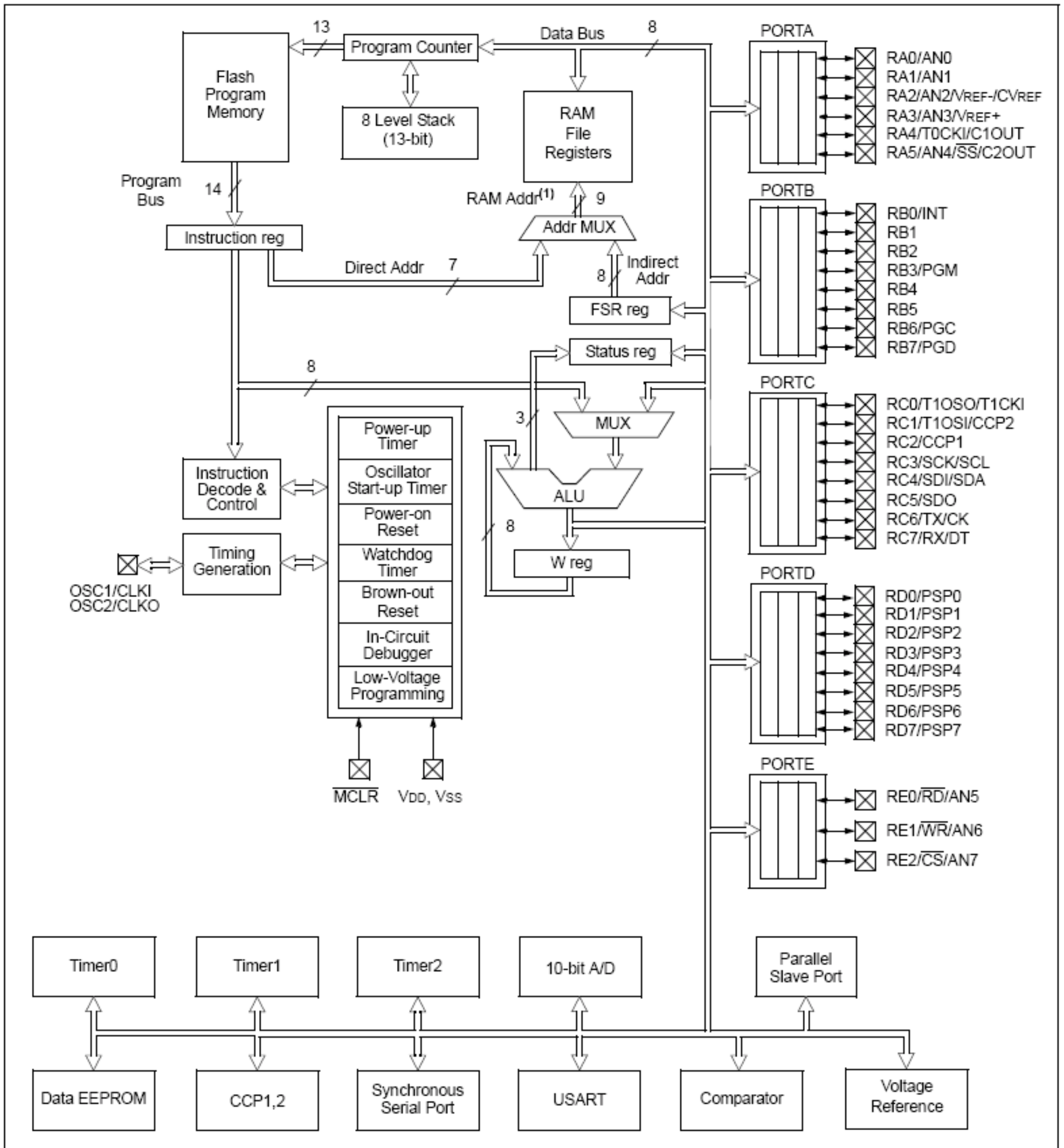
Since we need a client application on the client side and since the client will not be a computer, the application should be embedded on the client. We decided that the most suitable solution for an embedded microcontroller was a PIC. A long survey led us to choose PIC16F877A PDIP, which satisfies our needs with its 40 pins, 8K x 14 words of Flash Program Memory, 368 x 8 bytes of Data Memory (RAM), 256 x 8 bytes of EEPROM Data Memory and embedded analog-to-digital converter.





