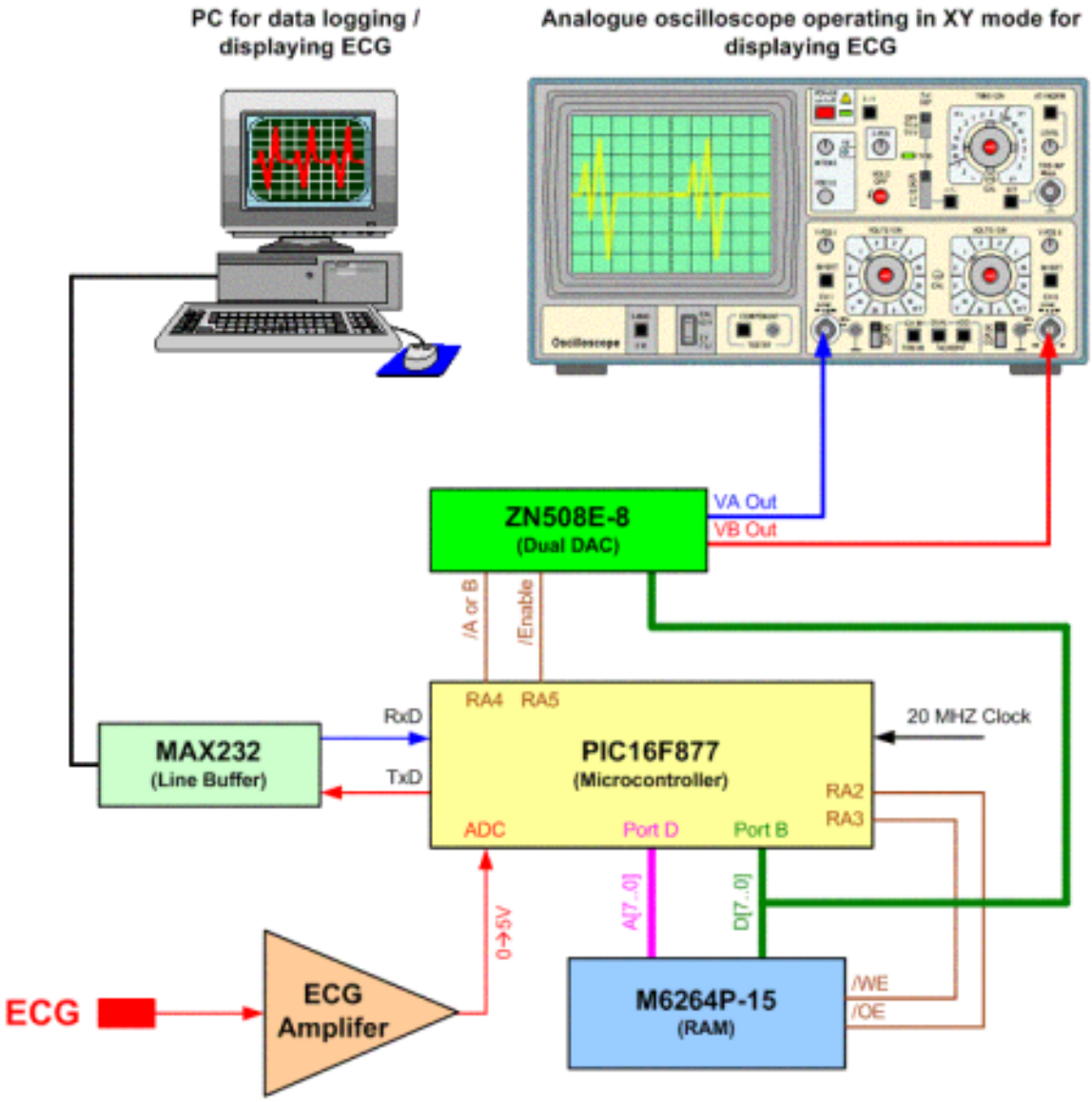
MiniProject

Real-Time Microcontroller Based ECG Monitor

Report B: Electronic Aspects



UNIVERSITY of ULSTER
school of electrical and mechanical engineering

COURSE: BEng (HONS) Electronic Systems
MODULE: EEE512J2 - Electronic Product Design
BY: Colin K McCord
DATE: Thursday, 17 July 2003

TABLE OF CONTENTS

1.0 INTRODUCTION

The heart's strong pumping action is driven by powerful waves of electrical activity in which the muscle fibres contract and relax in an orchestrated sequence. These waves cause weak currents to flow in the body, changing the relative electric potential between different points on the skin. An electrocardiogram is a biophysical instrumentation device that is used to view/record the electrical activity of the heart for various diagnostic purposes.

The electrocardiogram (or ECG) has been used extensively in medicine sine its invention in the early 1900's, and has since proven to be invaluable in various diagnostic applications, such as the detection of irregular heartbeat patterns (i.e. fibrillation or arrhythmia), heart murmurs (or other abnormal heart sounds), tissue/structural damage (such as valve malfunction) and coronary artery blockage. Other applications of the ECG are very effective in areas of sports medicine, or sports therapy, in tracking the heartbeat through various levels of physical activity to assist the patient in attaining a desired, optimum heart rate.

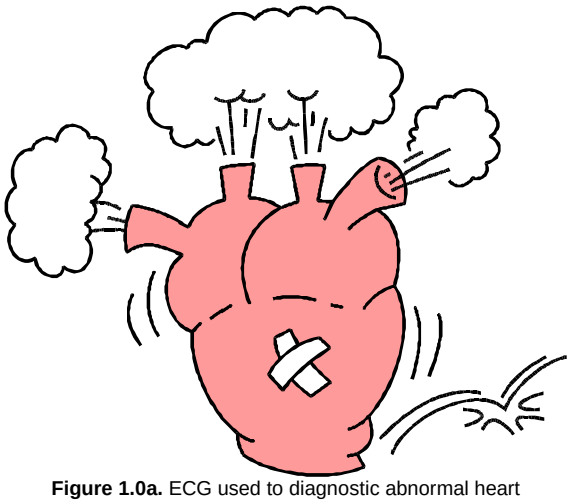


Figure 1.0a. ECG used to diagnostic abnormal heart

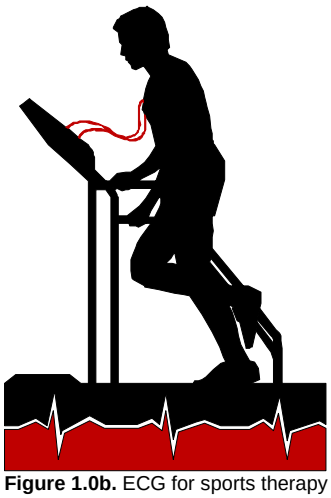


Figure 1.0b. ECG for sports therapy

Therefore, while the concept of an ECG is not a novel one, the attraction of this project lay in the challenge to build a simple, compact, operational medical device at a low cost. The basic design theory is as follows: -

- The electrical activity of the heart is detected using electrodes placed on the surface of the chest cavity. These electrodes act as bio-transducers to convert the signal from its existing form in the body (ionic) into electrical current in the wires.

- The generated signal is put through an amplifier to allow for observations, measurements, and recordings to be made. This stage is extremely important, as the cardiac signal is *very small*, i.e. on the order of mili-volts, thus a large amplification is necessary for any use to be made of the signal.

- The amplified signal is then sent to the PIC for Analogue-to-Digital conversion, signal manipulation, calculation of beats per minute (displayed using 3, 7-segment LED displays), data logging (RS232 communications) and analogue signal output (DAC) for a visual display of the ECG. Note that an oscilloscope can be used to provide a visual output.

This report will detail the development and implementation of a low-cost microcontroller base ECG monitor.

2.0. FUNDAMENTALS

2.1. Electrocardiography (ECG, EKG)

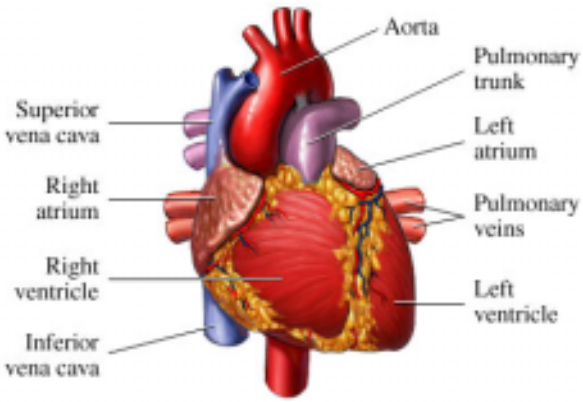


Figure 2.1a. Heart Anatomy [W1]

The heart is a muscular pump made up of four chambers. The two upper chambers are called atria, and the two lower chambers are called ventricles.

The purpose of the atria is to act as 'filling chambers' for the ventricles; the right side of the heart is the pulmonary pump, i.e. it pumps blood between the heart and the lungs, and the left side of the heart is the systemic pump, i.e. it pumps blood between the heart and the entire body.

The heart beats as a result of 'commands' passed in the form of bioelectric impulses and action potentials. These action potentials result in a series of rapid and successive patterns of depolarization and re-polarization across the cardiac muscle, generating an electric signal. The electrical activity of the heart can be detected through the skin by small metal discs called electrodes. The electrodes are attached to the skin on the chest, arms, and legs.

The cardiac cycle begins at the Sino-Atria node, located in the right atrium at the superior cava. The beginning of the cycle corresponds to the contraction of the atria. Following this is a 100ms delay until the activation of the Atria-ventricular node. This delay is important because it allows time for the ventricles to fill, increasing the efficiency of the heart. The signal is then propagated down the ventricular septum resulting in ventricular contraction. The signal generated over one period of the cardiac cycle is depicted in figure 2.1b (P-Wave: Atria Depolarization, QRS-Complex: Ventricular Depolarization, T-Wave: Ventricular Re-polarization).

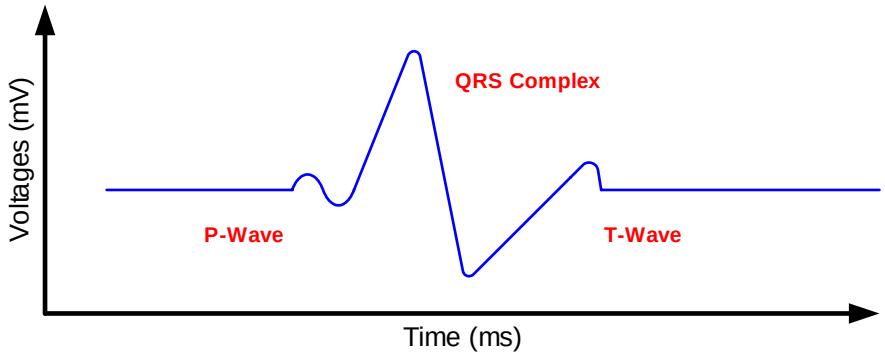


Figure 2.1b. One period of the cardiac cycle

Note that the signal generated from the heart is extremely small (about 2mV's in amplitude), and at a very low frequency, having a bandwidth of about 150Hz.

The heart can be considered as an electric dipole, repetitively changing both in magnitude and direction as it goes through the cardiac cycle. The magnitude of the dipole will be at a maximum during ventricular contraction. This is important note, as it is quite likely that the smaller P and T waves will be lost in the effects of noise. Therefore the theory behind detecting the cardiac signal is to place electrodes on the surface of the body, and simply measure the different differences in potential that arise as the dipole moves through its cycle.

The measured differences in potential are referred to as 'leads'. Note that it is always a difference in potential between at least two electrodes that is being measured, as there is no absolute zero reference voltage in the

body, only a dipole changing in both space and time. According to cardiac theory, in order to detect the strongest difference in potential (the peak signal); the optimum electrode placement is to have one on the right shoulder, and one on the left hip. This is what is usually referred to as “Lead II”, a convention that arises from the work of Willem Einthoven, a pioneer in ECG development, who observed the differences in signal strength as he took measurements between two electrodes with placement on the left shoulder, the right shoulder and the left hip (Einthoven’s Triangle). See reference [W3] for an overview of the different standard electrode placements.

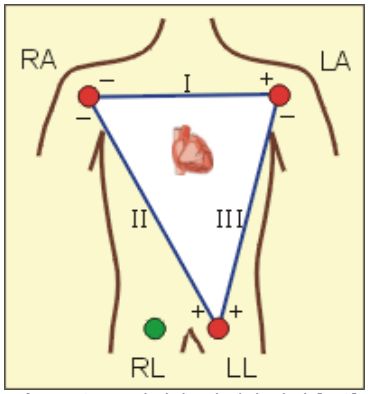


Figure 2.1c. Limb leads (Bipolar) [W3]

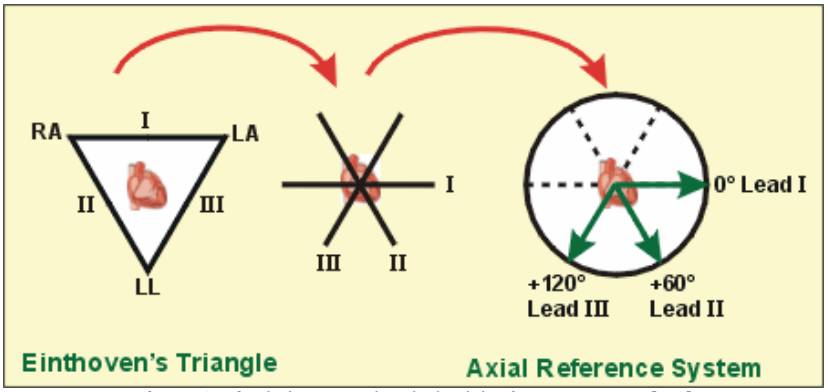


Figure 2.1d. Einthoven's triangle / axial reference system [W3]

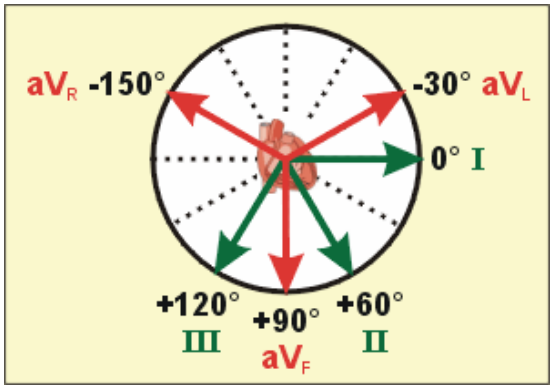


Figure 2.1e. Augmented limb leads (Unipolar) [W3]

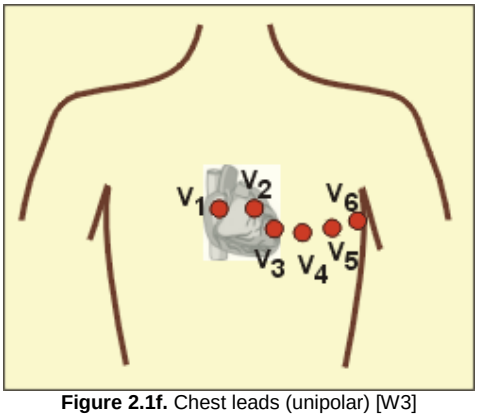


Figure 2.1f. Chest leads (unipolar) [W3]

The electrocardiogram (ECG) is a simple, non-invasive technique for detecting abnormalities and diagnosing heart defects, merely by noting the presence of irregularities in the PQRST waveform. For example an electrocardiogram may show: -

- Signs of insufficient blood flow to the heart.
- Signs of a new or previous injury to the heart (heart attack).
- Evidence of heart enlargement.
- Heart rhythm problems (arrhythmias).
- Signs of inflammation of the sac surrounding the heart.
- Changes in the electrical activity of the heart caused by a chemical (electrolyte) imbalance in the body.

Note: Electrocardiography cannot predict whether a person will have a heart attack.

2.2. Electrodes

The role of the electrodes is to act as bio-electric transducers at the interface between the body and the ECG. Inside the body, electricity exists in the form of ions. Thus, the purpose of the electrodes is to convert electricity from its ionic form in the body into an electric current in the wires.

Ag-AgCl electrodes are the current standard for use in medical applications related to biophysical instrumentation and measurements. The gel provides impedance matching at the interface between the electrode and the surface of the skin, which means that noise effects are reduced, increasing the signal-to-noise ratio, allowing for a clear signal to be detected. They are non-polarisable, meaning that the differences in potential that are measured do not depend on current variations in the wires. They are stable, easy to use, and inexpensive. See reference [W5] for detailed information on the Ag-AgCl electrode.

Figure 2.2b. 3M red dot electrodes cost $14.94 from [W7]

Figure 2.2c. 3M red dot resting electrodes cost $9.94 from [W7]

2.3. ECG Amplifier

The heart's strong pumping action is driven by powerful waves of electrical activity in which the muscle fibres contract and relax in an orchestrated sequence. These waves cause weak currents to flow in the body, changing the relative electric potential between different points on the skin by about 1mV. The signals can change sharply in as little as one fiftieth of a second. So boosting this signal to an easily measured one-volt level requires an amplifier with a gain of about 1,000 and a frequency response of at least 50 hertz.

At first it appears that an operational amplifier could be used. But two vexing subtleties make most op-amps unsuitable. First, when two electrodes are placed at widely separated locations on the skin, the epidermis acts like a crude battery, generating a continuously shifting potential difference that can exceed 2V. The cardiac signal is small in comparison. Second, the body and the wires in the device make good radio antennas, which readily pick up the 50Hz hum that emanates from every power cable connected to the mains supply. This adds a sinusoidal voltage that further swamps the tiny pulse from the heart and because these oscillations lie so close to the frequency range needed to rack the heart's action, this unwanted signal is difficult to filter out.

Both problems generate equal swells of voltage at the amplifier's two inputs. Unfortunately, op-amps usually can't reject these signals. To ensure that this "common-mode" garbage (whose amplitude, can be over 1,000 time greater than the cardiac signal) adds no more than a 1 percent error, a CMRR (Common-Mode Rejection Ratio) of at least 100,000 to one (100 decibels) is required. This precision eludes most op-amps.

When an application calls for both high gain and a CMRR of 80 dB or greater special devices known as "instrumentation amplifiers" are required. The AD624AD from Analog Devices (see [W12]) when set to a gain of 1,000 has a CMRR exceeding 110 dB. It is available from Farnell (order code 102-076) for £22.50. Clearly at bit expensive, hence another option is the AD620AN available from Farnell (order code 527-567) for £6.14.

Figure 2.3a shows a simple ECG amplifier using the AD624AD instrumentation amplifier. A gain of 1,000 is selected by shorting certain pins together as shown. The two-stage RC filter weeds out frequencies higher

than about 50 hertz. A 3 lead cable connects the circuit to the electrodes and two wires are required to connect the output to an ADC for sampling.

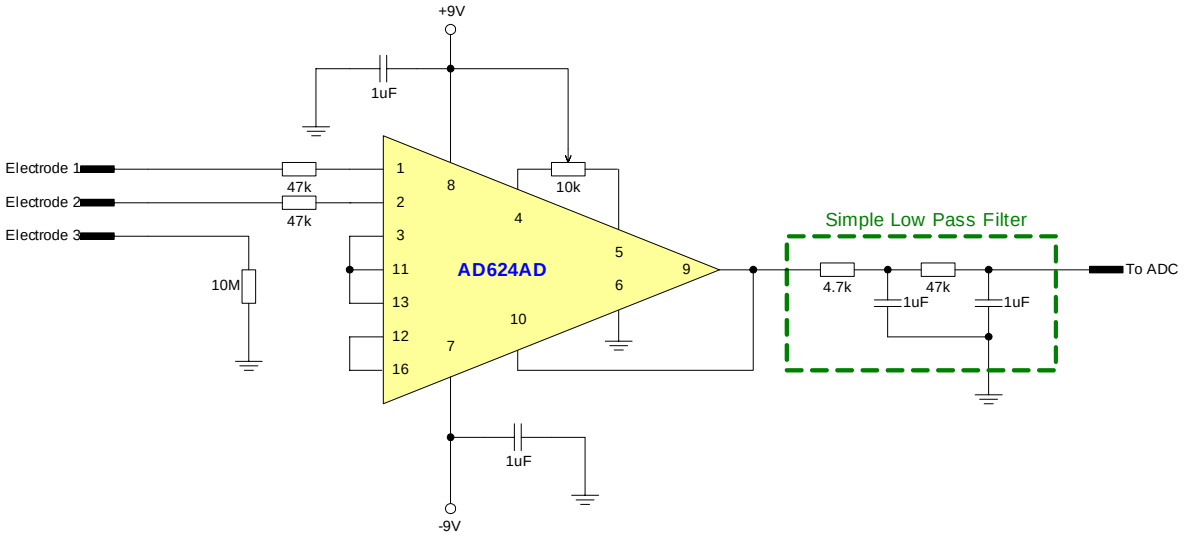


Figure 2.3a. Simple ECG Amplifier

2.4. The Cathode Ray Tube (CRT)

"The CRT is a glass bulb which has had the air removed and then been sealed with a vacuum inside. At the front is a flat glass screen which is coated inside with a phosphor material. This phosphor will glow when struck by the fast moving electronics and produce light, emitted from the front and forming the spot and hence the trace. The rear of the CRT contains the electron 'gun' assembly. A small heater element is contained within a cylinder of metal called the cathode. When the heater is activated by applying a voltage across it, the cathode temperature rises and it then emits a stream of electrons." [B2].