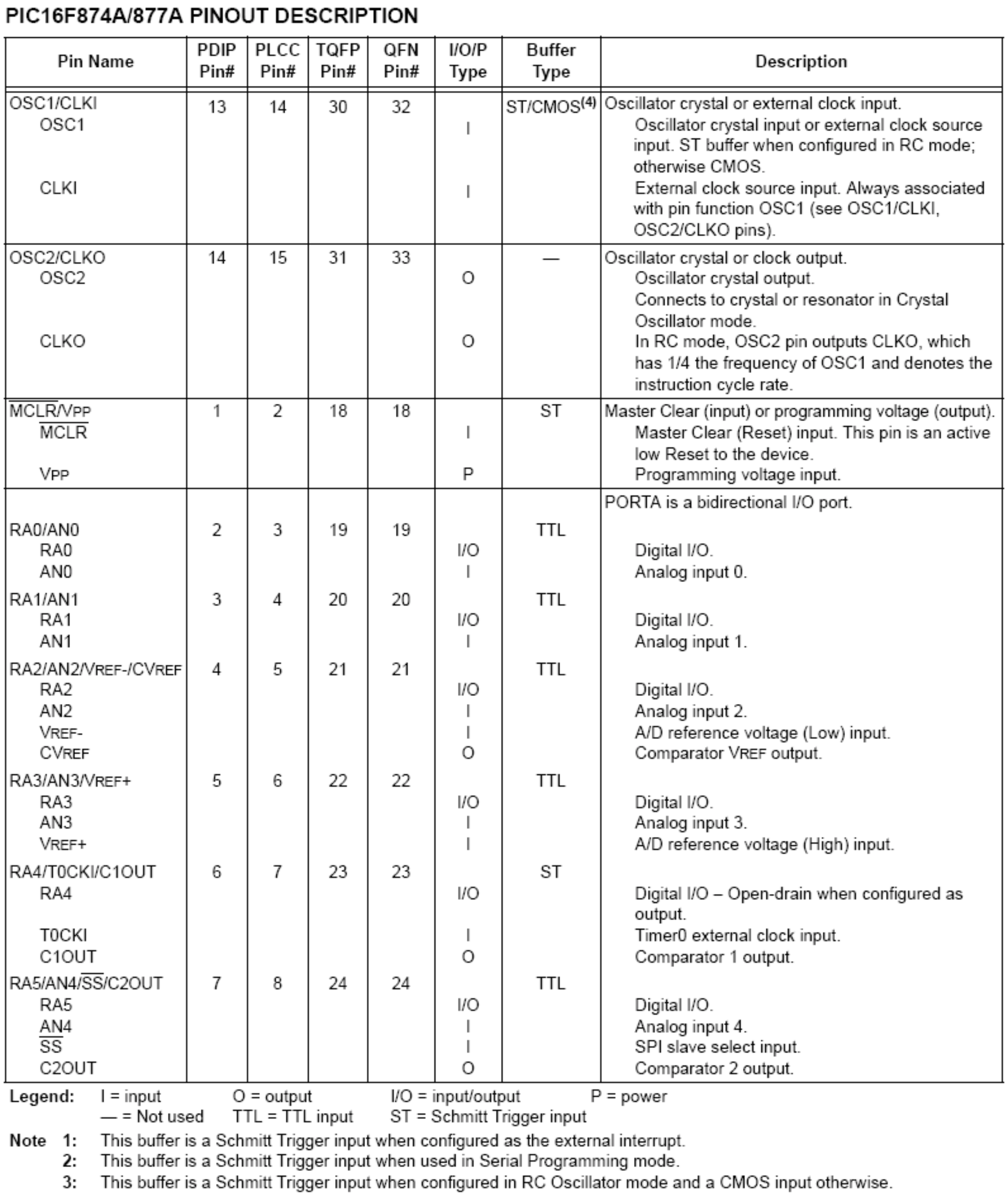
| Key Features | PIC16F877A |
|---|---|
| Operating Frequency | DC – 20 MHz |
| Resets (and Delays) | POR, BOR<br>(PWRT, OST) |
| Flash Program Memory<br>(14-bit words) | 8K |
| Data Memory (bytes) | 368 |
| EEPROM Data Memory (bytes) | 256 |
| Interrupts | 15 |
| I/O Ports | Ports A, B, C, D, E |
| Timers | 3 |
| Capture/Compare/PWM modules | 2 |
| Serial Communications | MSSP, USART |
| Parallel Communications | PSP |
| 10-bit Analog-to-Digital Module | 8 input channels |
| Analog Comparators | 2 |
| Instruction Set | 35 Instructions |
| Packages | 40-pin PDIP<br>44-pin PLCC<br>44-pin TQFP<br>44-pin QFN |

## PIC16F874A/877A BLOCK DIAGRAM



| Device | Program Flash | Data Memory | Data EEPROM |
|---|---|---|---|
| PIC16F874A | 4K words | 192 Bytes | 128 Bytes |
| PIC16F877A | 8K words | 368 Bytes | 256 Bytes |

**Note 1:** Higher order bits are from the Status register.

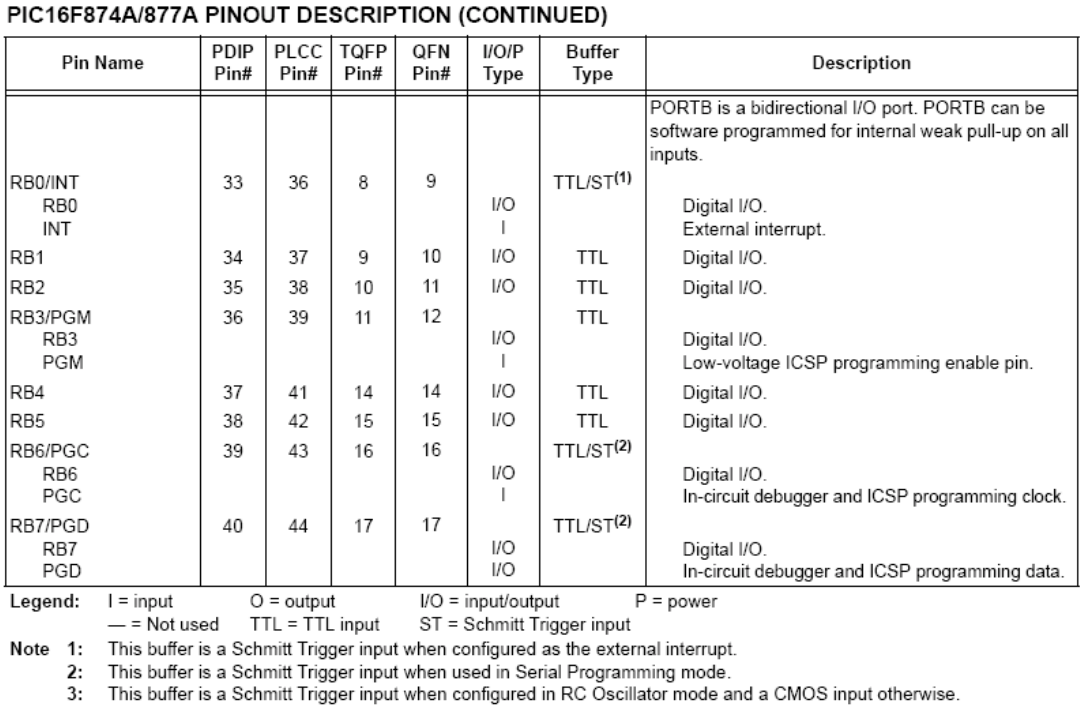


**PIC16F874A/877A PINOUT DESCRIPTION**

| Pin Name | PDIP Pin# | PLCC Pin# | TQFP Pin# | QFN Pin# | I/O/P Type | Buffer Type | Description |
|---|---|---|---|---|---|---|---|
| OSC1/CLKI | 13 | 14 | 30 | 32 | | ST/CMOS[4] | Oscillator crystal or external clock input. |
| OSC1 | | | | | I | | Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS. |
| CLKI | | | | | I | | External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins). |
| OSC2/CLKO | 14 | 15 | 31 | 33 | | — | Oscillator crystal or clock output. |
| OSC2 | | | | | O | | Oscillator crystal output.<br>Connects to crystal or resonator in Crystal Oscillator mode. |
| CLKO | | | | | O | | In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate. |
| $\overline{\text{MCLR}}$/VPP | 1 | 2 | 18 | 18 | | ST | Master Clear (input) or programming voltage (output). |
| $\overline{\text{MCLR}}$ | | | | | I | | Master Clear (Reset) input. This pin is an active low Reset to the device. |
| VPP | | | | | P | | Programming voltage input. |
| | | | | | | | PORTA is a bidirectional I/O port. |
| RA0/AN0 | 2 | 3 | 19 | 19 | | TTL | |
| RA0 | | | | | I/O | | Digital I/O. |
| AN0 | | | | | I | | Analog input 0. |
| RA1/AN1 | 3 | 4 | 20 | 20 | | TTL | |
| RA1 | | | | | I/O | | Digital I/O. |
| AN1 | | | | | I | | Analog input 1. |
| RA2/AN2/VREF-/CVREF | 4 | 5 | 21 | 21 | | TTL | |
| RA2 | | | | | I/O | | Digital I/O. |
| AN2 | | | | | I | | Analog input 2. |
| VREF- | | | | | I | | A/D reference voltage (Low) input. |
| CVREF | | | | | O | | Comparator VREF output. |
| RA3/AN3/VREF+ | 5 | 6 | 22 | 22 | | TTL | |
| RA3 | | | | | I/O | | Digital I/O. |
| AN3 | | | | | I | | Analog input 3. |
| VREF+ | | | | | I | | A/D reference voltage (High) input. |
| RA4/T0CKI/C1OUT | 6 | 7 | 23 | 23 | | ST | |
| RA4 | | | | | I/O | | Digital I/O – Open-drain when configured as output. |
| T0CKI | | | | | I | | Timer0 external clock input. |
| C1OUT | | | | | O | | Comparator 1 output. |
| RA5/AN4/$\overline{\text{SS}}$/C2OUT | 7 | 8 | 24 | 24 | | TTL | |
| RA5 | | | | | I/O | | Digital I/O. |
| AN4 | | | | | I | | Analog input 4. |
| $\overline{\text{SS}}$ | | | | | I | | SPI slave select input. |
| C2OUT | | | | | O | | Comparator 2 output. |

**Legend:** I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

**Note 1:** This buffer is a Schmitt Trigger input when configured as the external interrupt.
**2:** This buffer is a Schmitt Trigger input when used in Serial Programming mode.
**3:** This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