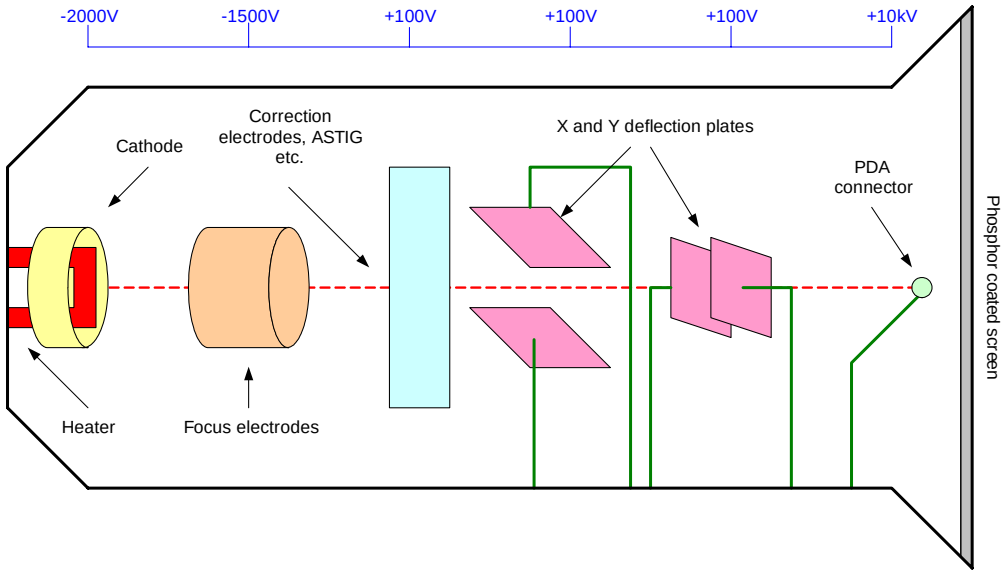


Figure 2.4a. Diagram of a typical Cathode-ray tube (CRT) construction.

A traditional analogue oscilloscope / analogue ECG machine draws its trace with a spot of light (produced by a deflectable beam of electrons) moving across the screen of its CRT (see Figure 2.4b). Basically an oscilloscope / ECG consists of the CRT, a 'time base' circuit to move the spot steadily from left to right across the screen at the appropriate time and speed, and some means (usually a 'Y' deflection amplifier) of enabling the signal to deflect the spot in the vertical or Y direction.

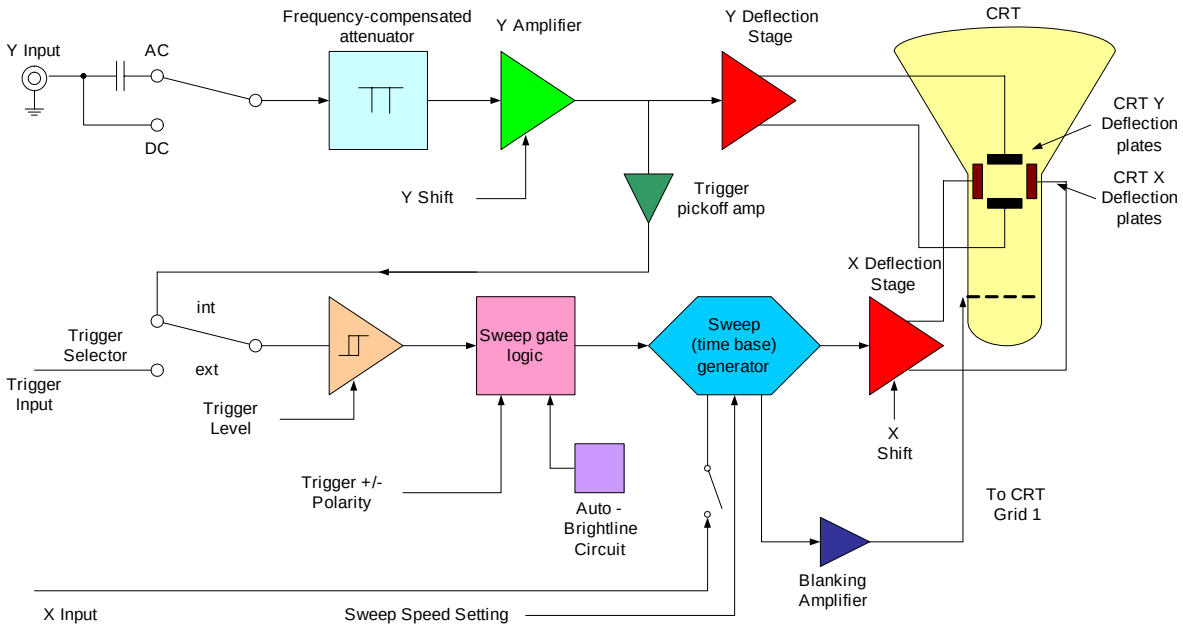


Figure 2.4b. Block diagram of a basic CRT oscilloscope; similar to a traditional analogue ECG display

2.5. Digital Sampling

Digital sampling requires an ADC (analogue-to-digital converter) to converter analogue voltages to binary representation. The sampling rate specifies the number of samples taken per second. Figure 2.5a demonstrates clearly how an analogue waveform is digitally sampled and displayed onto the screen (LCD, Computer Monitor, or a CRT using a DAC etc…).

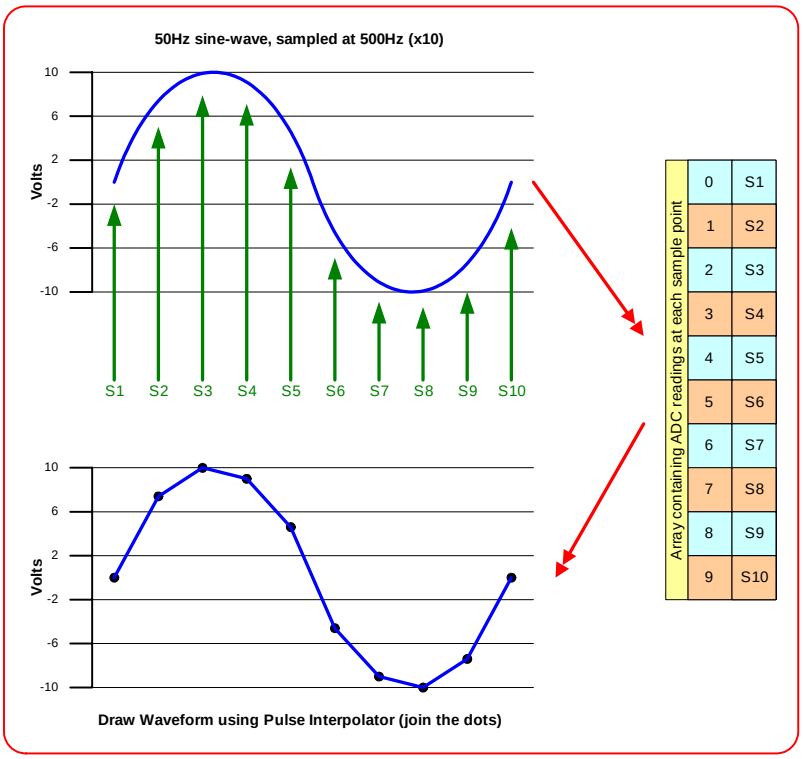


Figure 2.5a. Example showing how a sine-wave is digitally sampled

2.6. Aliasing

Aliasing is an undesirable effect that can occur when digital sampling analogue voltages. This is the display of an apparent signal which does not actually exist, usually caused by under-sampling.

Many samples should be taken per cycle (Nyquist theorem states that *"to define a sine wave, a sampling system must take more than two samples per cycle".*) to ensure an accurate representation of an analogue signal in a digital memory. If only one sample is taken per cycle, or one sample per several cycles, then aliasing occurs. For example say a waveform is being sampled every three cycles, these samples may form together, particularly when using pulse interpolation (join the dots), to look like a valid waveform.

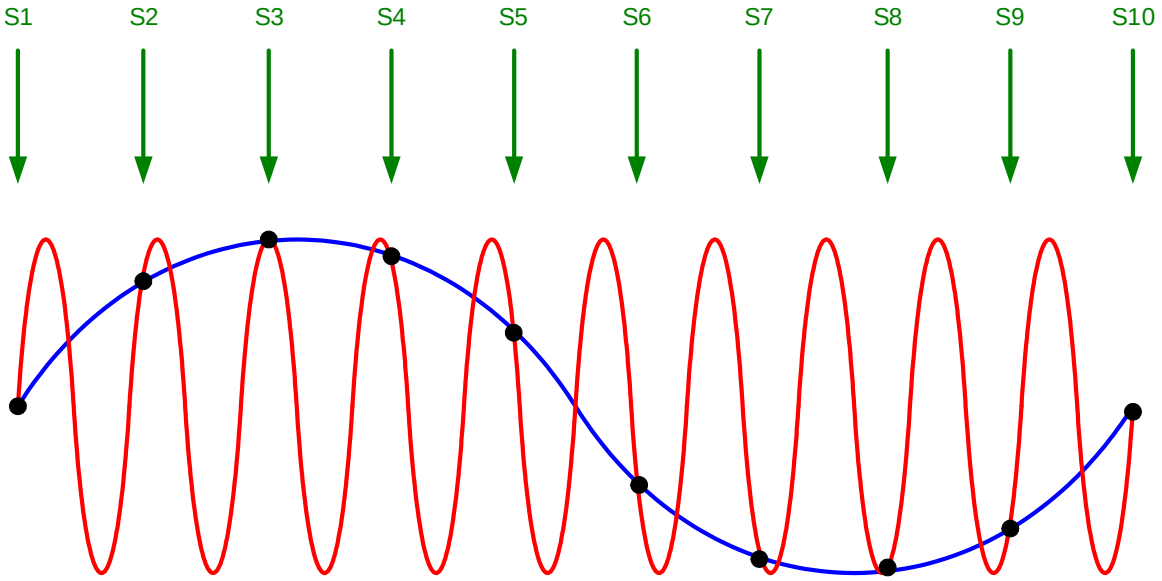


Figure 2.6a. Demonstrating aliasing, red is the real waveform, while blue is an alias.

Figure 2.6a clearly demonstrates how false signals (aliasing) are created. The red waveform is the real waveform, notice that the waveform is under sampled (see green arrows for sample points). The black dots shows were the real waveform (red) has been sampled, by joining the dots, it is clear that a perfect sine-wave is created (blue), which is an alias of the original signal. Note that it is impossible to tell that the blue signal is an alias.

There is nothing that can be done after sampling to correct aliasing; hence the solution is to filter out high frequencies by sending the input signal through a low-pass filter. Ideally all frequencies above half the sample rate should be filtered out.

2.7. The PIC Microcontroller

A PIC (**P**eripheral **I**nterface **C**ontroller) microcontroller is an IC manufactured by Microchip.

These ICs are complete computers in a single package. The only external components necessary are whatever is required by the I/O devices that are connected to the PIC.

The traditional Von-Neumann Architecture (Used in: 80X86, 8051, 6800, 68000, etc…) is illustrated in Figure 2.7a. Data and program memory share the same memory and must be the same width.

"All the elements of the von Neumann computer are wired together with the one common data highway or bus. With the CPU acting as the master controller, all information flow is back and forward along these shared wires. Although this is efficient, it does mean that only one thing can happen at any time. This phenomenon is sometimes known as the von Neumann bottleneck." [B3]

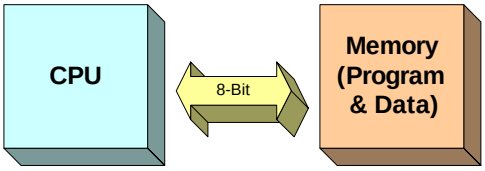


Figure 2.7a. Simplified illustration of the von Neumann architecture

PICs use the Harvard architecture. The Harvard architecture (Figure 2.7b) is an adaptation of the standard von Neumann structure with separate program and data memory: data memory is made up by a small number of 8-bit registers and program memory is 12 to 16-bits wide EPROM, FLASH or ROM.

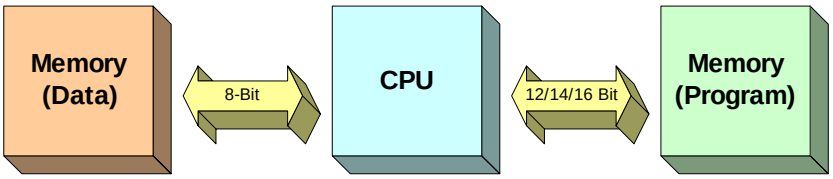


Figure 2.7b. Simplified illustration of the Harvard architecture

Traditional CISC (**C**omplex **I**nstruction **S**et **C**omputer) machines (Used in: 80X86, 8051, 6800, 68000, etc…) have many instructions (usually > 100), many addressing modes and it usually takes more than 1 internal clock cycle to execute. PIC microcontrollers are RISC (**R**educed **I**nstruction **S**et **C**omputer) machines, which have 33 (12-bit) to 58 (15-bit) instructions, reduced addressing modes (PICs have only direct and indirect), each instruction does less, but usually executes in one internal clock.

"The combination of single-word instructions, the simplified instruction decoder implicit with the RISC paradigm and the Harvard separate program and data buses gives a fast, efficient and cost effective processor implementation." [B3]

2.7.1. Summary of the PICs Built-in Peripherals

SPI (**S**erial **P**eripheral **I**nterface) uses 3 wires (data in, data out, clock), Master/Slave (can have multiple masters), very high speed (1.6 Mbps), and full speed simultaneous send and receive (full duplex).

I^2C (**I**nter **IC**) uses 2 wires (data and clock), Master/Slave. There are lots of cheap I^2C chips available; typically < 100kbps.

UART (**U**niversal **A**synchronous **R**eceiver/**T**ransmitter) with baud rates of 300bps to 115kbps, 8 or 9 bits, parity, start and stop bits, etc. Outputs 5V hence an RS232 level converter (e.g. MAX232) is required.

Timers, both 8 and 16 bits, many have prescalers and some have postscalers. In 14 bit cores they generate interrupts. External pins (clock in/clock out) can be used for counting events.

Ports have two control registers: TRIS sets whether each pin is an input or an output and PORT sets their output bit levels. Note: Other peripherals may steal pins, so in this respect peripheral registers control ports as well. Most pints have 25mA source/sink (LED enabled), but not all pins, it is important to look up the datasheet. Floating input pints must be tied off (or set to outputs).

ADCs (**A**nalogue to **D**igital **C**onverter) are currently slow, less than 54 KHz sampling rate (8, 10 or 12 bits), theoretically higher accuracy when PIC is in sleep mode (less digital noise) once the sample is complete the ADC sends an interrupt waking the PIC. Note that the PIC must wait until the sampling capacitor is charged; see datasheets.

2.8. RS232 Serial Interface

RS232 is simple, universal, well understood and supported, but it has some serious shortcomings as a data interface. Its origins predate modern computers and it contains many features that are not relevant to the modern user. It can control very old primitive modems and has many control signals to do this in hardware, but often it is used without these old control and status lines.

Its major feature is that it does not require the transmission of a clock, the reception of a 'start bit' is enough to cause the receiver to time all its actions from this one edge. This is called asynchronous transmission. RS232 allows a 5% difference in transmitted timings and receiver chip timings. This is important if using a PIC as the datasheet specifies the % error of the baud rate generator at certain baud rates (the higher the baud rate, the higher the % error), as long as this error is less than 5% the RS232 standard is capable of coping.

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumours for its early demise, has remained widely used.

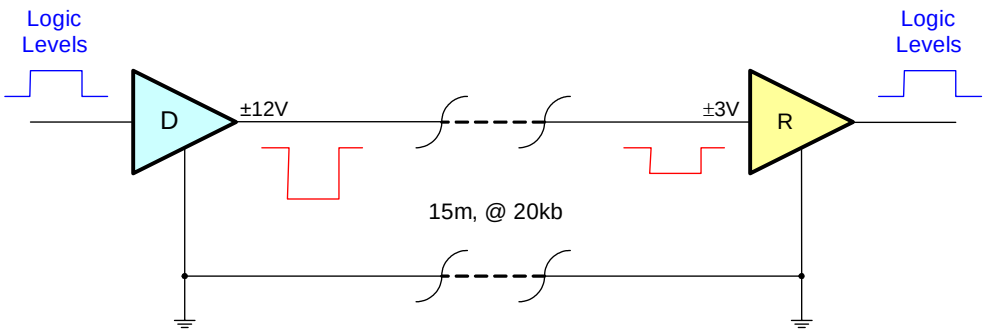


Figure 2.8a. Illustration of RS232, 1 driver and 1 receiver

"Both RS232 and RS423 are unbalanced (or single-ended) standards, where the receiver measures the potential between signal line and ground reference. Even though the transmitter and receiver grounds are usually connected though the transmission line return, the impedance over a long distance may support a significant difference in the two ground potentials, which will degrade noise immunity. Furthermore, any noise induced from the outside will affect signal lines differently from the ground return due to their dissimilar electrical characteristics – hence the name unbalanced." [B3]

RS232 data is bi-polar, e.g. a +3 to +12 volt indicates an SPACE (ON) while a -3 to -12 volt indicates an MARK (OFF). Modern computer equipment ignores the negative level and accepts a zero voltage level as the MARK (OFF) state. This means circuits powered by 5 VDC are capable of driving RS232 circuits directly; however, the overall range that the RS232 signal may be transmitted/received is dramatically reduced.

The output signal level usually swings between +12V and -12V. The 'dead area' between +3v and -3v is designed to absorb line noise. This dead area can vary for various RS232 like definitions, for example the definition for V.10 has a noise margin from +0.3V to -0.3V. Many receivers designed for RS232 are sensitive to differentials of 1v or less.

Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Receive Data	7	Request to Send
3	Transmit Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figure 2.8b. 9-pin RS232 D-connector, pin signal description

Typical line drivers / receivers chips for RS232 are the Maxim MAX232 or MAX233 chips (see http://www.maxim-ic.com) the original specification states that RS232 should drive 50 feet, but modern line driver/receivers can manage much better than this.

Baud Rate	Max Distance Shielded Cable	Max Distance Unshielded Cable
110 bps	5000 feet	3000 feet
300 bps	5000 feet	3000 feet
1200 bps	3000 feet	3000 feet
2400 bps	1000 feet	500 feet
4800 bps	1000 feet	250 feet
9600 bps	250 feet	250 feet

Figure 2.8c. Typical maximum distance modern line driver/receivers can manage before errors occur.

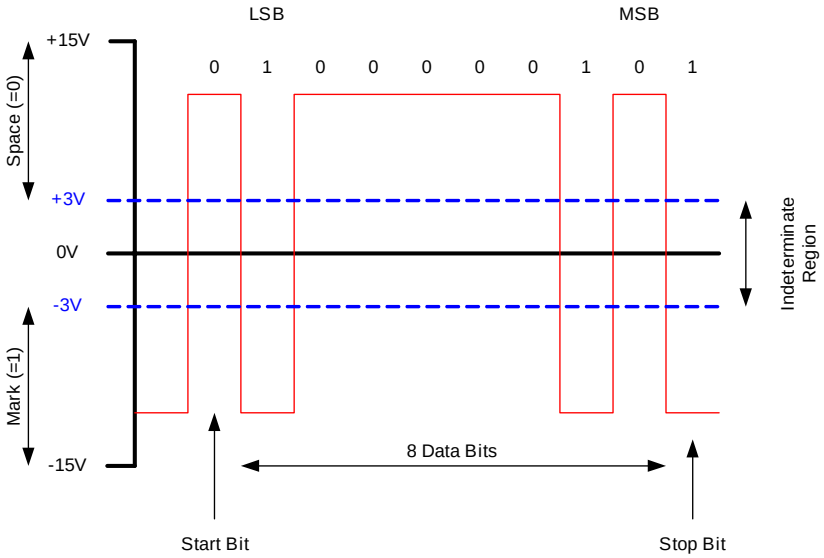


Figure 2.8d. Illustration of how data is transmitted over RS232

3.0. THE PIC16F877 MICROCONTROLLER

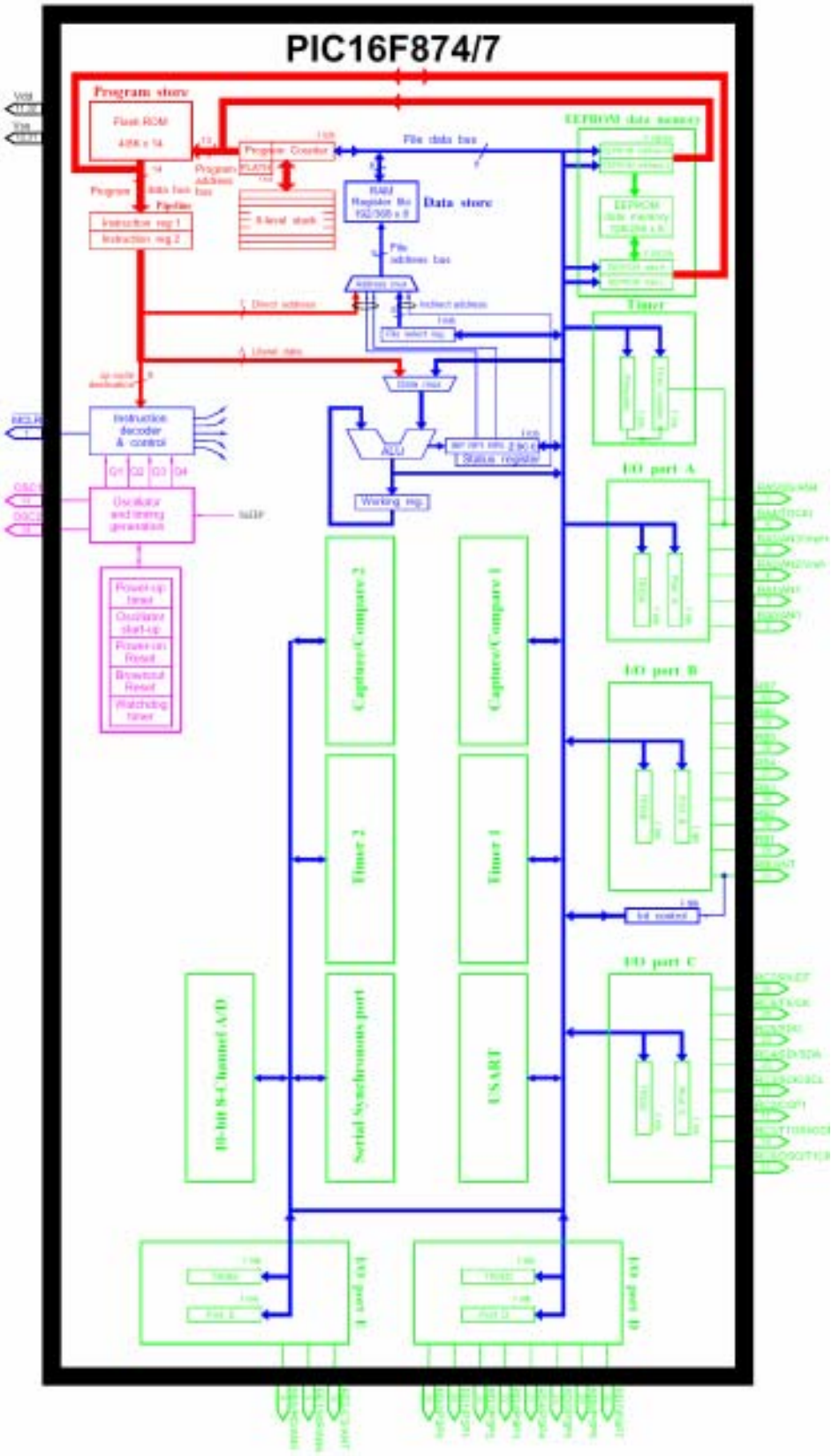


Figure 3.0a: Architecture of the PIC16F877 microcontroller [W10]

"The PIC16F877 is a high-performance FLASH microcontroller that provides engineers with the highest design flexibility possible. In addition to 8192x14 words of FLASH program memory, 256 data memory bytes, and 368 bytes of user RAM, PIC16F877 also features an integrated 8-channel 10-bit Analogue-to-Digital converter. Peripherals include two 8-bit timers, one 16-bit timer, a Watchdog timer, Brown-Out-Reset (BOR), In-Circuit-Serial Programming™, RS-485 type UART for multi-drop data acquisition applications, and I2C™ or SPI™ communications capability for peripheral expansion. Precision timing interfaces are accommodated through two CCP modules and two PWM modules." [W7]

3.1. Overview of the File Registers

The data memory is partitioned into multiple banks which contain the general purpose registers and the special function registers. Bits RP1 and RP0 are the bank select bits, these bits are found in the STATUS register (b6 & b5).