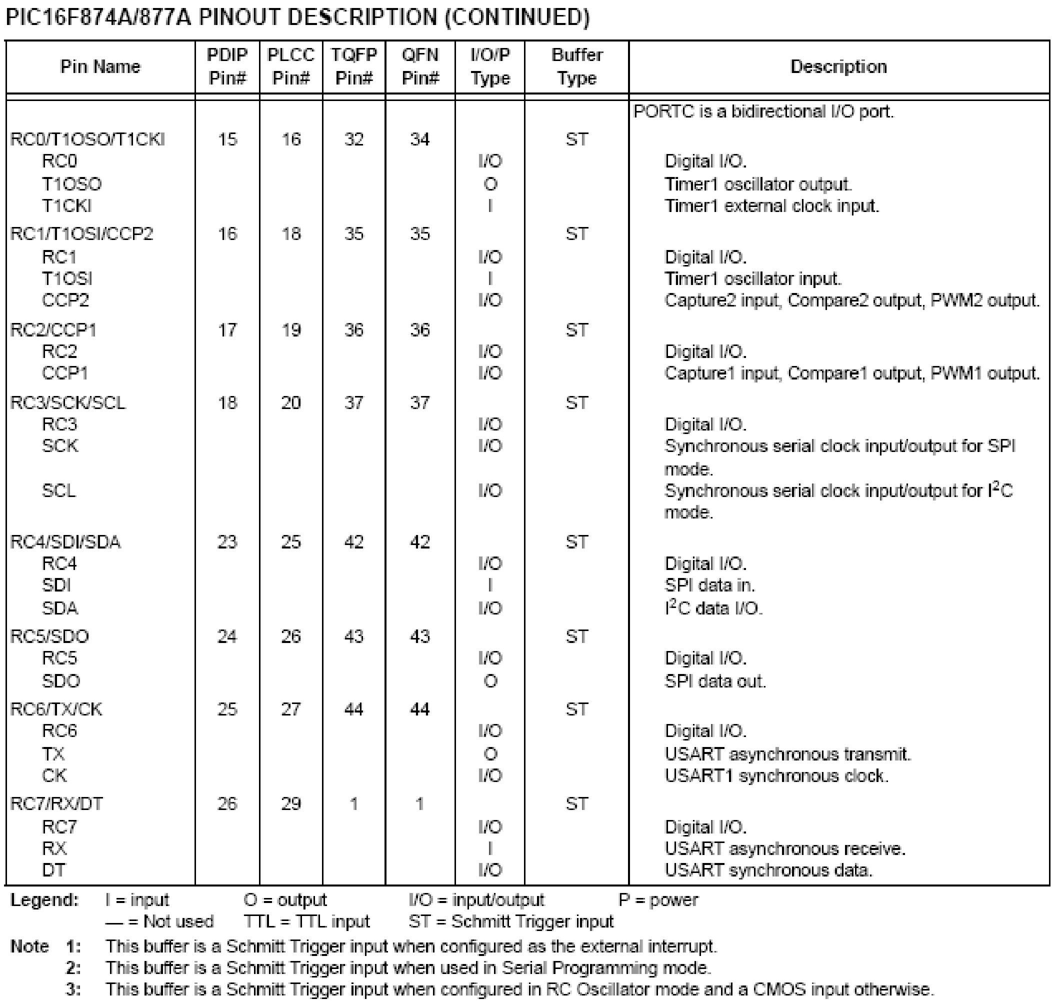


**PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)**

| Pin Name | PDIP Pin# | PLCC Pin# | TQFP Pin# | QFN Pin# | I/O/P Type | Buffer Type | Description |
|---|---|---|---|---|---|---|---|
| | | | | | | | PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs. |
| RB0/INT | 33 | 36 | 8 | 9 | | TTL/ST(1) | |
| RB0 | | | | | I/O | | Digital I/O. |
| INT | | | | | I | | External interrupt. |
| RB1 | 34 | 37 | 9 | 10 | I/O | TTL | Digital I/O. |
| RB2 | 35 | 38 | 10 | 11 | I/O | TTL | Digital I/O. |
| RB3/PGM | 36 | 39 | 11 | 12 | | TTL | |
| RB3 | | | | | I/O | | Digital I/O. |
| PGM | | | | | I | | Low-voltage ICSP programming enable pin. |
| RB4 | 37 | 41 | 14 | 14 | I/O | TTL | Digital I/O. |
| RB5 | 38 | 42 | 15 | 15 | I/O | TTL | Digital I/O. |
| RB6/PGC | 39 | 43 | 16 | 16 | | TTL/ST(2) | |
| RB6 | | | | | I/O | | Digital I/O. |
| PGC | | | | | I | | In-circuit debugger and ICSP programming clock. |
| RB7/PGD | 40 | 44 | 17 | 17 | | TTL/ST(2) | |
| RB7 | | | | | I/O | | Digital I/O. |
| PGD | | | | | I/O | | In-circuit debugger and ICSP programming data. |

**Legend:** I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

**Note 1:** This buffer is a Schmitt Trigger input when configured as the external interrupt.
**2:** This buffer is a Schmitt Trigger input when used in Serial Programming mode.
**3:** This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.



PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

| Pin Name | PDIP Pin# | PLCC Pin# | TQFP Pin# | QFN Pin# | I/O/P Type | Buffer Type | Description |
|---|---|---|---|---|---|---|---|
| | | | | | | | PORTC is a bidirectional I/O port. |
| RC0/T1OSO/T1CKI | 15 | 16 | 32 | 34 | | ST | |
| RC0 | | | | | I/O | | Digital I/O. |
| T1OSO | | | | | O | | Timer1 oscillator output. |
| T1CKI | | | | | I | | Timer1 external clock input. |
| RC1/T1OSI/CCP2 | 16 | 18 | 35 | 35 | | ST | |
| RC1 | | | | | I/O | | Digital I/O. |
| T1OSI | | | | | I | | Timer1 oscillator input. |
| CCP2 | | | | | I/O | | Capture2 input, Compare2 output, PWM2 output. |
| RC2/CCP1 | 17 | 19 | 36 | 36 | | ST | |
| RC2 | | | | | I/O | | Digital I/O. |
| CCP1 | | | | | I/O | | Capture1 input, Compare1 output, PWM1 output. |
| RC3/SCK/SCL | 18 | 20 | 37 | 37 | | ST | |
| RC3 | | | | | I/O | | Digital I/O. |
| SCK | | | | | I/O | | Synchronous serial clock input/output for SPI mode. |
| SCL | | | | | I/O | | Synchronous serial clock input/output for I$^2$C mode. |
| RC4/SDI/SDA | 23 | 25 | 42 | 42 | | ST | |
| RC4 | | | | | I/O | | Digital I/O. |
| SDI | | | | | I | | SPI data in. |
| SDA | | | | | I/O | | I$^2$C data I/O. |
| RC5/SDO | 24 | 26 | 43 | 43 | | ST | |
| RC5 | | | | | I/O | | Digital I/O. |
| SDO | | | | | O | | SPI data out. |
| RC6/TX/CK | 25 | 27 | 44 | 44 | | ST | |
| RC6 | | | | | I/O | | Digital I/O. |
| TX | | | | | O | | USART asynchronous transmit. |
| CK | | | | | I/O | | USART1 synchronous clock. |
| RC7/RX/DT | 26 | 29 | 1 | 1 | | ST | |
| RC7 | | | | | I/O | | Digital I/O. |
| RX | | | | | I | | USART asynchronous receive. |
| DT | | | | | I/O | | USART synchronous data. |