00h	Indirect addr. (*)
01h	TMR0
02h	PCL
03h	STATUS
04h	FSR
05h	PORTA
06h	PORTB
07h	PORTC
08h	PORTD
09h	PORTE
0Ah	PCLATH
0Bh	INTCON
0Ch	PIR1
0Dh	PIR2
0Eh	TMR1L
0Fh	TNR1H
10h	T1CON
11h	TMR2
12h	T2CON
13h	SSPBUF
14h	SSPCON
15h	CCPR1L
16h	CCPR1H
17h	CCP1CON
18h	RCSTA
19h	TXREG
1Ah	RCREG
1Bh	CCPR2L
1Ch	CCPR2H
1Dh	CCP2CON
1Eh	ADRESH
1Fh	ADCON0
20h 6Fh	General Purpose Register 80 Bytes
70h . . 7Fh	General Purpose Global Register 16 Bytes

Bank 0

80h	Indirect addr. (*)
81h	OPTION_REG
82h	PCL
83h	STATUS
84h	FSR
85h	TRISA
86h	TRISB
87h	TRISC
88h	TRISD
89h	TRISE
8Ah	PCLATH
8Bh	INTCON
8Ch	PIR1
8Dh	PIR2
8Eh	PCON
8Fh	Unimplemented
90h	Unimplemented
91h	SSPCON2
92h	PR2
93h	SSPADD
94h	SSPSTAT
95h	Unimplemented
96h	Unimplemented
97h	Unimplemented
98h	TXSTA
99h	SPBRG
9Ah	Unimplemented
9Bh	Unimplemented
9Ch	Unimplemented
9Dh	Unimplemented
9Eh	ADRESL
9Fh	ADCON1
A0h . . . . . EFh	General Purpose Register 80 Bytes
F0h . . FFh	Accesses Global Register 70h-7Fh

Bank 1

100h	Indirect addr. (*)
101h	TMR0
102h	PCL
103h	STATUS
104h	FSR
105h	Unimplemented
106h	PORTB
107h	Unimplemented
108h	Unimplemented
109h	Unimplemented
10Ah	PCLATH
10Bh	INTCON
10Ch	EEDATA
10Dh	EEADR
10Eh	EEDATH
10Fh	EEADRH
110h 16Fh	General Purpose Register 96 Bytes
170h . . 17Fh	Accesses Global Register 70h-7Fh

Bank 2

180h	Indirect addr. (*)
181h	OPTION_REG
182h	PCL
183h	STATUS
184h	FSR
185h	Unimplemented
186h	TRISB
187h	Unimplemented
188h	Unimplemented
189h	Unimplemented
18Ah	PCLATH
18Bh	INTCON
18Ch	EECON1
18Dh	EECON2
18Eh	RESERVED
18Fh	RESERVED
190h 1EFh	General Purpose Register 96 Bytes
1F0h . . 1FFh	Accesses Global Register 70h-7Fh

Bank 3

Figure 3.1a. PIC16F877 register file map

Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the special function registers (shown in yellow). Above the special function registers are general purpose registers (shown in blue), implemented as static RAM. All implemented banks contain special function registers. Some "high use" special function registers from one bank may be mirrored in another bank for code reduction and quicker access. Also notice that there are 16 general purpose global registers (shown in green), these registers can be accessed from any bank.

3.2. Overview of the 8-Channel 10-bit ADC

At first it appears that the PIC16F877 has 8 built-in ADCs, but this is not the case. Figure 3.2a shows a simplified block diagram of the analogue-to-digital converter module, clearly there is only one 10-bit ADC which can be connected to only one of eight input pins at any one time.

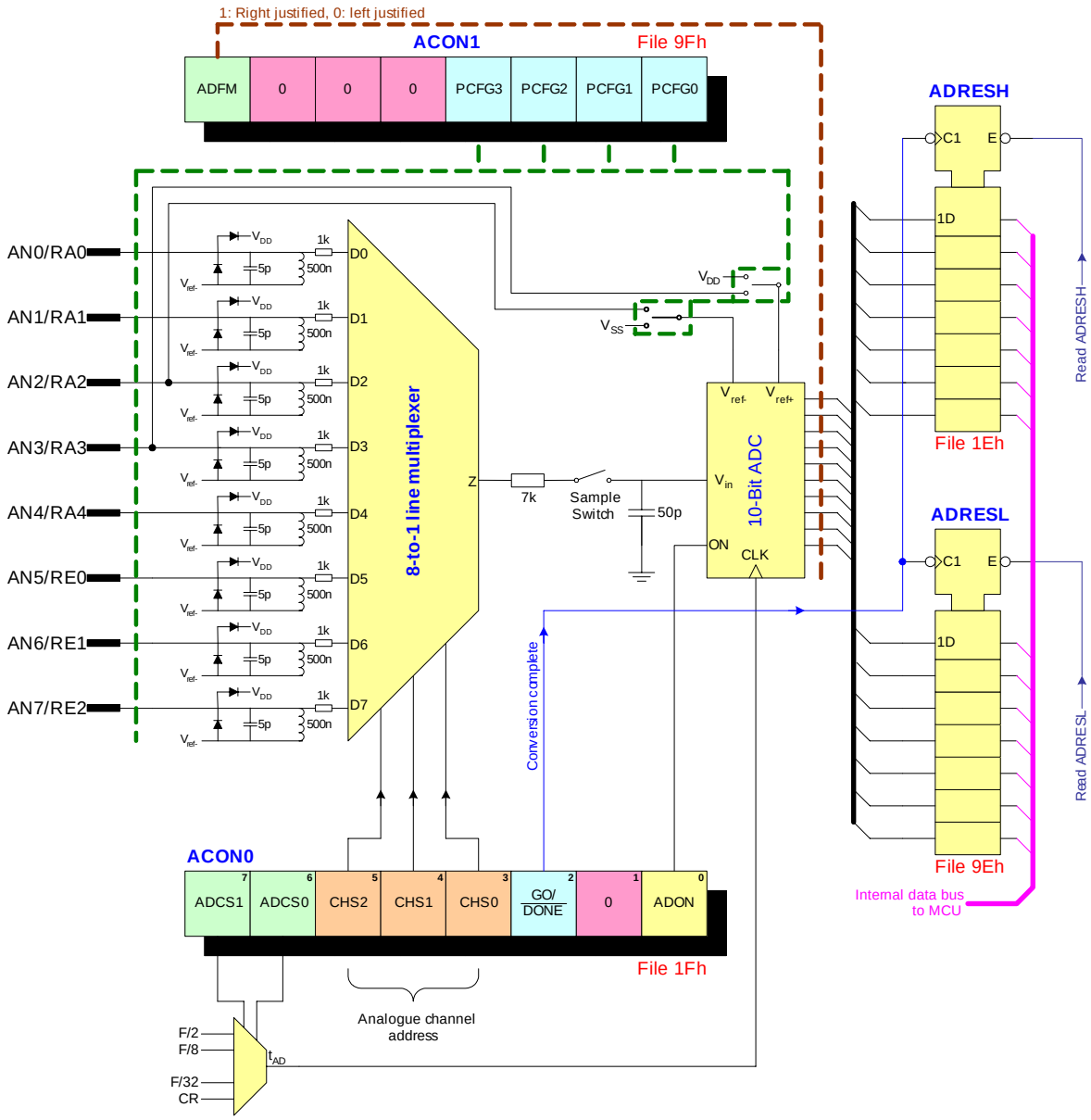


Figure 3.2a. Simplified block diagram of the PIC16F877 ADC module

The input analogue channels AN4..0 are shared with port A, and channels AN7..5 are shard with port E. If less than eight analogue channels are required then some of the pins can be assigned as digital I/O port lines using PCFG3..0 bits (see datasheet). For example, if PCFG3..0 = 0010 then AN4..0 are configured as analogue inputs, while AN7..5 are digital (port E free), with V_{DD} used as the reference.

"On reset all pins are set to accept analogue signals. Pins that are reconfigured as digital I/O should never be connected to an analogue signal. Such voltage may bias the digital input buffer into its linear range and the resulting large current could cause irreversible damage." [B3]

The 10-bit ADC uses a technique know as ‘successive approximation’, the following mechanical analogy will help explain how it works. Suppose there is an unknown weight, a balance scale and a set of precision known weights 1, 2, 4 and 8 grams. A systemic technique can be used to calculate the unknown weight.

Place the 8g weight on the pan and remove if it is too heavy. Next place the 4g weight on the pan and remove if it is too heavy. Next place the 2g weight on the pan and remove if it is too heavy. Next place the 1g weight on the pan and remove if it is too heavy. The sum of the weights still on the pan yields the nearest lower value of the unknown weight. This is illustrated in figures 3.2b to 3.2g.

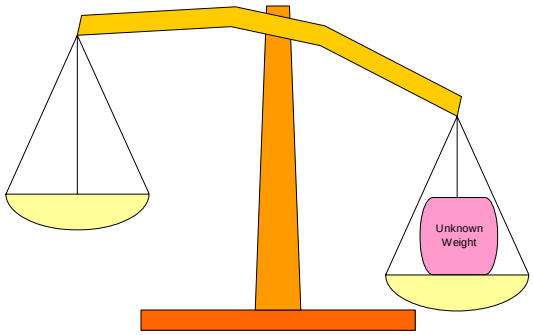


Figure 3.2b. Unknown weight placed on the scales

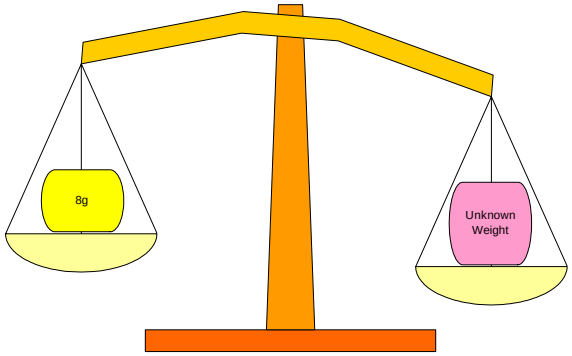


Figure 3.2c. 8g weight placed on the pan, not too heavy (keep)

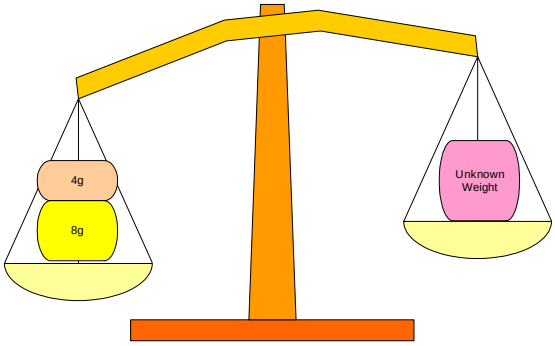


Figure 3.2d. 4g weight placed on the pan, too heavy (remove)

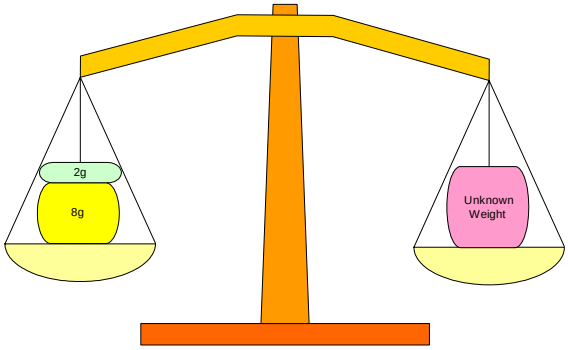


Figure 3.2e. 2g weight placed on the pan, not too heavy (keep)

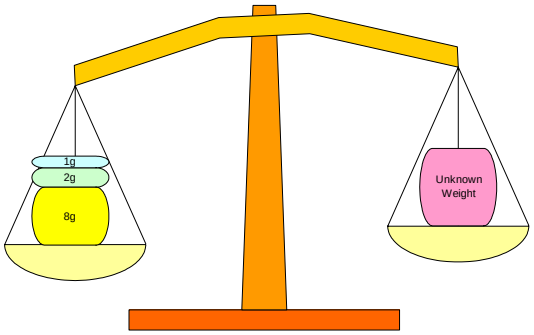


Figure 3.2f. 1g weight placed on the pan, too heavy (remove)

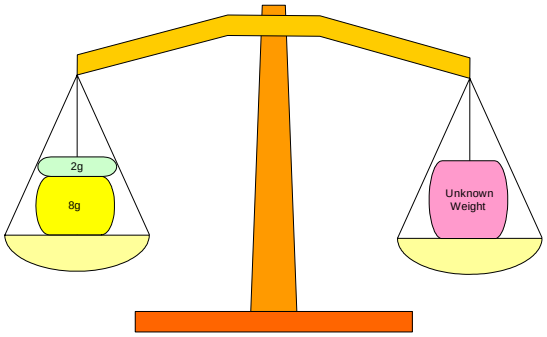


Figure 3.2g. Unknown weight is about 10g (1010)

The electronic equivalent to this successive approximation technique uses a network of precision capacitors configured to allow consecutive halving of a fixed voltage V_{REF} to be switched in to an analogue comparator, which acts as the balance scale.

Generally the network of capacitors are valued in powers of two to subdivide the analogue reference voltage (e.g. 1,2,4,8,16, etc…). This sampling acquisition process takes a finite time due to the charging time constant and is specified in the data sheet as 19.72µS.

3.3. Overview of the Hardware USART

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. The USART can be configured for asynchronous operation (UART) for communication with a PC or synchronous operation for communicating with peripheral devices such as DAC or DAC integrated circuit.

Note bit SPEN (RCSTA:7) and bits TRISC:7..6 have to be set in order to configure pin PC6/TX/CK and RC7/RX/DT for USART operation. CSS (C compiler) will automatically configure these bits, but it is important to be aware that if using fast_io(C) mode to manually configure port C, bits 7 & 6 must also be manually set if using the hardware UART.

3.3.1. Baud-Rate Generator, BRG

This is basically a programmable 8-bit counter followed by a switchable frequency $\div 4$ flip flop chain which can be set up to give the appropriate sampling and shifting rates for the desired baud rate, based on the PIC's crystal frequency XTAL (e.g. for 20MHz, XTAL = 20) giving: -

$$\text{Baud Rate (Low Speed Mode)} = \frac{XTAL}{64 \times (X+1)} \qquad \{3.3.1.1\}$$

$$\text{Baud Rate (High Speed Mode)} = \frac{XTAL}{16 \times (X+1)} \qquad \{3.3.1.2\}$$

$$X = \frac{XTAL \times 10^6}{64 \times BAUD} \qquad \{3.3.1.3\}$$

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks as this may reduce baud rate error in some cases.

4.0. HARDWARE DEVELOPMENT

The purpose of this project is to design, built and test a **low-cost** digital real-time ECG Monitor. The main reasoning behind hardware development was to keep the hardware cost to an absolute minimum.

It was decided to use the PIC16F877 (flash version), because it has enough RAM for time-compressed memory without the need for an external RAM chip. *"The PIC16F877 is a high-performance FLASH microcontroller that provides engineers with the highest design flexibility possible. In addition to 8192x14 words of FLASH program memory, 256 data memory bytes, and 368 bytes of user RAM, PIC16F877 also features an* integrated 8-channel 10-bit Analogue-to-Digital converter*. Peripherals include two 8-bit timers, one 16-bit timer, a Watchdog timer, Brown-Out-Reset (BOR), In-Circuit-Serial Programming™, RS-485 type UART for multi-drop data acquisition applications, and I2C™ or SPI™ communications capability for peripheral expansion. Precision timing interfaces are accommodated through two CCP modules and two PWM modules."* [W7].

The PICs ADC is used for data acquisition, a MAX232 buffer is used to convert the TTL serial logic of the PICs UART to the correct RS232 format (connect to a PC for data logging). A ZN508E-8 dual DAC (digital to analogue converter) is used to display the ECG on a CRT display (time-compressed memory). Additionally an analogue circuit is required to amplifier the ECG to a level that the PIC ADC can monitor.

4.1. Design Considerations

Accuracy, dependability, and precision are an absolute must if the device were to be used for diagnostic, or other medical purposes. Any small fluctuation in the waveform generated could carry critical diagnostic value, thus it is extremely important that the clinician can confidently and fully rely on the equipment. This means that the ECG must faithfully display the cardiac signal exactly as it exists in reality, such that any irregularity detected did in fact arise from an unhealthy cardiac cycle, *not* from the equipment that was used. Therefore, there were many special considerations that had to be taken into account when designing the ECG Monitor.

Noise:
First and foremost in these considerations is the effect of *noise*. Noise interference in the signal detection process would be detrimental to the experiment, as the ECG signal is at such small amplitudes it could easily be masked by noise related fluctuations. Therefore in order to detect the signal accurately, there must be strict limitations on the acceptable level of noise allowed, and every possible attempt must be made to minimise this level and reduce the effects of noise on the data acquisition process.

Signal Amplitude:
Another consideration that strongly influenced the design of the ECG is the fact that the cardiac signal generated has a very small peak amplitude. (As stated above, this is the very signal attribute that makes noise control so vital). Considerable amplification is necessary if there is any use to be made from the cardiac signal in terms of analysis and output. Also, the small size of the signal plays a very influential role in the approach to creating a system of visual output. Caution has to be taken to effectively differentiate between actual changes in the signal amplitude as opposed to a random variation in noise amplitude.

Low frequency:
Because the signal that is generated from the cardiac muscle has such a low bandwidth, it is very important that the ECG have a good low frequency response. This is because any shifts in the frequency of the detected signal, especially the S-T portion of the waveform, carry critical diagnostic value.

4.2. Simplified Block Diagram

Figure 4.2a, shows a simplified block diagram of the overall system. It is clear the PIC is connected to a MAX232 buffer which is connected to the PCs RS232 port. Notice that the clock is specified as 20MHz this was not the original plan, as a slower clock speed would reduce power consumption (critical if using battery power supply) and less noise would have been generate; hence ADC readings would be more accurate,

allow it is possible to put the PIC to sleep while taking the ADC reading, the ADC will send an interrupt waking the PIC once the acquisition is complete. This is not an option for this application because it takes a long time for the PICs oscillator to return to full speed, after a sleep operation; hence this means that the CRC display would not be refreshed fast enough to avoid flicker.

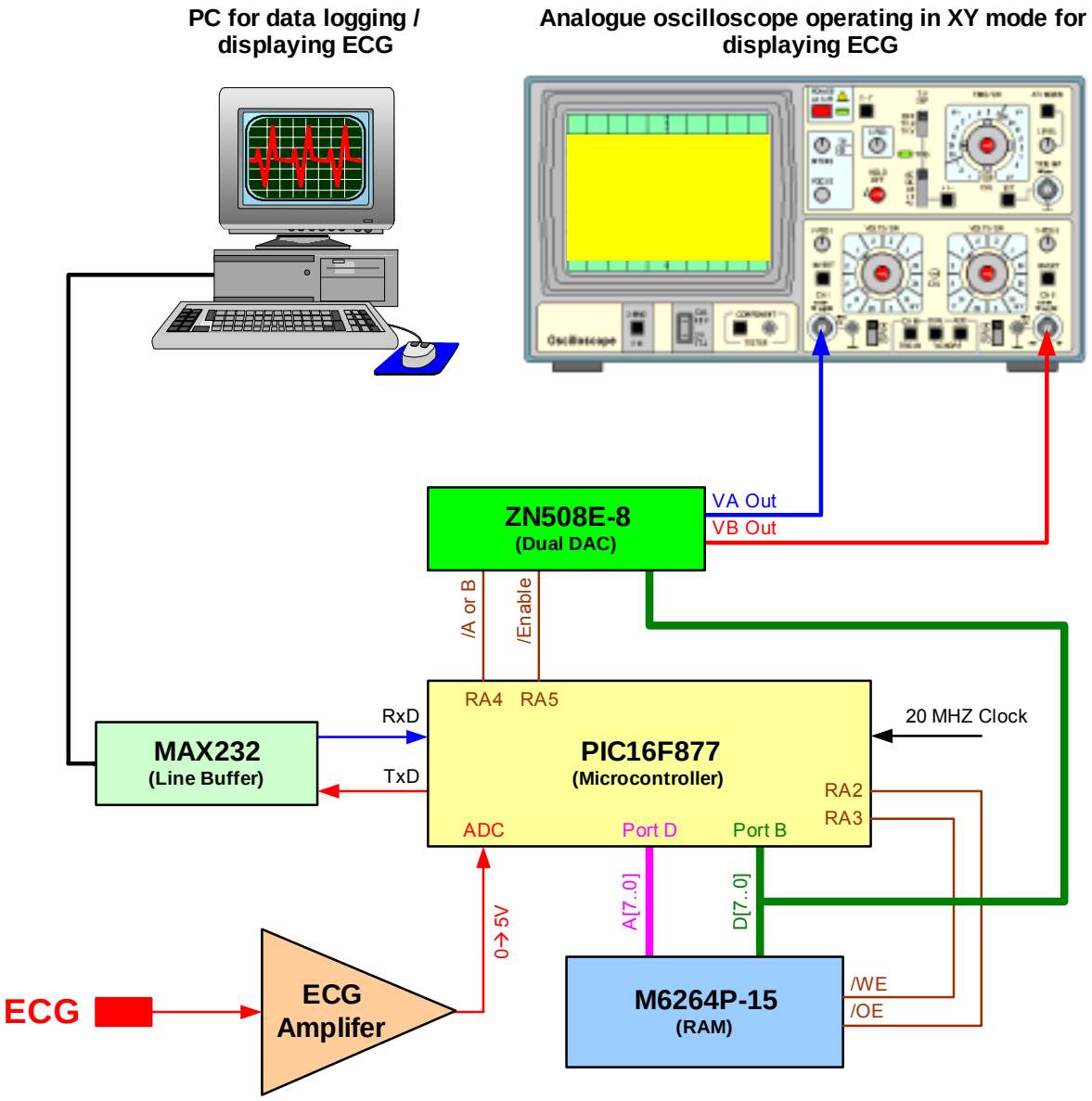


Figure 4.2a. Simplified block diagram of the system

The main reason for using a 20MHz clock was because the PIC program was developed in C, normally a C program will take 2-times longer to execute than if the program was written in assembly code. The chosen PIC is extremely powerful with a large program store (8K) and has no problem running C programs, making it ideally suited for the high-level programming language C. Microchip's 18 series (40MHz) are more powerful (design especially for C), these chips are much more expensive then the 16 series, hence were not a real option for this project.

The ECG signal (form electrodes place on the chest, arms and legs) is inputted into the ECG amplifier. The propose of the amplifier is to amplify the signal to a level that the PICs Analogue-to-Digital Converter (ADC) can sample (e.g. a common mode gain of 1000, hence the 1mV ECG signal is amplified to 1V).

Notice that the block diagram shows an external RAM chip, this chip is optional because the chosen PIC has more than enough internal RAM. The external RAM chip is for flexibility, as the time-compressed memory uses 75% of the PICs internal RAM and if complex software processing (automatic diagnostic) of the signal is required (sometime in the future) there may not be enough free internal RAM.

A dual digital-to-analogue converter (DAC) is used to convert digital data into analogue voltage. This chip is controlled by the PIC to manipulate the CRT trace (e.g. oscilloscope in XY mode), e.g. X (DAC A) and Y (DAC B) specifies the position of the spot, this spot moves faster than 50Hz across the screen hence it appears as a solid waveform to the human eye.

The MAX232 line buzzer converts PIC TTL logic (0V for logic 0, 5V for logic 1) into RS232 logic (12V for logic 0, -12V for logic 1). Serial communications is used to display ECG waveforms in real-time using a PC; the significant advantage of this is that ECG data can be streamed to disk (data logging) and/or transmitted in real-time through the internet (TCP/IP communications) for remote monitoring of ECG from anywhere in the world.

4.3. Digital Circuit Diagram

Figure 4.3a shows the digital circuit diagram, the design is simple with a low chip count (7 ICs).

S2	S1	S0	Baud Rate
0	0	0	115,200 bps
0	0	1	57,600 bps
0	1	0	38,400 bps
0	1	1	32,768 bps
1	0	0	19,200 bps
1	0	1	14,400 bps
1	1	0	9,600 bps
1	1	1	4,800 bps

Figure 4.3b. Dip-switch configuration

The dip-switches connected to port E are used to select RS232 baud rate (see figure 4.3b). The MAX232CPE (RS232 line buffer) is connected to pins RC7 and RC6 of the PIC; these pins are for use with the PICs hardware UART. The push button (connected to RA1) has three functions, if the user holds down the button during power up, it will put the ECG into test mode, e.g. 7-segment, RS232, RAM chip, DAC, ADC, will run through a simple diagnostics program. If the push button is pressed during normal operation, this will pause the ECG display, which resumes when the button is pressed again. The final operation of the push button is to mute the buzzer when a patient has “flat lined” (e.g. heart beat stopped).

Port D is used for an 8-bit address bus, the only component that requires an address is the RAM chip (M6264P-15). This RAM has 8k of memory, but because the address bus is only 8-bits wide it is only possible to access 256 bytes of memory (this is OK). If it is decided that more RAM is required at a later date, it is possible to use all 8k of RAM even though it appears that there are not enough free ports on the PIC. The trick is to use two 8-bit latches (as used for the 7-seg displays), were one latch sets the MSB of the address and the other sets the LSB of the address. Since both of these latches should not be enabled at the same time, the one free pin (RC3) can be used to select between the latches (e.g. use a not gate to one latch, and connect the pin directly to the other). But these increases chip count, therefore product cost, hence there would need to be a good reason of the usage of the entire RAM chip.

Port B is used for the shared data bus, the RAM (M6264P-15), the DAC (ZN508E-8), and the 8-bit latches for the 7-segment displays (74LS377) are all connected to this bus. Note that only one item can be enabled at any one time, for example if writing to the RAM chip all other devices on the data bus must be disabled. This is the classic microprocessor method of interfacing with devices, but it was not the only option, it is possible to use a serial RAM chip, serial DAC and serial latches, this would reduce the number of pins required on the PIC (no need for data bus or address bus) hence a 28-pin device could be used, reducing product cost. But the disadvantage of this is speed: time compressed memory is used to display the ECG on a CRT; it requires a refresh rate of at least 50Hz (flicker free), since 256 bytes of data is displayed on the screen, this means that the DAC must be updated 12,800 (50 x 256) times per second. If a serial DAC was used each bit is clocked in one at a time, hence a serial baud rate of 102,400kbps (12,800 x 8) would have been required, the PIC can achieve this, but there may not be enough processor power left to carry out other critical operations.

The reason why the MAX232CPE (RS232 line buffer) was chosen was because it can be powered from a single 5V power supply. Recall that RS232 requires +3 to +12 volts for a logic ‘0’ and -3 to -12 volts for a logic ‘1’, the MAX232CPE has a built-in (external capacitors required) voltage doubler circuit (+10V) and a voltage inverter circuit (-10V). This reduces product cost as the other option is to using a switch mode DC to

DC converter (cost about £5) to generate the required power supply. Allow a +9 and -9 supply (could use -10, +10) is required for the ECG amplifier circuit, hence a DC to DC converter is required anyway. Allow the max232 data sheet states that the +10 and -10 voltage pins could be used to drive other circuits, but this is not recommended, plus its good design practice to keep analogue and digital circuits separate (problems with noise).

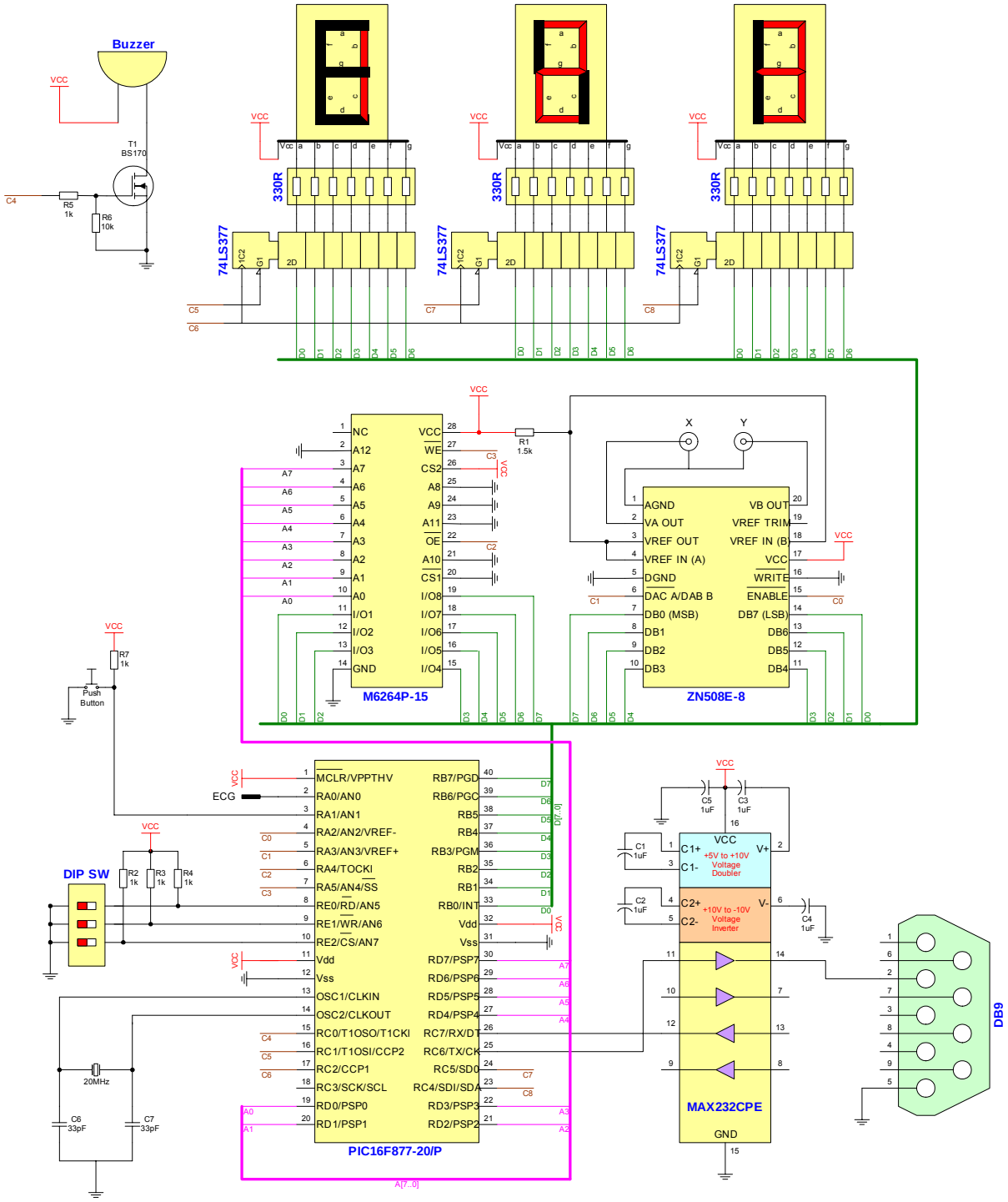


Figure 4.3a. Digital Circuit Diagram

Notice that only the transmit wire is connected to the RS232 cable, the ECG monitor does receive any feedback from the PC, just transmits ECG data continuously. Perhaps it is a good idea to connect the

receive wire, as this increases the possibilities for future product enhancement. For example the chosen PIC can protect blocks of program memory, hence a bootstrap program can be written to check the serial interface before calling the main program. If a certain block of characters are received during bootstrap, it moves into program mode, using a bidirectional communications protocol (RTS, ACK, NAG, etc…) a new program can be download from the PC directly into program memory, when download is complete the new program is called. This offers greater product flexible as software updates (bug fixes, new features) can be downloaded free of charge and download directly into the product using the same serial interface and software used to display the ECG.

Three 7-segment LED displays are used to display the bpm (beat per minute) of the ECG, notice each display is connected to a 330R register network (limit LED current) and a 8-bit latch (74LS377). All 3 latches are connected to the shared data bus; hence only one must be enabled at any one time. The enable line of each latch is used to disable / enable the latch, and the clock inputs are shared and connected to RC4. Note the 74LS377 latch was a bad choice because it is synchronised with a clock, hence the PIC must software generate a clock to operate these latches (time delay), a better latch to use is the 74LS373, this latch is not synchronised to a clock (simplifies PIC code) and is pin compliable with the 74LS377 (no hardware changes required).

A BS170 transistor is used to switch on the buzzer (Piezo Alarm); the PIC controls the transistor via RC0. The reason why the transistor was required was because the PIC cannot supply enough current to power the buzzer. Notice that 1kΩ pull-up resistors are used for the dip-switches and the push button; this holds the digital input pin on the PIC high until the switch is turn ON connecting the pin to ground. The reason why the pull-up resistors are required is to limit current flow when the switches are ON, for example if there was no pull-up resistor there would be a complete short between VCC and GND, when the switch is ON. Note port B has built-in pull-up resistors, the reason why they were not used was because a complete port was required for the data bus.

The ZN508E-8 chip is a dual DAC (digital to analogue converter), it is been configured so that the internal reference voltage of 2.5V (i.e. 0xFF = 2.5V, 0x00 = 0V) is used. This chip has three control lines (see figure 4.3c); only 2 are required because the chip is always in write mode. C1 selects DAC A or DAC B and C0 Enabled or Disables the DAC (Write or Hold).

$\overline{\text{DAC A}}$ / DAC B (C1)	**$\overline{\text{ENABLE}}$ (C0)**	**$\overline{\text{WRITE}}$**	**DAC A**	**DAC B**
L	L	L	WRITE	HOLD
H	L	L	HOLD	WRITE
X	H	X	HOLD	HOLD
X	X	H	HOLD	HOLD

Figure 4.3c. ZN508E-8 logic truth table

The M6264P-15 (RAM) chip has four control lines (see figure 4.3d); only 2 are required because the chip is never powered down. C3 is used to put the chip in read (high) or write (low) mode (assuming C2 is low), if C2 and C3 are both high the chip is disabled allowing for other devices to use the data bus.

$\overline{\text{WE}}$ (C3)	**$\overline{\text{CS}_1}$**	**CS_2**	**$\overline{\text{OE}}$ (C2)**	**Mode**	**I/O Pin**
X	H	X	X	Not Selected (Power Down)	High Z
X	X	L	X		
H	L	H	H	Output Disable	
H	L	H	L	Read	D_{OUT}
L	L	H	H	Write	D_{IN}
L	L	H	L		

Figure 4.3d. M6264P-15 logic truth table

Note each control line is connected directly to the PIC, this uses 9 pins. Many control signals must not be enabled at the same time (PIC software takes care of this), hence it is possible to reduce the number of control lines on the PIC using additional hardware circuitry (e.g. decoder chip and a couple of logic gates), But this increases chip count and product cost, hence this option would have only been considered if there were not enough free PINs on the PIC for all of the required control lines.

4.4. Analogue Circuit Diagram (ECG Amplifier)

Figure 4.4a shows the circuit diagram; this design is extremely simple, the chosen amplifier is analogue devices AD624AD, which is an "Instrumentation amplifier" with a CMRR (Common Mode Rejection Ration) of 110dB and a gain of 1,000. Note the AD624AD chip is the most expensive chip in the design, costing £22.50 (retail price); hence perhaps a lower cost solution is possible. The reason why this expensive chip was chosen was because of its good CMRR (Recall at least 80dB was required) and high precision.

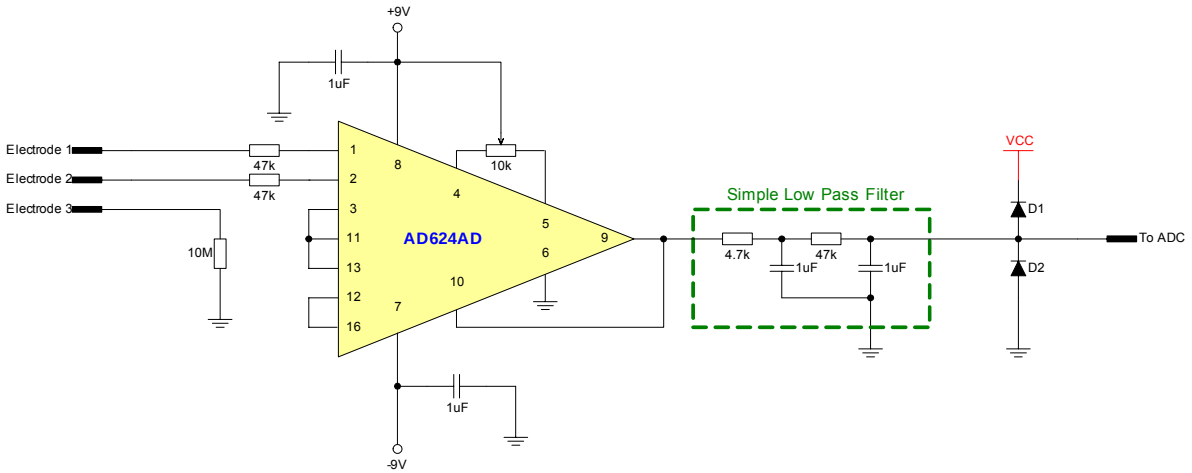


Figure 4.4a. ECG Amplifier circuit

A simple CR second order low pass filter was used to filter high frequencies (anti-aliasing) and the two diodes ensure that the PIC is not damage in the event of over voltage or negative voltages.

4.5. System Powering

Figure 4.5a shows the circuit diagram; clearly a 5V regulator is used to generate a 5 volt DC output to power the circuit. The 0.1µF capacitors absorb line noise, while the 100µF capacitors are used for storage in the event of a minor drop in power (milliseconds) the circuit operation will not be affected.

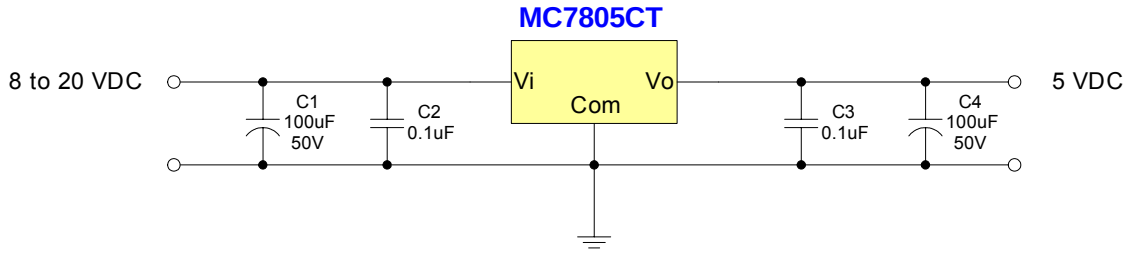


Figure 4.5a. System powering circuit

This means that the circuit has now got a wide operating voltage range as 8 to 20 volts DC will power the circuit. Note there are higher spec 7805 chips available that can operate up to 30 volts DC, if there is a need for a higher voltage range.

The system can be powered from a battery source (e.g. PP3 9V), or a DC power supply (e.g. 12V). It has been decided, not to design a complete power supply unit from scratch, but to use commercially available units. It is important to remember that voltage regulators are not efficient and as the input voltage increases the least efficient they become, energy is lost in the form of heat. A heat sink is normally required to keep the chip within its maximum operational temperature.

4.6. Cost of Components

All order numbers refer to Farnell catalogue.

Digital Circuit: -

Order Number	Item	Qty.	Cost Each	Cost
325-5573	PIC16F877 – 20/P Microcontroller	1	£7.09	£7.09
407-150	MAX232CPE – RS232 Line Buffer	1	£2.40	£2.40
115-873	HT6264-70 (8k x 8) CMOS SRAM, 70ns acc	1	£1.65	£1.65
170-234	20MHz Crystal A147C	1	£0.75	£0.75
747-014	33pF Ceramic Capacitor	2	£0.13	£0.26
664-315	1µF 25V Electrolytic Capacitor	5	£0.27	£1.35
543-380	1kΩ Metal Oxide Film 1% Resistor	5	£0.02	£0.10
134-3040	40-way dip docket	1	£0.33	£0.33
134-2988	16-way dip socket	1	£0.12	£0.12
134-3002	20-way dip socket	4	£0.14	£0.56
134-3038	28-way dip socket	1	£0.22	£0.22
151-138	Push Button, Yellow PCB SPNO	1	£0.64	£0.64
932-840	BS170 Transistor	1	£0.11	£0.11
543-627	10KΩ Metal Oxide Film 1% Resistor	1	£0.02	£0.02
543-421	1.5KΩ Metal Oxide Film 1% Resistor	1	£0.02	£0.02
476-468	330R in-line package	3	£0.31	£0.93
382-371	74HCT377N	3	£0.44	£1.32
583-560	BNC PCB Socket	2	£2.18	£4.36
927-200	Buzzer (Piezo Alarm)	1	£1.51	£1.51
780-091	4-way dip switch	1	£0.40	£0.40
150-812	9-way D-type Socket	1	£0.55	£0.55
	*ZN508E-8 is obsolete.		**Total**	**£24.69**

Analogue Circuit (ECG Amplifier): -

Order Number	Item	Qty.	Cost Each	Cost
102-076	AD624AD	1	£22.50	£22.50
543-780	47KΩ Metal Oxide Film 1% Resistor	2	£0.02	£0.04
543-548	4.7KΩ Metal Oxide Film 1% Resistor	2	£0.02	£0.04
336-907	10MΩ Metal Oxide Film 1% Resistor	1	£0.04	£0.04
543-627	10KΩ Metal Oxide Film 1% Resistor	1	£0.02	£0.02
352-5340	1N4007 Diode	2	£0.038	£0.08
583-560	BNC PCB Socket	1	£2.18	£2.18
746-063	100pF Ceramic Capacitor	4	£0.13	£0.52
330-747	1W – DC to DC converter 5V input, -9, 9V output	1	£5.05	£5.05
			Total	**£30.45**

System Powering Circuit: -

Order Number	Item	Qty.	Cost Each	Cost
701-853	MC7805CT, 5V 1A Voltage Regulator	1	£0.30	£0.30
920-757	100µF 50V Electrolytic Capacitor	2	£0.10	£0.21
746-063	100pF Ceramic Capacitor	2	£0.13	£0.26
178-701	Clip On Heat Sink	1	£0.36	£0.36
			Total	**£1.13**

Total component cost = 24.69 + 30.45 + 1.13 = £56.27

5.0. SOFTWARE DEVELOPMENT

The philosophy used during the development of the PIC code was to keep it simple, straightforward, comprehensible, and to a minimum. There are many small programs designed for testing the hardware and ideas, each program is labelled mark 1, 2, 3, etc… The end result is that the PIC code is gradually built up step-by-step, instead of writing the entire program at once. This ensures that operational results are obtained, as testing producers are carried out at each stage, while if the program was written all at once, there is little chance it will work and could prove difficult to debug.