**Legend:** I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

**Note 1:** This buffer is a Schmitt Trigger input when configured as the external interrupt.
**2:** This buffer is a Schmitt Trigger input when used in Serial Programming mode.
**3:** This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

## PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

| Pin Name | PDIP Pin# | PLCC Pin# | TQFP Pin# | QFN Pin# | I/O/P Type | Buffer Type | Description |
|---|---|---|---|---|---|---|---|
| | | | | | | | PORTD is a bidirectional I/O port or Parallel Slave Port when interfacing to a microprocessor bus. |
| RD0/PSP0 | 19 | 21 | 38 | 38 | | ST/TTL$^{(3)}$ | |
| RD0 | | | | | I/O | | Digital I/O. |
| PSP0 | | | | | I/O | | Parallel Slave Port data. |
| RD1/PSP1 | 20 | 22 | 39 | 39 | | ST/TTL$^{(3)}$ | |
| RD1 | | | | | I/O | | Digital I/O. |
| PSP1 | | | | | I/O | | Parallel Slave Port data. |
| RD2/PSP2 | 21 | 23 | 40 | 40 | | ST/TTL$^{(3)}$ | |
| RD2 | | | | | I/O | | Digital I/O. |
| PSP2 | | | | | I/O | | Parallel Slave Port data. |
| RD3/PSP3 | 22 | 24 | 41 | 41 | | ST/TTL$^{(3)}$ | |
| RD3 | | | | | I/O | | Digital I/O. |
| PSP3 | | | | | I/O | | Parallel Slave Port data. |
| RD4/PSP4 | 27 | 30 | 2 | 2 | | ST/TTL$^{(3)}$ | |
| RD4 | | | | | I/O | | Digital I/O. |
| PSP4 | | | | | I/O | | Parallel Slave Port data. |
| RD5/PSP5 | 28 | 31 | 3 | 3 | | ST/TTL$^{(3)}$ | |
| RD5 | | | | | I/O | | Digital I/O. |
| PSP5 | | | | | I/O | | Parallel Slave Port data. |
| RD6/PSP6 | 29 | 32 | 4 | 4 | | ST/TTL$^{(3)}$ | |
| RD6 | | | | | I/O | | Digital I/O. |
| PSP6 | | | | | I/O | | Parallel Slave Port data. |
| RD7/PSP7 | 30 | 33 | 5 | 5 | | ST/TTL$^{(3)}$ | |
| RD7 | | | | | I/O | | Digital I/O. |
| PSP7 | | | | | I/O | | Parallel Slave Port data. |
| | | | | | | | PORTE is a bidirectional I/O port. |
| RE0/$\overline{\text{RD}}$/AN5 | 8 | 9 | 25 | 25 | | ST/TTL$^{(3)}$ | |
| RE0 | | | | | I/O | | Digital I/O. |
| $\overline{\text{RD}}$ | | | | | I | | Read control for Parallel Slave Port. |
| AN5 | | | | | I | | Analog input 5. |
| RE1/$\overline{\text{WR}}$/AN6 | 9 | 10 | 26 | 26 | | ST/TTL$^{(3)}$ | |
| RE1 | | | | | I/O | | Digital I/O. |
| $\overline{\text{WR}}$ | | | | | I | | Write control for Parallel Slave Port. |
| AN6 | | | | | I | | Analog input 6. |
| RE2/$\overline{\text{CS}}$/AN7 | 10 | 11 | 27 | 27 | | ST/TTL$^{(3)}$ | |
| RE2 | | | | | I/O | | Digital I/O. |
| $\overline{\text{CS}}$ | | | | | I | | Chip select control for Parallel Slave Port. |
| AN7 | | | | | I | | Analog input 7. |
| VSS | 12, 31 | 13, 34 | 6, 29 | 6, 30, 31 | P | — | Ground reference for logic and I/O pins. |
| VDD | 11, 32 | 12, 35 | 7, 28 | 7, 8, 28, 29 | P | — | Positive supply for logic and I/O pins. |
| NC | — | 1, 17, 28, 40 | 12,13, 33, 34 | 13 | — | — | These pins are not internally connected. These pins should be left unconnected. |