The high-level programming language C was chosen and not the low-level assembly code normally associated with PIC programming. There are many advantages for using C including: ease of programming, ease of modification, reusability of code, use of standard functions (e.g. printf, getc, putc, etc…), etc… But there is one drawback, C code is much less efficient, for example typically code produced by the C compiler (CCS) is at least twice as large as that programmed in assembly. Since the PIC16F877 has a large program store (8k) and runs at 20MHz this drawback is not a problem.

5.1. Mark1.c (Test RS232 Communications)

This program is extremely simple; basically it initialises RS232 communications and transmits “Testing…” once a second (see figure 5.1a). This program was used to test RS232 communications (transmit mode), including MAX232, PIC UART and cable. The PIC is connected to a PC which is running a terminal program to receive the incoming characters. Complete source code for mark1.c can be found in appendix 1, section A1.1.

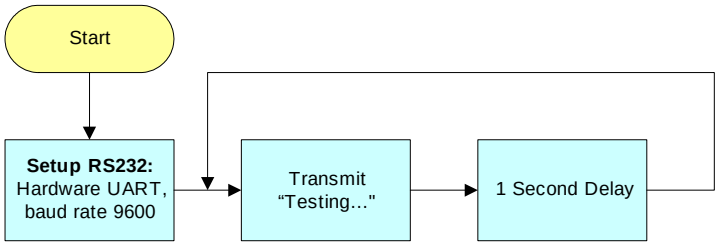


Figure 5.1a. Mark1 flowchart

5.2. Mark2.c (Test PIC ADC)

This program reads ADC channel AN0 and transmits the reading through RS232 using the final year project (PC based oscilloscope) real-time frame structure with a fixed sampling delay of 8mS (that’s a sample rate of 125Hz) before repeating (see figure 5.2a). Complete source code for mark2.c can be found in appendix 1, section A1.2.

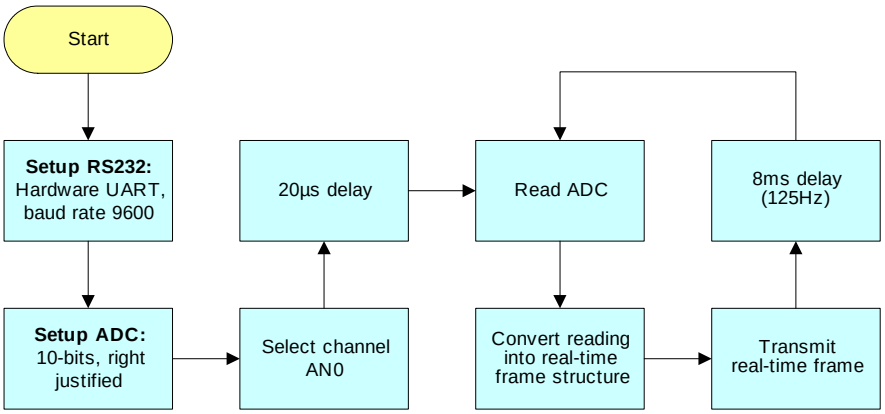


Figure 5.2a. Mark2 flowchart

Appendix 2 gives a summary of Colin McCord's final year project (PC based oscilloscope program), basically a Windows application called "scope.exe" displays analogue waveforms, If ECG data is transmitted using the real-time frame structure, the scope program can be used to display the ECG on a PC (scope program used in scroll mode, hence waveform scrolls across the screen).

5.3. Mark3.c (Test RS232 Communications, Baud-Rate Set by Dip-Switches)

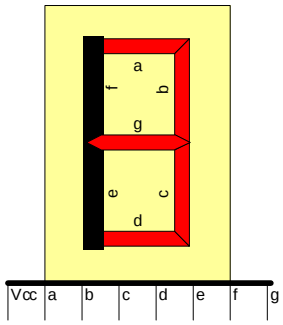
Same as mark1.c with the addition of function SetBaudRate(), this function sets the UART baud-rate based on the positions of the DIP switches that are connected to port E. This program is used to test the reading of the DIP switches and RS232 communications at different baud rates. Note that the PIC must be running at 20MHz because the percentage error for 115Kbps is too large at slower clock speeds. Complete source code for mark3.c can be found in appendix 1, section A1.3.

5.4. Mark4.c (Read ADC, Transmit to PC, Adjustable Baud-Rate)

Same as mark2.c with the addition of function SetBaudRate(), this function sets the UART baud-rate based on the positions of the DIP switches that are connected to port E. This program is uses the scope program (final year project) to display an ECG signal in real-time scrolling across the screen. Complete source code for mark3.c can be found in appendix 1, section A1.4.

5.5. Mark5.c (Test 7-Segment Display and Buzzer)

This program is simple, basically the 7-segments are programmed to count from 000 → 999 → 000 forever. Every time the most significant digit increments the buzzer will beep. Note this program also fully configures the input / output ports of the PIC, setups the controls lines with user-friendly names, and contains a lookup table for setting the 7-segment displays. Complete source code for mark5.c can be found in appendix 1, section A1.5.



	d_7	d_6(g)	d_5(f)	d_4(e)	d_3(d)	d_2(c)	d_1(b)	d_0(a)
0	0	1	0	0	0	0	0	0
1	0	1	1	1	1	0	0	1
2	0	0	1	0	0	1	0	0
3	0	0	1	1	0	0	0	0
4	0	0	0	1	1	0	0	1
5	0	0	0	1	0	0	1	0
6	0	0	0	0	0	0	1	0
7	0	1	1	1	1	0	0	0
8	0	0	0	0	0	0	0	0
9	0	0	0	1	0	0	0	0

Figure 5.5a. 7-segment lookup table (active low)

5.6. Mark6.c (Test Timer Interrupts)

This program uses all of the PICs built-in timers, basically timers 0-2 are setup to cause an interrupt. An interrupt subroutine has been written for each timer; at preset intervals (0.5s, 0.25s, & 0.1s) a message is transmitted to the PC. The main program is stuck in a loop transmitting "main…" every second, Timer0_ISR transmits "Interrupt_T0…" every 0.5s, Timer1_ISR transmits "Interrupt_T1" every 0.25s and Timer2_ISR transmits "Interrupt_T2" 10-times a second. These messages are received using a PC running a terminal program, hence testing of the PIC timer interrupts including crude timing analysis is achieved, for example there should be two "Interrupt_T0…" messages between every "Main…". See figure 5.6a for a simplified flowchart of the program. Complete source code for mark6.c can be found in appendix 1, section A1.6.

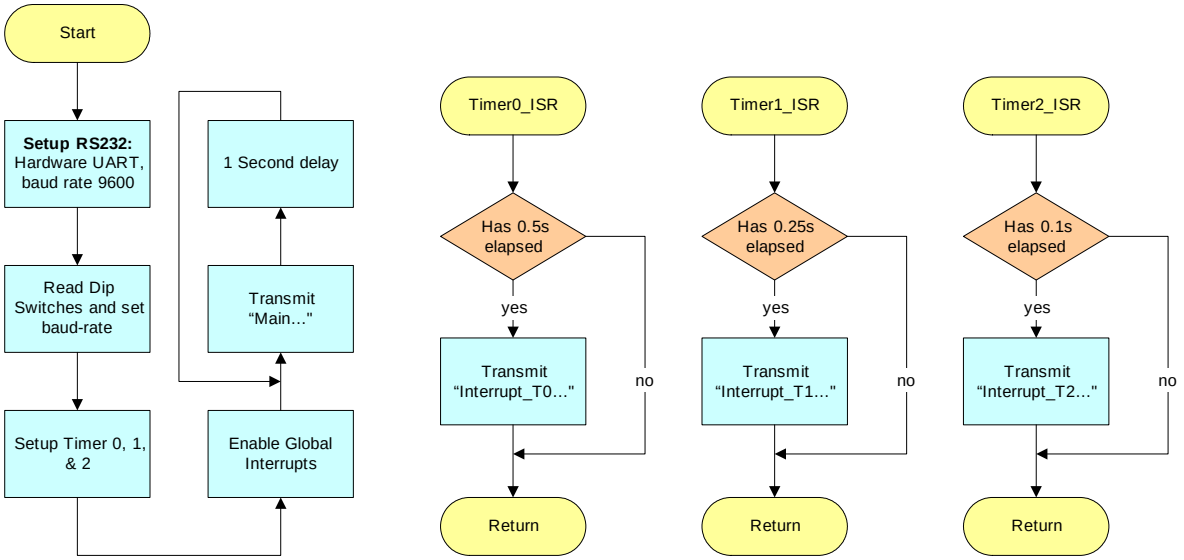


Figure 5.6a. Mark6 flowchart

5.7. Mark7.c (Test Dual DAC)

This program fully tests the 8-bit dual DAC (ZN508E-8), a function has been written to write to the DAC "SetDAC(char x, char y) ". This function has two inputs x and y, x is written to DAC A and y is written to DAC B. A simple test routine loops continuously in the main program, basically two variables are setup (DAC_X initialised to 0x00, DAC_Y initialised to 0xFF), these variables are inputted into the SetDAC() function, then DAC_X is incremented and DAC_Y is decremented, after a 78µS delay these new values are written into the DAC and the process repeats forever.

The reason for the 78µS delay was to limit the refresh rate of the CRT to 50Hz e.g. 256 writes are required to complete one cycle, hence 50 x 256 = 12,800 Hz (loop refresh rate) that's approximately a delay of 78µS. Note variables DAC_X and DAC_Y are 8-bits hence DAC_X overflows from 0xFF to 0x00 and DAC_Y underflows from 0x00 to 0xFF. An oscilloscope in x/y mode can easily be used to test if this program and the hardware is working correctly, e.g. there should be a diagonal line across the display. Complete source code for mark7.c can be found in appendix 1, section A1.7.

5.8. Mark8.c (Test External RAM Chip)

This program carries out three tests: -

1. Fill all memory locations with 0xFFh and check.
2. Fill all memory locations with 0x00h and check.
3. Fill all memory locations with the address and check.

RS232 communication is used to relay the results to the PC; a terminal program is used to receive and display the results. This program is designed to test the functions used to access the RAM (ReadRAM, WriteRAM), and every usable location (0x00 to 0xFF) on the RAM chip.

Notice that there are 1 cycle delays in the ReadRAM and WriteRAM functions these delays make sure that the address and data lines are valid before a read or write operation is completed. Note the RAM chip used has a reassembly fast access time; hence the RAM may operate correctly with these delays removed. Complete source code for mark8.c can be found in appendix 1, section A1.8.

5.9. Mark9.c (ECG Monitor, CRT & PC Display)

This program is designed to display ECG signal on a CRT display (time compressed memory) and/or a PC (via RS232). The program is separated into two main parts; sampling (interrupt) and refreshing CRT display (main function). The program is more complex than the previous programs and it's recommend that the reader studies the source code in appendix 1, section A1.9 in detail. Because this program is the main program (not a test program) mark9.ls has also been included in appendix 1 (section A1.10), this file contains complete assembly code (with C code) generated by the C complier, hence detailed timing analyse can be made (e.g. 1 cycle for most instructions, 2 for jump instructions, 1 instruction cycle takes 200ns to execute at 20MHz).

5.9.1. Time Compressed Memory

Figure 5.9.1a illustrates the concept of time-compressed memory, basically a large array is filled with ADC readings from right to left and the CRT display is refreshed by continuously scanning through the array[]. Because ADC readings scroll through the array[] from right to left, this gives the appearance that the ECG signal is scrolling across the screen.

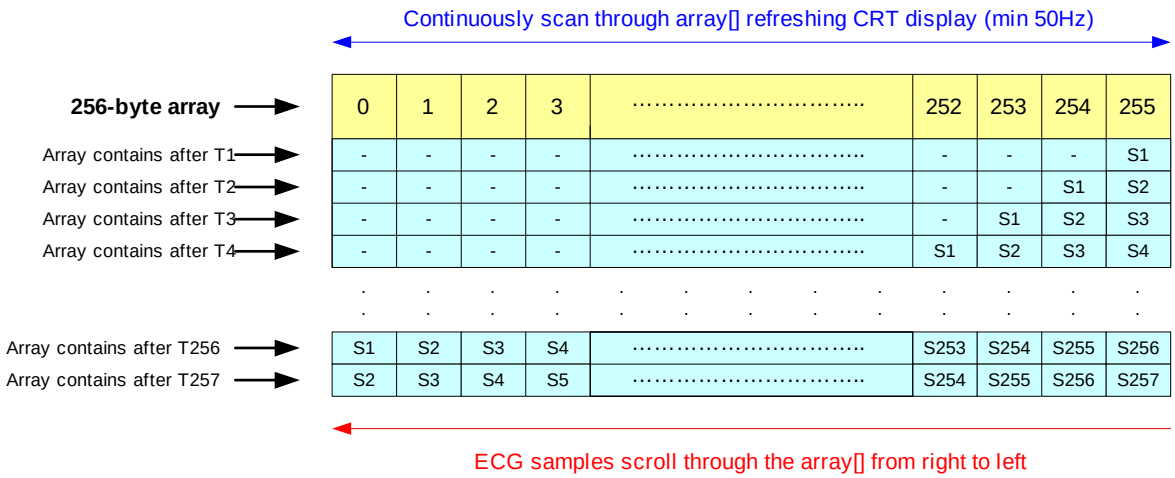


Figure 5.9.1a. Illustration of time-compressed memory

The CRT display must be refreshed at least 50 Hz (50-times a second) to avoid flicker (waveform dissipates before redrawn), the screen is 256 bytes wide, hence the required scan rate is 12,800 Hz (50 x 256), therefore there must not be anymore than a 78.125µs (1/12,800) delay moving between each scan location. The array is scanned on both the forward and return joining (e.g. 0 → 255 → 0 → 255), this avoids the need for fly back (move trace off screen and quickly move it back to the start location).

The initial design specification states that 2 seconds of the ECG signal should be displayed on the screen, hence a sample rate of 128 Hz is required to achieve this, e.g. it take 2 seconds to fill the array (256-byte array, sampling at 128 bytes per second, hence 256/128 seconds required to fill array). For a sample rate of 128 Hz the delay between samples is 7.81ms (1/128), hence each 'T' (T1, T2, T3, etc…) shown in figure 5.9.1a represents 7.81ms.

5.9.2. Internal RAM Used For Time Compressed Memory

The chosen PIC has more than enough RAM to implement time compressed memory, but there is a problem the PICs internal RAM is split into four banks. Unfortunately the C compiler does not allow arrays to be created over multipliable memory banks; hence the solution was to create four arrays (one in each bank) of 64 bytes.

There are two C functions (WriteArray, ReadArray) which give the appearance of one large 256 byte array. Function WriteArray(char address, char data) is used to write to the array, the address variable (0 to 255) specifies the location to store the data, this function automatically works out which bank to save the data in. Function ReadArray(char address) returns the data within the specified address, automatically calculating which bank the required address is in.

Figure 5.9.2a. Illustration of how time-compressed memory was implemented using internal RAM

5.9.3. Interrupt Sampling Routine

Timer2 is setup to interrupt the PIC 128.48 (as close to 128Hz as possible) times a second. When timer2 interrupt service routine is called the first thing it does is check the state of the push button, if the push button is pressed the routines ends, hence the ECG display is paused as no new readings are being taken.

If the screen is not paused the PIC ADC is read (10-bit), this 10-bit reading is then converted to 8-bits and stored into array[] location pStartOfArray, and then the pStartOfArray pointer is incremented. This is a simple trick for putting the new ADC reading at the end of the array, e.g. put the new reading at the start of the array and increment the start of array pointer, hence the new value is actually placed at the end of the array, overriding the oldest value.

The 10-bit ADC reading is then reformatted into the correct frame-structure for use with the scope program (final year project) and transmitted through the RS232 port.

This interrupt sample routine contains 115 assembly level instructions most of which required 200ns (1~ @ 20MHz) to execute, hence an approximate execution time of the interrupt is 23µS (115 x 200ns). Clearly this should not effect the redraw of the CRT display as this routine is called every 7.8ms (1/128), this allows for many redraws between calls (100 assuming refresh rate is limited to 50Hz, which it is not) and is executed in less than 78µs (min delay between sweeps, to ensure 50Hz).

5.9.4. CRT Refresh Main Routine

Figure 5.9.4a shows how the CRT screen is refreshed, the trace is drawn from left to right (Bforward == TRUE), then right to left (Bforward == FALSE). Pointer PStartOfArray points to the start of the array, hence pStartOfArray + pScanPOS specifies the memory location to be read for Y, and pScanPOS is X.

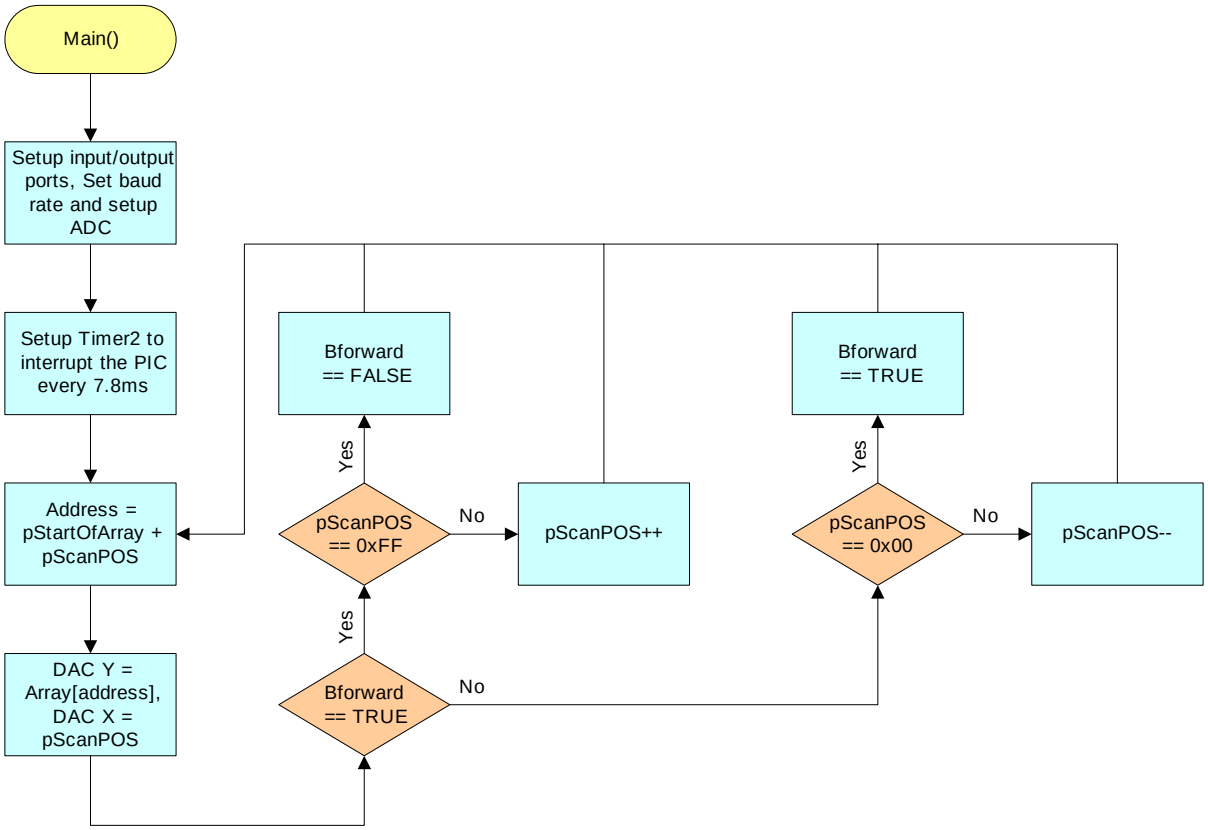


Figure 5.9.4a. Flowchart of CRT refresh main routine

The refresh rate of the CRT can be manually calculated as done for the “interrupt sample routine”, but there is an easier method Microchip’s MPLAB can be used to simulate the program. Using the stopwatch feature it is possible to measure how long certain parts of the program takes to execute. Figure 5.9.4a shows a screen dump of MPLAB, where line “address = pStartArray + pScanPOS” is setup as a break point, the program is then single stepped to that location and the stopwatch is reset. The program is then simulated until the break point is detected, this stop the simulation and the stopwatch timer, which now reads the time taken to complete one loop.

The stopwatch measured the time taken to complete one loop to be 21.60µS (108 cycles), hence the time taken make one pass across the screen is 5.5296mS (21.60µS * 256), that’s 180 Hz. Therefore the PIC is actually refreshing the screen nearly 4-times after than required to ensure a flicker free trace. Note this figure is not completely accurate because it did not take into account the time spent processing the interrupt routine (23µs) that is called 128-times a seconds. Therefore a more accurate approximation would be 5.5296ms +

2.944ms (23µs x 128) = 8.4738ms, that's 118Hz (1/8.4738ms). Which means there is excess processing power available for calculation of beats per minute (see mark10.c).

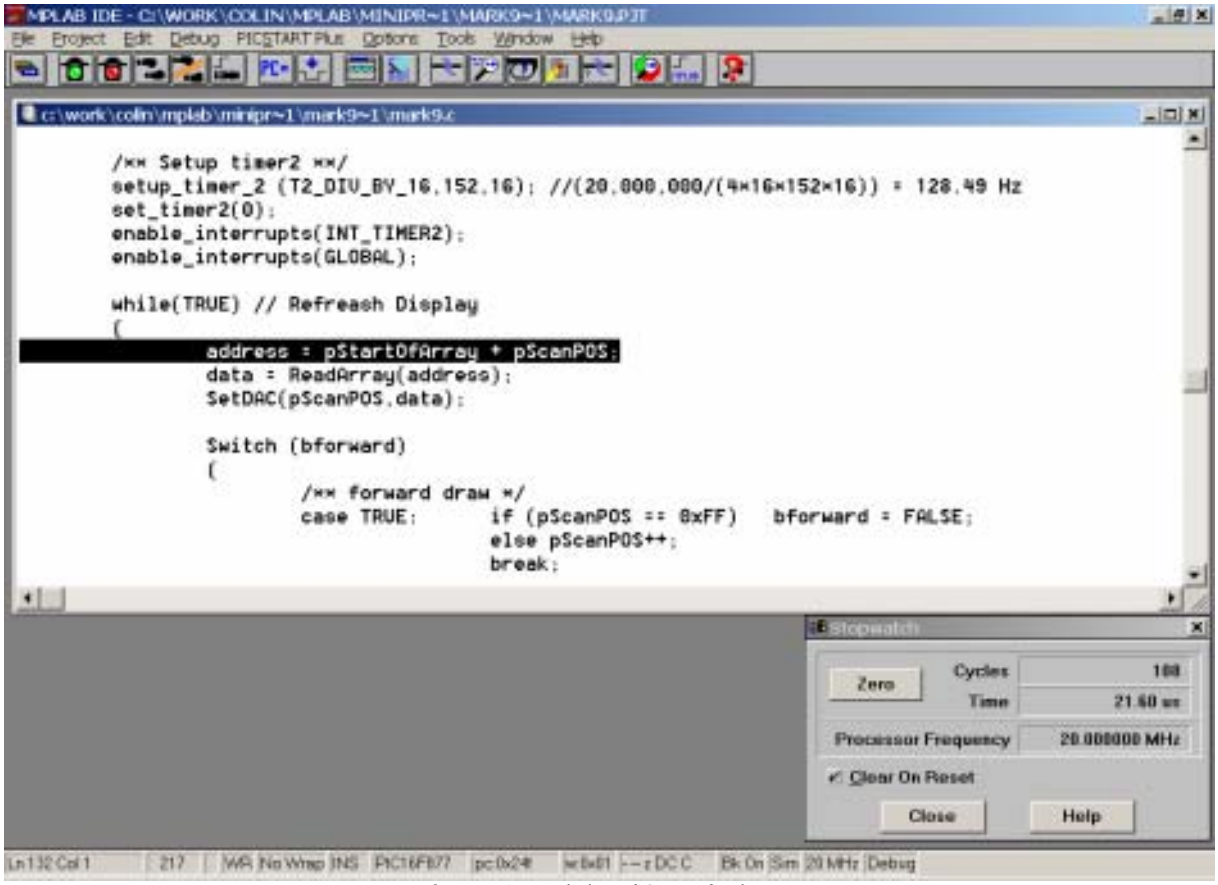


Figure 5.9.4a. Timing of CRT refresh rate.

5.10. Mark10.c (ECG Monitor, CRT & PC Display, Buzzer and BPM Display)

This program is extremely complex it's recommend that the reader studies the source code in appendix 1, section A1.10 in detail. Because this program is the main program (not a test program) mark10.ls has also been included in appendix 1 (section A1.11, very long file), this file contains complete assembly code (with C code) generated by the C complier, hence detailed timing analyse can be made (e.g. 1 cycle for most instructions, 2 for jump instructions, 1 instruction cycle takes 200ns to execute at 20MHz).

This program is based on mark9.c, with the addition of displaying beats per minute on the 7-segment display and beeping (buzzer) on every ECG peak. The program has an additional interrupt (timer0), that is used to calculate the beats per minute and the "interrupt sampling routine" has been modified so that the peaks of the ECG signal are detected (would not work correctly on a real ECG waveform, hence more development work required here).

5.10.1. Modifications Made to the Interrupt Sampling Routine

Figure 5.10.1a shows the modifications made to the "Interrupt Sampling Routine", variable ECG_MAX is compared with the sampled ADC reading and if this ECG_MAX variable is less than the sampled ADC reading, the ECG_MAX variable is updated. The program then detects the peak of the ECG waveform by checking if the sampled ADC reading is greater or equal to ECG_MAX-50, the reason for the -50 is to allow for some voltage fluctuations. When the peak is detected the buzzer is switched ON and the ECG peak counter is incremented if CountDelay >= 25 (makes sure not to detect the same peak twice, e.g. a delay

before next peak, this limits max bpm to about 300). The buzzer is switched off after 15 calls, that's 117ms (7.8ms x 15) and the CountDelay is incremented.

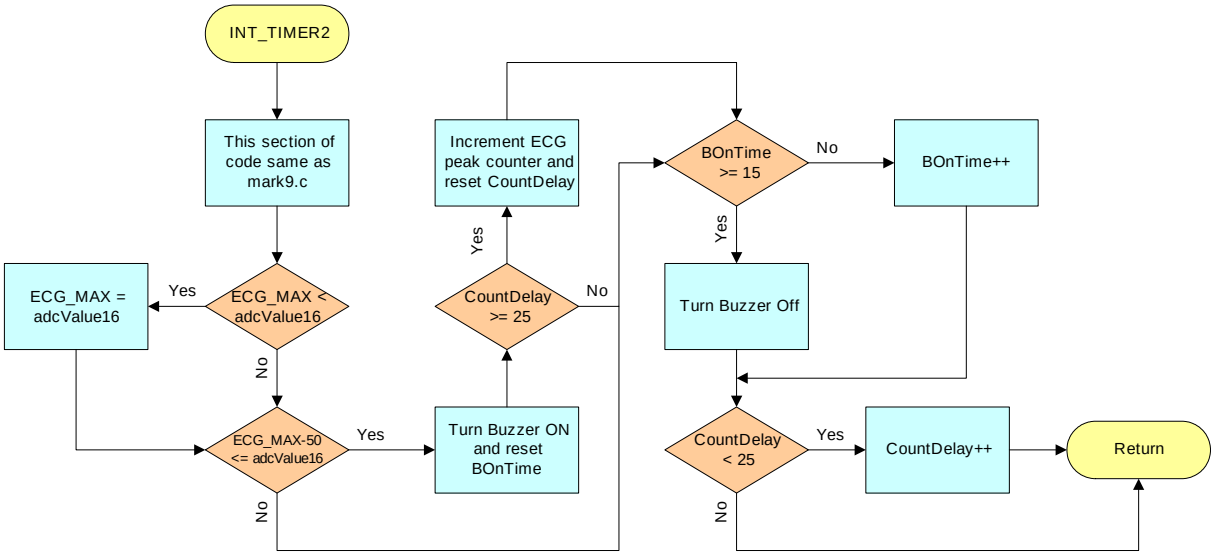


Figure 5.10.1a. Flowchart showing modifications made to interrupt sampling routine

5.10.2. 7-Segment Interrupt Routine

Timer0 is used to cause an interrupt 76-times a second, variable int_count0 is decremented, until this variable reaches zero the routine ends here. When int_count0 is equal to zero this means a second has elapsed and int_count0 is reset to 76. Counters SEG_Update and SEG_Average are incremented.

The ECG_Peak_Counter which contains the number of beats that occurred during the last second is stored in bpsArray[] at location SEG_Average-1 (-1 required because array range is between 0 to 9, while counter counts from 1 to 10). The ECG_Peak_Counter is then reset so that the number of beats in the next second can be counted.

Once SEG_Update is larger or equal to 4 (4 seconds has elapsed) variable UpdateBPM is made equal to TRUE and SEG_Update counter is reset (e.g. update LED display every 4 seconds). Once counter SEG_Average is larger or equal to 10 (10 seconds has elapsed) the counter is reset.

This interrupt sample routine contains 31 assembly level instructions most of which required 200ns (1~ @ 20MHz) to execute, hence an approximate execution time of the interrupt is 6.2µS (31 x 200ns) once per second.

5.10.3. Modifications Made to the CRT Refresh Main Routine

This line of code was added (inside refresh loop): -

```
/** Update 7-segment Display, approx. every 4 seconds **/
If (UpdateBPM == TRUE && pScanPOS == 0) Update7SEG();
```

Function Update7SEG() is called when variable UpdateBPM is equal to TRUE (occurs every 4 seconds, set in the "7-Segment Interrupt Routine") and pScanPOS equals zero (this means that the CRT display is always refreshed before calling function Update7SEG).

5.10.4. Updating 7-Segment Displays

Function Update7SEG() carries out the update in 5 steps, the reason of this is to reduce the affect caused to the waveform, e.g. between each step (some steps require may require multiple calls) the waveform redrawn, for 50Hz each step must execute in less than 78µs.

Step 1: Calculate the number of beats per 10 seconds, this is achieved by adding all of the locations of bpsArray[] together. Then calculate beats per minute this is achieved by multiplying beats per 10 seconds by 6, since timing is critical it is faster to add the BeatsPer10Sec variable with itself 6-times. Reset unit, ten, and hundred variables and move to step 2 when the function is next called (e.g. waveform redrawn between calls).

Timing: *49 assembly level instructions, hence an approximate execution time of 9.8µs.*

Step 2: Retrieve the hundreds from the m_bpm variable by subtracting 100 and checking for underflow. Note unsigned 8-bit numbers are used hence a trick is used to detect the underflow, e.g. a temperately 8-bit variable is made equal to m_bpm before it is subtracted by 100, hence if this temperately variable is smaller than m_bpm-100 clearly a underflow has occurred. If no underflow occurred increment the 'hundred' variable and exit the function, hence the waveform is redrawn and this function is called again repeating this step. This continues until an overflow occurs, hence 100 is added to m_bpm leaving only the units and tens. Move to step 3 when the function is next called.

Timing: *26 assembly level instructions, hence an approximate execution time of 5.2µs.*

Step 3: Retrieve the tens from the m_bpm (note hundreds already removed, leaving only the tens and units) variable by subtracting 10 and checking for underflow. Note unsigned 8-bit numbers are used hence a trick is used to detect the underflow, e.g. a temperately 8-bit variable is made equal to m_bpm before it is subtracted by 10, hence if this temperately variable is smaller than m_bpm-10 clearly a underflow has occurred. If no underflow occurred increment the 'ten' variable and exit the function, hence the waveform is redrawn and this function is called again repeating this step. This continues until an overflow occurs, hence 10 is added to m_bpm leaving only the units. Move to step 4 when the function is next called.

Timing: *25 assembly level instructions, hence an approximate execution time of 5µs.*

Step 4: Retrieve the units from the m_bpm (note hundreds & tens already removed, leaving only the units) variable by subtracting 1 and checking for underflow. Note unsigned 8-bit numbers are used hence a trick is used to detect the underflow, e.g. a temperately 8-bit variable is made equal to m_bpm before it is subtracted by 1, hence if this temperately variable is smaller than m_bpm-1 clearly a underflow has occurred. If no underflow occurred increment the 'unit' variable and exit the function, hence the waveform is redrawn and this function is called again repeating this step. This continues until an overflow occurs. Move to step 5 when the function is next called.

Timing: *23 assembly level instructions, hence an approximate execution time of 4.6µs.*

Step 5: Update 7-segment display using lookup table and make UpdateBPM = FALSE, stopping the function being called until another 4 seconds has passed. Let SEG_state = 0, hence when function is called again 4 seconds later it starts in step 1.

Timing: *44 assembly level instructions, hence an approximate execution time of 8.8µs.*

Clearly updating the 7 segment displays will not cause any problems to the waveform, as timing analysis shows that processing delays are well within limits.

6.0. TEST RESULTS

Photograph of the circuit is shown in figure 6.0a; the circuit was constructed on stripboard, note there was not enough time available to built and test the analogue circuit (ECG amplifier), hence only the digital and system powering circuits were constructed.

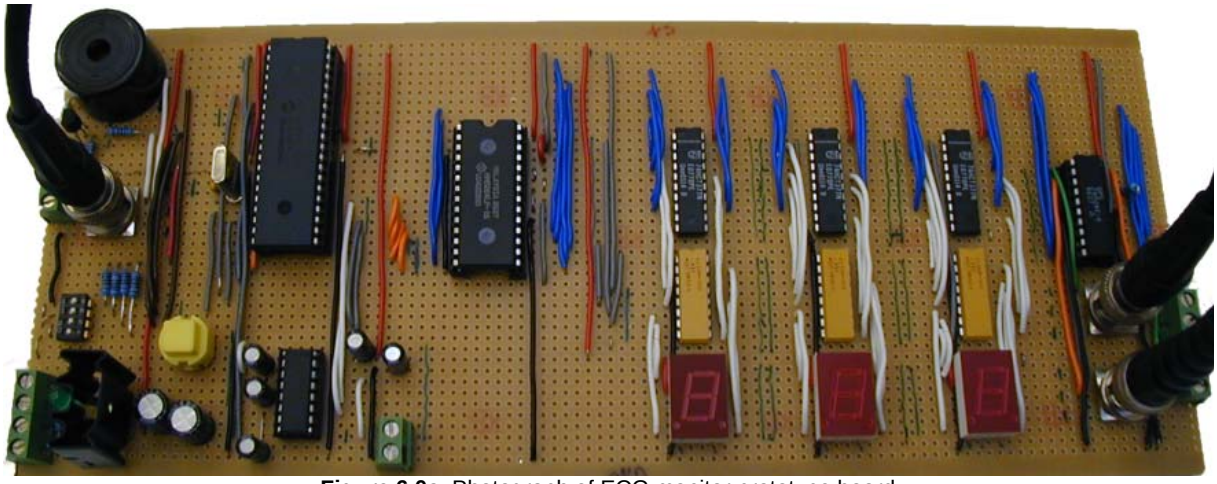


Figure 6.0a. Photograph of ECG monitor prototype board

Photograph of the test setup is shown in figure 6.0b, equipment used included: dual trace oscilloscope, DC power supply, signal generator, PIC programmer, and a laptop computer running MPLAP / scope.exe.

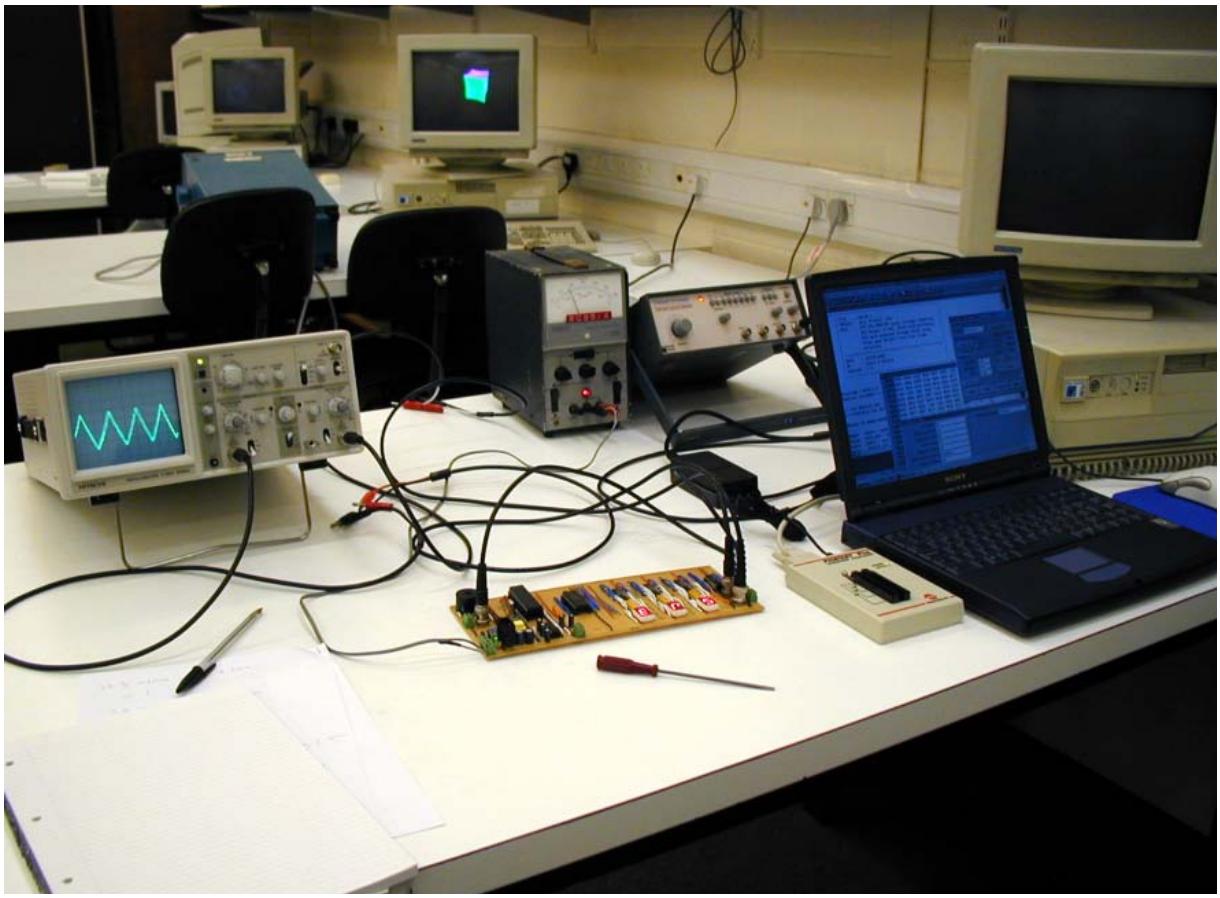
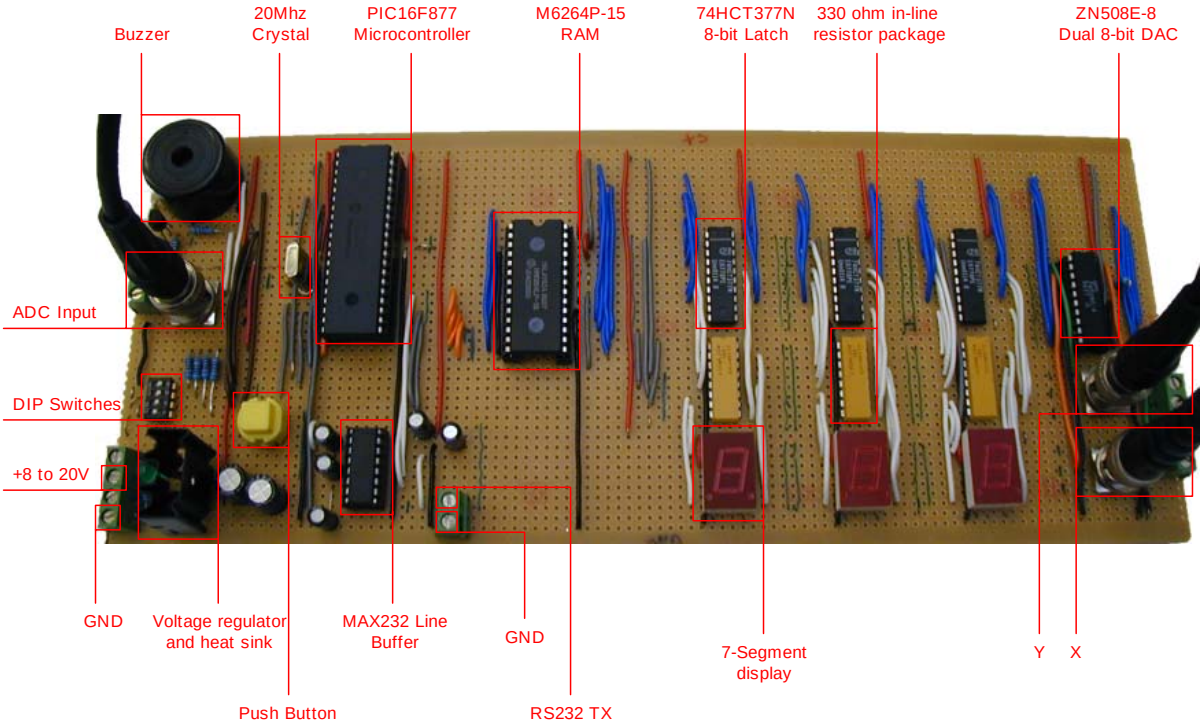


Figure 6.0b. Photograph of test setup

6.1. Prototype Layout



6.2. Test 1: Test RS232 Communications

Program mark1.c was compiled and loaded into PIC using Microchip MPLAB. This program is extremely simple, basically it transmits “Testing…” continuously with a 1 second delay between each “Testing…”. An RX terminal program running on PC was used to receive the incoming characters.

At first, it was discovered that nothing was happening. Program was checked and double checked, no problems were found. After sometime it was discovered that the oscillator pins on the PIC were short-circuited, after removing the short the PIC started to operate as expected.

Screen dump of terminal program after a couple of minutes: -

```
RX Terminal
-----------
Waiting for message (press a key to stop)
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...Testing...Testing...Testing...
Testing...Testing...Testing...Testing...Testing...
```

Clearly the test was a success, and it has been proven that the transmit mode of the RS232 communications is working. Note: Baud rate = 9600bps, OSC = 20 MHz, RS232 cable length = 1.5m (homemade unshielded cable).

6.3. Test 2: Test PIC ADC

Program mark2.c was compiled and loaded into PIC using Microchip MPLAB. Basically it reads the PICs ADC (AN0), and transmits the result through RS232 using the real-time frame format (final year project see appendix 2 for details), every 8ms (sample rate of 125Hz). The scope.exe program was used to display the real-time frames graphically and a signal generator was use to generate a 0 to 5V (e.g. 5V peak to peak with a DC offset of 2.5V) sine-wave at a variable frequency (e.g. 1Hz, 2Hz, 5Hz, etc…).

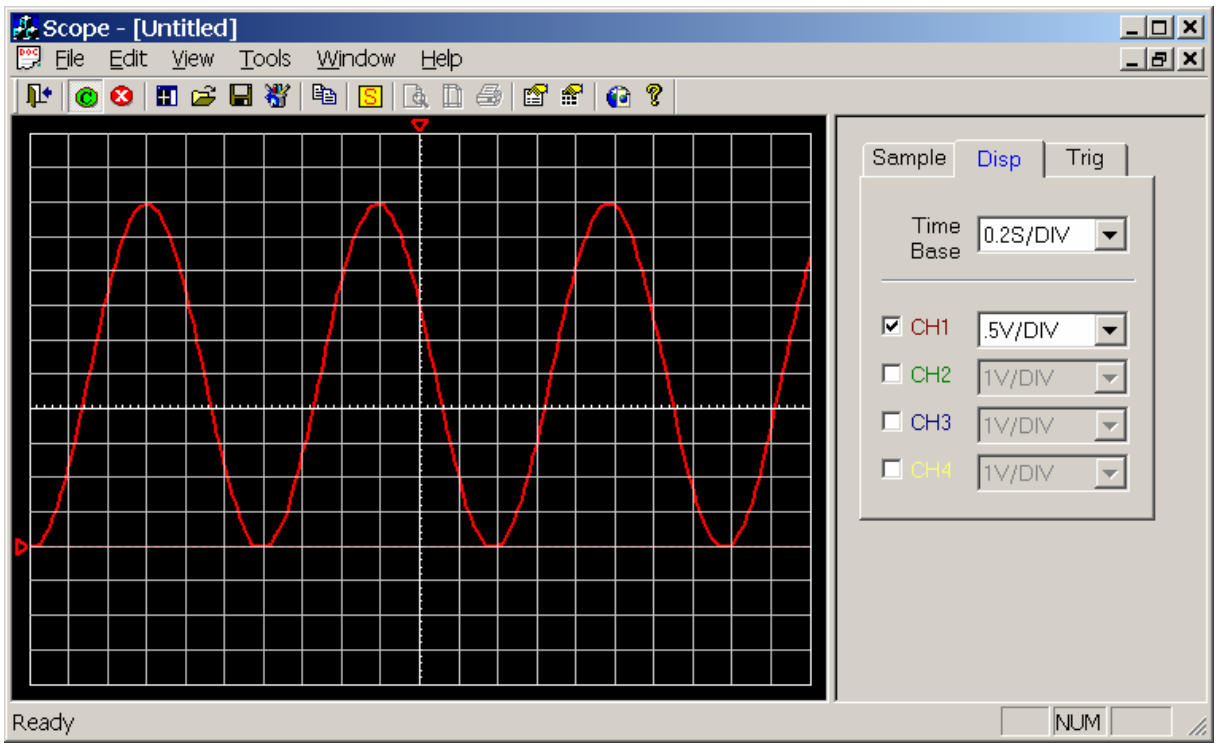


Figure 6.3a. Screen dump of scope program displaying sampled waveform

6.4. Test 3: Test RS232 Communications, Baud-Rate Set by DIP-Switches

Program mark3.c was compiled and loaded into the PIC using Microchip MPLAB. This program is extremely simple (basically the same as mark1.c except baud rate is set by the DIP switches), basically it reads the DIP switches and sets the baud rate, then it transmits "Testing…" continuously with a 1 second delay between each "Testing…". An RX terminal program running on the PC was used to receive the incoming characters.