**Legend:** I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

**Note 1:** This buffer is a Schmitt Trigger input when configured as the external interrupt.
**2:** This buffer is a Schmitt Trigger input when used in Serial Programming mode.
**3:** This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

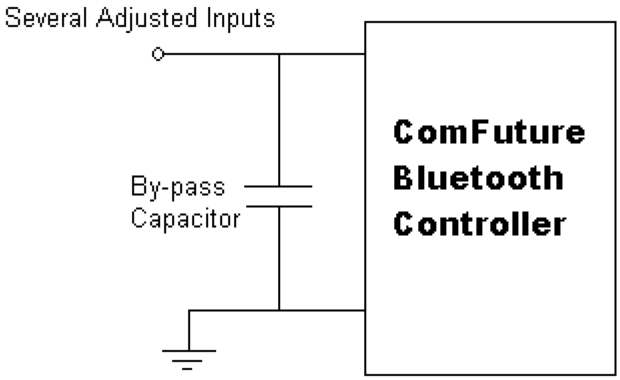
## 7.2 Programming Software

As a result of our researches and discussions after considering various software we decided on using MPLAB v6.42 as our IDE. As it is a comprehensive editor, project manager and design desktop for application development of embedded designs using

Microchip PICmicro microcontrollers. In addition, a few of the many MPLAB IDE system features are presented to help finish applications quickly.

## 7.3 Intermediate Hardware

We are going to use custom parts separately purchased to use as noise filters and voltage adjusters. Also we will need digital-to-analog converter and several logic chips to format the output and a simple logic selection circuit to direct dynamically selected outputs.



The above circuit model illustrates the By-pass capacitor that we will use as a primitive noise filter.

# 8. Proof of Concept

In order to prove that our general purpose bluetooth controller works properly for typical industrial devices, we will implement a higher level library for a specific device such as a remote controlled car, a robot or an air conditioner. The client application will run on the device, receiving orders from the application server and giving feedbacks to the server. By the use of this device, we will prove that any device can be automated using our controller. Although the proof of concept part is not a part of our project, we will make use of it in the presentation of our product and provide the classes we added as a sample for the future users.

For instance, in case of controlling a car, the car can travel in a room without hitting the walls, directed by a computer and no human interaction needed at all. Furthermore, various sensors can be placed on top of the car for a wide variety of purposes, this way the car can be used for many different purposes like spraying insecticides, other chemical pesticides that threatens health.

It is also possible to use a robot which has already added sensors on it and define some actions for it as a combination of a server and client applications. This approach is also promising as a testing and evaluation tool.

# 9. Conclusion

We choosed our intended tools and built our guidelines in this report. We made throughout research about our hardware and project requirements and chose our hardware which was a great experience in our project.

This initial design report is prepared to establish a connection between our design and implementation. The information given here such as diagrams and other design products are produced in order to guide us through our way in the implementation of our project. Despite being an initial design, this document is a milestone that will
help us make our prototype and real design report. We believe that this report will contribute to our project in a quite useful way.

# 10. References

(1) Dynamic Class Loading in java
http://www-h.eng.cam.ac.uk/help/tpl/languages/java/javaplugins.html

(2) Dynamically Extend Java Applications
http://www.javaworld.com/javaworld/jw-08-2001/jw-0810-extend.html

(3) PIC16F877A PDIP modeled PIC documentation.

(4) MPLAB IDE documentation.

(5) Java OBEX API documentation.