All baud rates were tested and there were no problems.

6.5. Test 4: Test PIC ADC, Adjustable Baud-Rate

Program mark4.c was compiled and loaded into the PIC using Microchip MPLAB. This program is extremely simple (basically the same as mark2.c except baud rate is set by the DIP switches). All baud rates were tested successfully and there were no problems displaying the waveform.

6.6. Test 5: Test 7-Segment Display and Buzzer

Program mark4.c was compiled and loaded into the PIC using Microchip MPLAB. This program is simple; basically the 7-segment displays count from 000 → 999 → 000 forever and every time the most significant digit increment the buzzer will beep. This test was a complete success.

6.7. Test 6: Test Timer Interrupts

Program mark6.c was compiled and loaded into the PIC using Microchip MPLAB. This program is simple, basically it setups all of the PIC onboard timers to interrupt the main program at preset timer intervals, then "Main…" is transmitted once every second forever. Timer 0 interrupt is used to transmit "Interrupt_T0" twice a second, timer 1 interrupt is used to transmit "Interrupt_T1" 4-times a second and timer 2 interrupt is used to transmit "Interrupt_T2" 10-times a second. An RX terminal program running on the PC was used to receive the incoming characters. This test fully tests the timer interrupts, which will be used for sampling and updating 7-segment displays.

Screen dump of terminal program after a couple of seconds: -

```
Waiting for message (press a key to stop)
Main...Interrupt_T2...Interrupt_T2...Interrupt_T1...Interrupt_T2...Interrupt_T2.
..Interrupt_T2...Interrupt_T0...Interrupt_T1...Interrupt_T2...Interrupt_T2...Int
errupt_T2...Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T0...Interrup
t_T1...Interrupt_T2...Main...Interrupt_T2...Interrupt_T2...Interrupt_T1...Interr
upt_T2...Interrupt_T2...Interrupt_T2...Interrupt_T0...Interrupt_T1...Interrupt_T
2...Interrupt_T2...Interrupt_T2...Interrupt_T1...Interrupt_T2...Interrupt_T2...I
nterrupt_T0...Interrupt_T1...Interrupt_T2...Main...Interrupt_T2...Interrupt_T2..
.Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T2...Interrupt_T0...Inte
rrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T1...Interrupt_T2...Interrupt
_T2...Interrupt_T2...Interrupt_T0...Interrupt_T1...Interrupt_T2...Main...Interru
pt_T2...Interrupt_T2...Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T2
...Interrupt_T0...Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T1...In
terrupt_T2...Interrupt_T2...Interrupt_T2...Interrupt_T0...Interrupt_T1...Interru
pt_T2...Interrupt_T2...Main...Interrupt_T2...Interrupt_T1...Interrupt_T2...Inter
rupt_T2...Interrupt_T0...Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_
T2...Interrupt_T1...Interrupt_T2...Interrupt_T2...Interrupt_T2...Interrupt_T0...
```

Clearly the timer interrupts are working, note that there were 11 "Interrupt_T2" between "Main…", hence timing is slightly out (should be 10). Timer 1 and Timer 0 timing were correct, allow it is important to realise that the main program uses a loop delay, this delay does not take into account the delay taken to process the interrupts hence this may explain way timer 2 appeared to be slightly out.

6.8. Test 7: Test Dual DAC

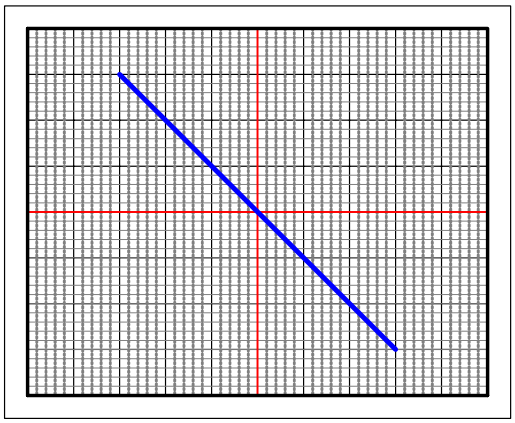


Figure 6.8a. Sketch of Oscilloscope display

Program mark7.c was compiled and loaded into the PIC using Microchip MPLAB. This program fully tests the 8-bit dual DAC, basically DAC A is incremented and DAC B is decremented continuously. Hence an oscilloscope in x/y mode was used to detect a diagonal line across the display (see figure 6.8a).

But it did not work first time, there were a number of time-wasting problems. First it was forgotten to connect the analogue and digital grounds together, secondly the write enable pin was held high, when it should have been held low. But there was still a problem only one DAC channel was working, after sometime it was discovered that the DAC selection control line was shorted with the ground line hence DAC A was always selected. Removed the short

and discovered that the DAC worked perfectly. In total about 5 hours was lost finding these minor problems.

6.9. Test 8: Test External RAM Chip

Program mark8.c was compiled and loaded into the PIC. This program carries out three tests: -

1. Fill all memory locations with 0xFFh and check.
2. Fill all memory locations with 0x00h and check.
3. Fill all memory locations with the address and check.

RS232 communication is used to relay the results to the PC; a terminal program running on a PC was used to receive and display the results.

Screen dump of the result is shown below: -

```
RX Terminal
-----------
Waiting for message (press a key to stop)
TEST RAM CHIP...

Test 1 - Fill RAM with 0xFF ---> Passes = 256, Fails = 0

Test 2 - Fill RAM with 0x00 ---> Passes = 256, Fails = 0

Test 3 - Fill RAM with Address ---> Passes = 256, Fails = 0

End of RAMTEST
```

Clearly the RAM test was a success, but the first attempt was a failure. Later it was discovered that the write enable line was floating and the control line that was supposed to be connected to that pin was connected to the wrong pin.

6.10. Test 9: Test ECG Monitor, CRT & PC Display

Program mark9.c was compiled and loaded into the PIC. This program is designed to display an ECG signal on a CRT display (time compressed memory) and/or a PC (via RS232). A signal generator was used to generate a trianglewave because the analogue ECG amplifier was not constructed.

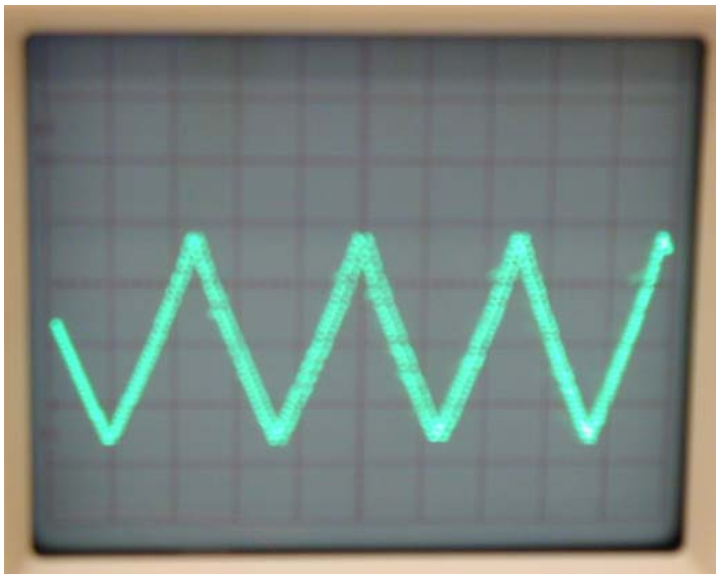


Figure 6.10a. Photograph of oscilloscope display

The trianglewave successfully scrolled across the screen of an oscilloscope in X/Y mode. Photograph of oscilloscope display shown in figure 6.10a.

But there was a double image that appeared to improve as the waveform scrolled across the screen. This can clearly be seen in figure 6.10b.

Recall that the waveform is being drawn both on the forward and the return journeys. It was discovered that the oscilloscopes available have trouble drawing the waveform on the return journey as a more expensive analogue storage oscilloscope displayed the waveform perfectly.

The time it took for the waveform to

travel the full length of the screen was timed, e.g. waited for a peak to appear started a stopwatch and followed the peak across the screen, stopping the stopwatch when it disappeared off the left hand side of the screen. This time was exactly 2 seconds as designed. The push button switch successfully paused to display.

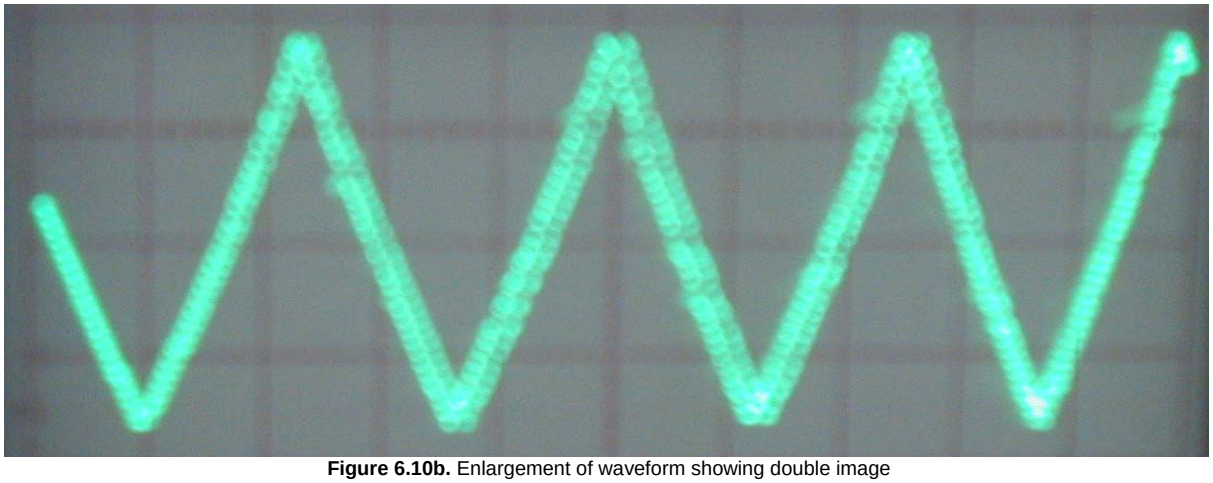


Figure 6.10b. Enlargement of waveform showing double image

The waveform was also successfully displayed on the scope program (final year project) running on a PC, see figure 6.10c.

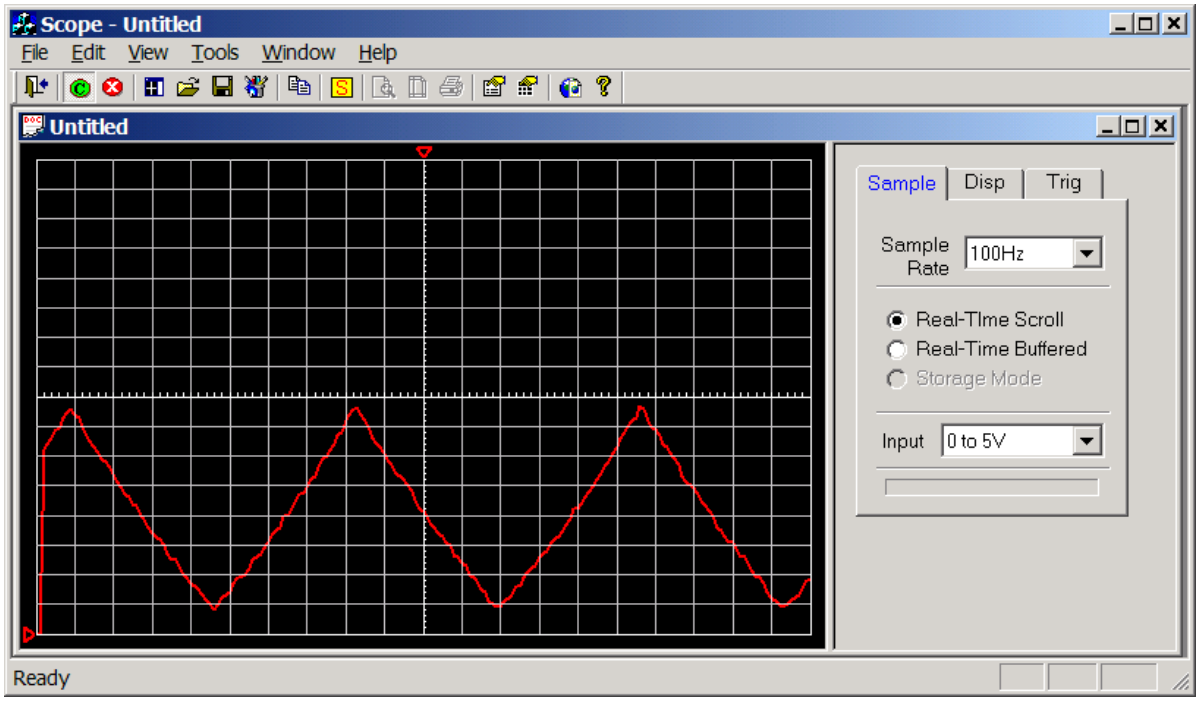


Figure 6.10c. Screen dump of scope program

6.11. Test 10: Test ECG Monitor, CRT & PC Display, Buzzer and BPM Display

Program mark10.c was compiled and loaded into the PIC. This program is designed to display an ECG signal on a CRT display (time compressed memory) and/or a PC (via RS232). The beats per minute of the ECG are displayed on the 7-segment displays and the buzzer beeps for every beat. A signal generator was used to generate a trianglewave because the analogue ECG amplifier was not constructed.

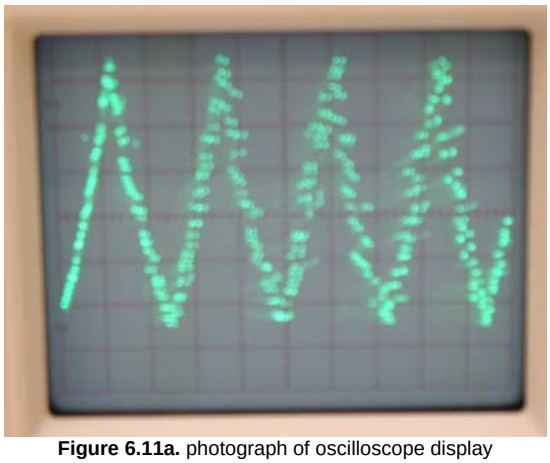


Figure 6.11a. photograph of oscilloscope display

The buzzer successfully beeped on every peak, and the 7-segment display successfully displayed the correct beats per minute of the waveform. It took 10 seconds for the bpm to settle after the frequency of the input waveform was modified, as expected because bpm is calculated over a 10 second period.

When the beats per minute was taken above 250, the buzzer stayed on contentiously until the bpm was reduced. This could be problem as this should only occur when a patient 'flat lines'.

When the signal was completely removed, after ten seconds the bpm reading was 000. But obviously the buzzer was not ON, as the code to turn the buzzer on when bpm equal 0 has not been written yet.

Originally there was a problem displaying the ECG signal on the CRT display, figure 6.11a clearly shows the waveform is distorted. This was due to the time taken to convert the binary bpm value into its key components, originally the following notation was used: -

```
unit = m_bpm % 10;
ten = m_bpm % 100;
ten = (ten – unit) / 10;
hundred = m_bpm * 1000;
hundred = (hundred – ten) /100;
```

This method of conversion was very inefficient as these five lines of C code produced nearly 500 lines of assembly code, that's an execution time of 100µS (500 x 200ns). This method was changed to make the conversion more efficient (the version contained in this report, see sub-chapter 5.10), the C code is much more complicated but the assembly code has been reduced significantly, with the conversion split into multiple steps, with the display being redrawn between steps.

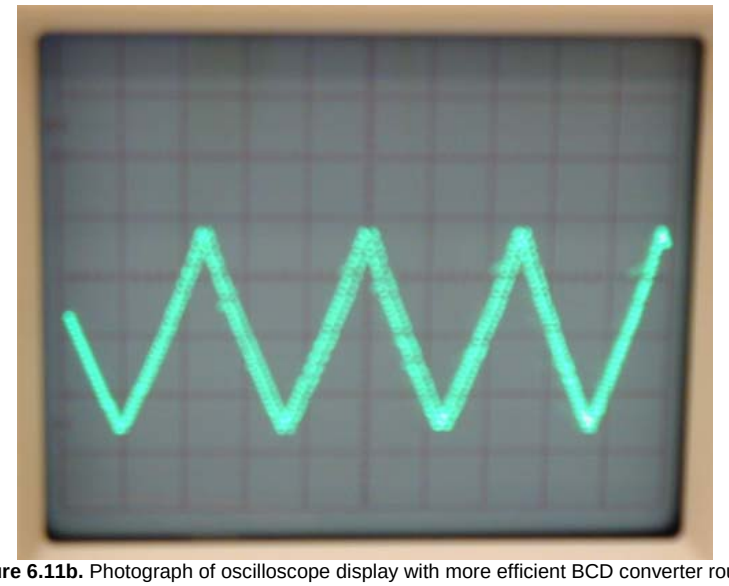


Figure 6.11b. Photograph of oscilloscope display with more efficient BCD converter routine

6.11.1. Measurement of CRT Refresh Rate

The X channel connected to oscilloscope operating in normal mode. Hence the time taken to draw the waveform on the forward and return journey can be measured. Figure 6.11.1 shows a sketch of the oscilloscope display; from this the refresh rate can be calculated.

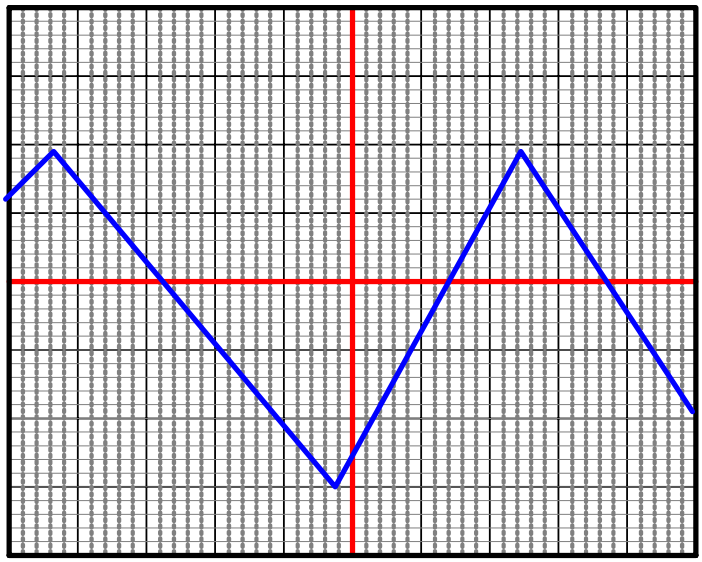


Figure 6.11.1a. Sketched waveform of channel X, timebase = 2ms/div, voltage = 0.5v/div

The downward slope represents the return journey and the upward slope represents the forward journey. Therefore the refresh rate is the frequency of either the upward or downward slope.

Slope time = 3.5 x 2ms = 7ms (142.87Hz Refresh Rate)

Nearly 3 times faster than required (min 50Hz), hence the PIC has a lot of excess processing power available for signal processing.

6.11.2. Operation at Different BPM

50bpm (displayed on 7-segments)

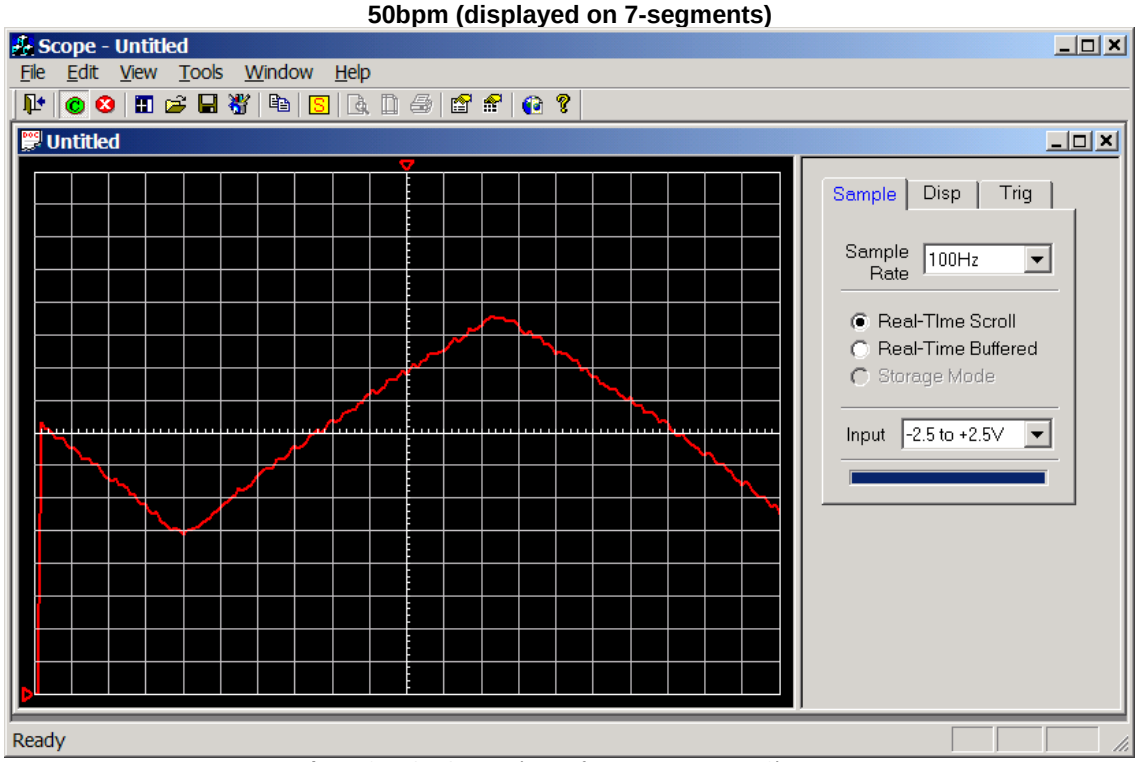


Figure 6.11.2a. Screen dump of scope program at 50bpm

Figure 6.11.2a shows a screen dump of the scope program when the 7-segment displays were showing 50 beats per minute. Note this program is not designed for use with the ECG monitor as the selectable sample rates are limited, ignore the 100Hz, the actual sample rate is 128Hz and the grid resolution is 200 x 160. Each square is made up of 10 readings; hence the time-base of the display is 78.125ms per division.

Calculated BPM: 16 divisions between peaks, that's 16 * 78.125mS = 1.25S (60/1.25 = 48bpm)

72bpm (displayed on 7-segments)

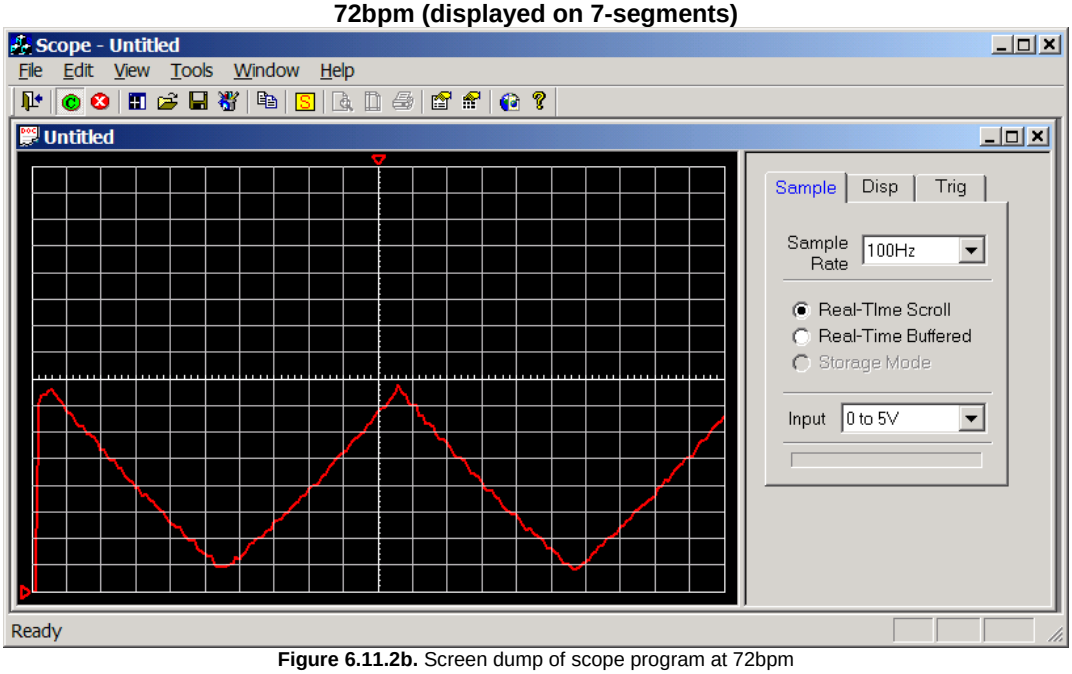


Figure 6.11.2b. Screen dump of scope program at 72bpm

Calculated BPM: 10 divisions between peaks, that's 10 * 78.125mS = 0.78125S (60/0.78 = 76.8bpm)

102bpm (displayed on 7-segments)

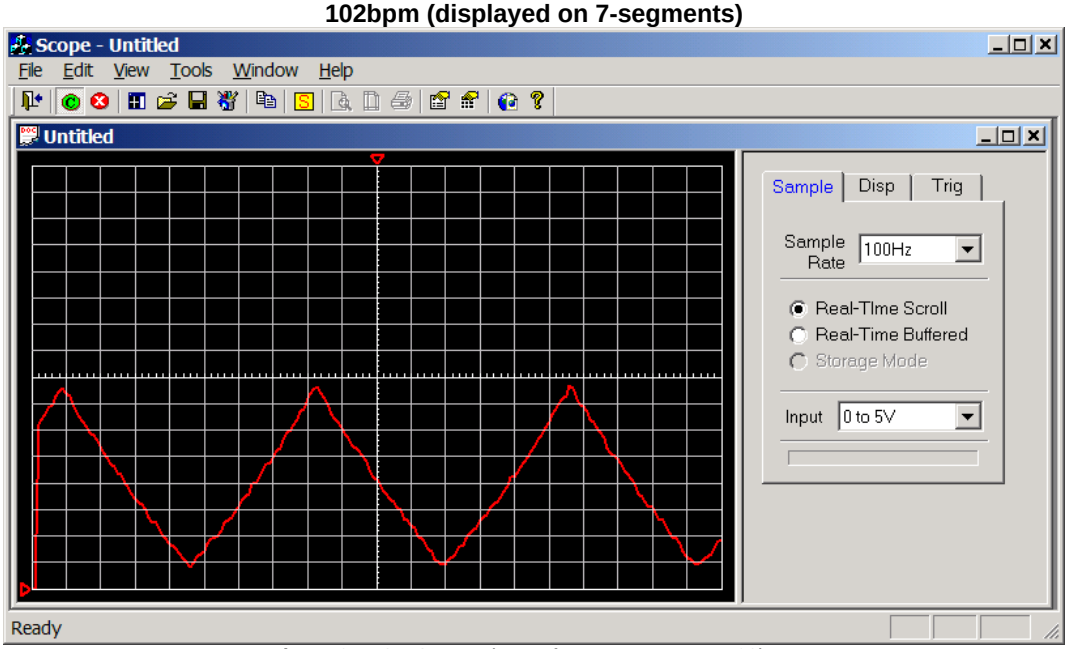


Figure 6.11.2c. Screen dump of scope program at 102bpm

Calculated BPM: 7 divisions between peaks, that's 7 * 78.125mS = 0.547S (60/0.547 = 109bpm)

180bpm (displayed on 7-segments)

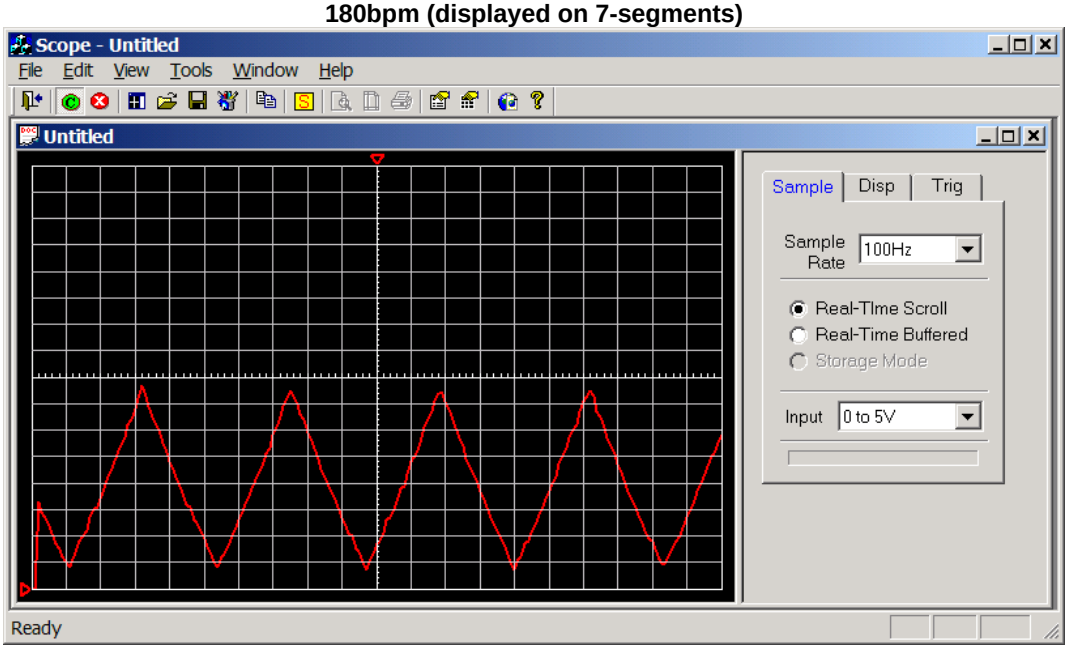


Figure 6.11.2d. Screen dump of scope program at 180bpm

Calculated BPM: 4.2 divisions between peaks, that's 4.2 * 78.125mS = 0.328S (60/0.273 = 182bpm)

240bpm (displayed on 7-segments)

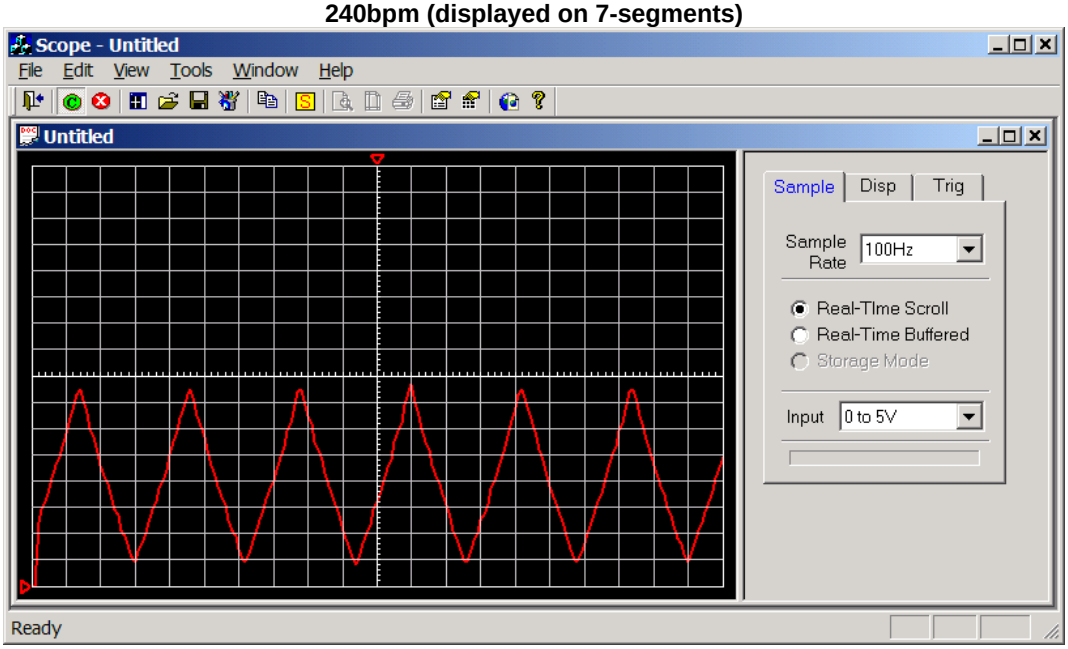


Figure 6.11.2e. Screen dump of scope program at 240bpm

Calculated BPM: 3.2 divisions between peaks, that's 3.2 * 78.125mS = 0.25 (60/0.25 = 240bpm)

014bpm (displayed on 7-segments)

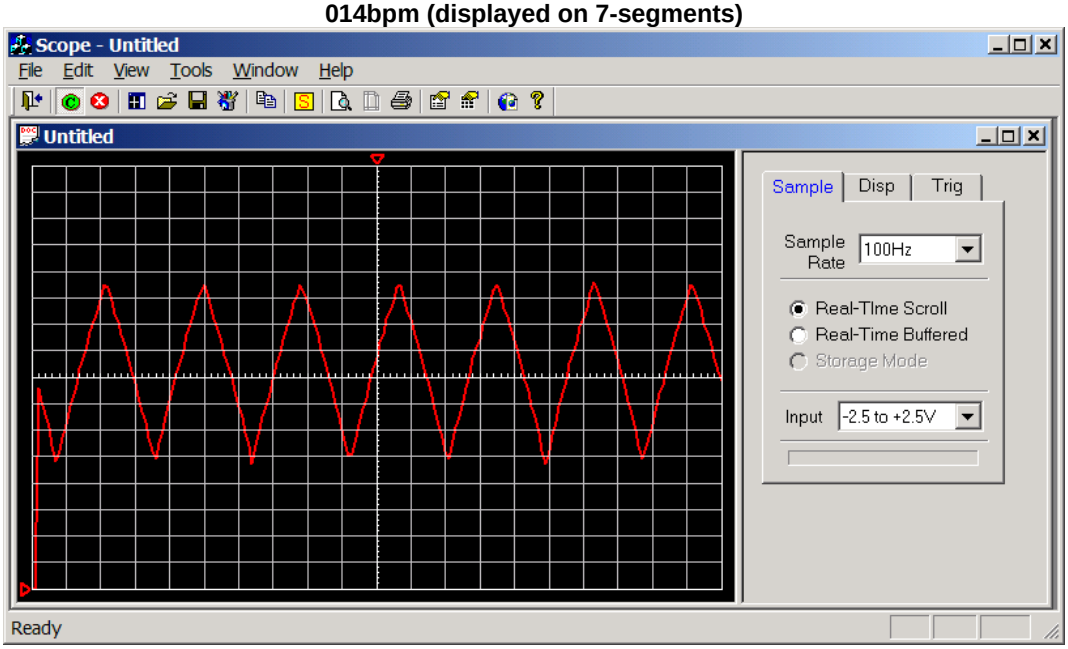


Figure 6.11.2f. Screen dump of scope program at 274bpm

Calculated BPM: 2.8 divisions between peaks, that's 2.8 * 78.125mS = 0.219 (60/0.219 = 274bpm)

7-segments over the top, e.g. recall that calculation of bpm uses an 8-bit integer hence max displayable is 255bpm, hence 255+14 = 269bpm, therefore bpm is actually calculated correctly.

Aliasing started to occur at 64Hz

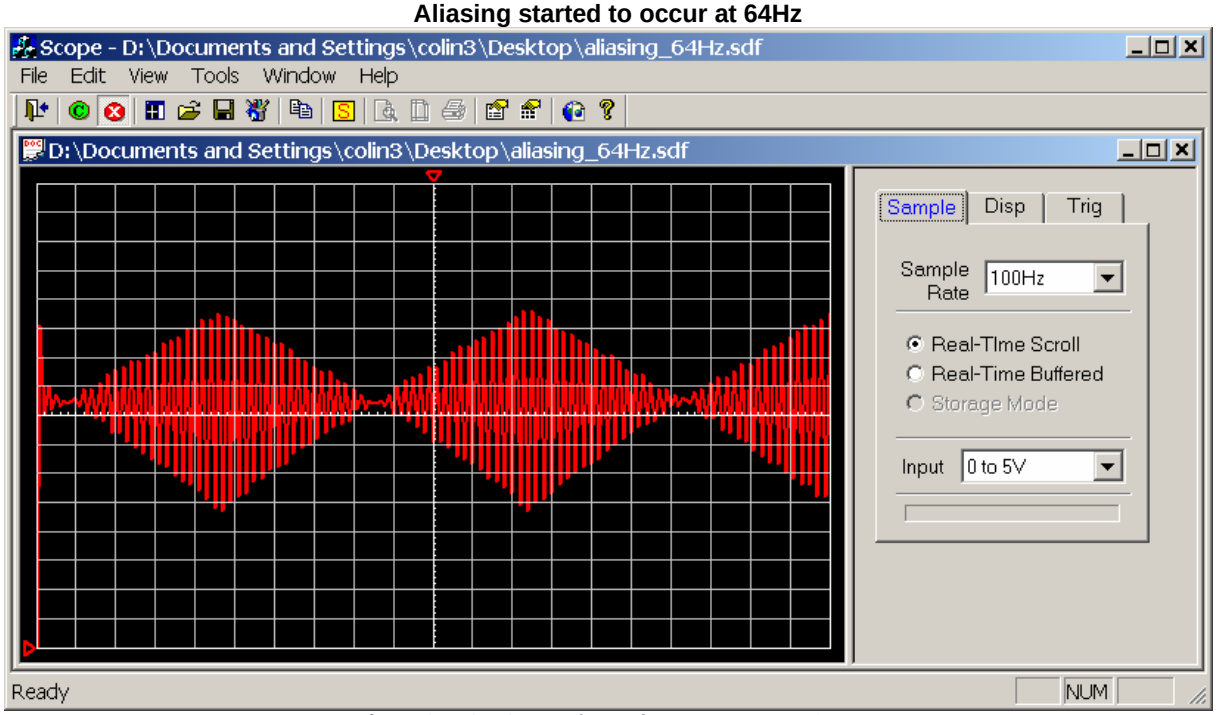


Figure 6.11.2g. Screen dump of scope program at 64Hz

Aliasing started to occur at 64Hz, the CRT display was worse it was doing summersaults.

7.0. CONCLUSIONS

The purpose of this project was to design, construct and test a simple low-cost microcontroller based ECG monitor. Clearly this objective has been achieved successfully, as it has been proven under testing that the design works, although further development is required as there are limitations and flaws in the current design.

Clearly the heart's strong pumping action is driven by powerful waves of electrical activity, which are detected by attaching electrodes to the skin. It is clear that these electrical signals are extremely small and must be amplified considerably (about 1000 times) to be of any use. Evidently an ECG monitor displays these electrical signals graphically just like an oscilloscope displays voltage variations; except that the trace on an ECG monitors scrolls across the screen. The concept of an ECG is not novel one; the attraction of this project lay in the challenge to build a simple, compact, operational medical device at a low cost.

It is clear that the electrocardiogram (ECG) is a simple, non-invasive technique for detecting abnormalities and diagnosing heart defects, merely by noting the presence of irregularities in the PQRST waveform. Clearly other applications are very effective in areas of sports medicine, or sports therapy, in tracking the heartbeat through various levels of physical activity to assist the patient in attaining a desired, optimum heart rate.

The CRT display (Cathode Ray Tube) is one of the common display types in use today (TVs, monitors, oscilloscopes, etc...), clearly this technology is coming to the end of its life as new compact low power technologies like TFT and LCD are becoming more widely used. The main disadvantage of the CRT (besides it high power consumption) is the way it draws the image, the spot is moved across the screen at 50 Hz, even at 50 Hz some people will still see the display flickering badly (modern computer monitors have a refresh rate of over 100Hz), and most people experience problems when an CRT display is within their peripheral vision, causing headaches, dizziness, and eye strain when exposed for long periods of time. The use of a TFT or LCD display instead of a CRT display is clearly a better option, as the screen does not flicker, the reason why it does not flicker is because the screen is split up into pixels, each pixel can be modified independently of each other, hence only the changes made to the display are refreshed.

The RS232 transport medium was chosen to transmit the ECG to a PC in real-time, the main reasons why it was chosen is because it is easy to program, reliable and every PC has at least one RS232 port. The main disadvantage of RS232 is that it is slow (max speed 115kbps) when compared to other mediums (e.g. USB 12Mbps). Because the ECG monitor is only sampling at 128Hz, the speed of the RS232 port is more than enough.

Obviously accuracy, dependability, and precision are an absolute must for the ECG monitor as the device is to be used for diagnostic, and other medical purposes. Clearly any small fluctuation in the waveform generated could carry critical diagnostic value. It is obvious that noise is the main design consideration, as the ECG is extremely small and can easily be masked by noise related fluctuations. The ECG amplifier must amplify the ECG single (1000 times) and not the noise, hence the need for an expensive "instrumentation amplifier" with a high CMRR. This explains why the ECG amplifier is the single most expensive module within the ECG amplifier.

The PIC16F877 microcontroller was chosen; this is one of Microchip's most powerful midrange chips. Clearly this chip was chosen because it has an internal ADC, an internal UART and enough internal RAM to carryout time-compressed memory. Recall that a 50Hz refresh rate is required to ensure that the CRT display is flicker free for most people. Clearly this requires considerable processing power, especially when using a high-level programming language (e.g. C), hence the reason why it was chosen to run the PIC at 20MHz.

The hardware was first proven to work under testing (many small simple test programs), before work began on the main program. Evidently this technique was a success as it allowed for the PIC code to be gradually built up step-by-step, instead of writing the entire program at once. There would have been little chance of the main program working if the entire program was written at once, without the hardware being tested. Clearly this is the case as many problems occurred during development (most of which were hardware based), hence if it was not for the simple test programs, these small problems would have been extremely difficult to track down and may have lead to failure of the project.

A triangle waveform was generated from a signal generator to test the system as the analogue ECG amplifier was not implemented. Test results show that the X and Y outputs of the system successfully controlled the position of the spot on a CRT display, which was moving fast enough so that the waveform did not flicker. It was also noted that the ECG signal (triangle wave in this case) scrolled across the screen as designed, and two seconds of data were displayed on the screen. Clearly there was a problem; the CRT display was showing a slight double image, which appeared to improve as the signal moved across the screen. Evidently this problem is not any fault of the design of the ECG monitor, but the oscilloscope used to represent a CRT display as it had trouble drawing the waveform on the return journey. Obviously the solution to this problem is to purchase a more expensive oscilloscope or to rewrite the software so that the waveform is only drawn on the forward journey then 'fly back' (move the spot off the screen and quickly move it back) to the start position and redraw again (easier to achieve than drawing both directions, but reduced refresh rate, hence increased flicker)

It has been proven that the 7-segment LED displays showed the correct bpm. But recall that originally there was a problem, the CRT display was badly distorted and it was assumed that it was taking to long to calculate the bpm, the program was optimised and this solved the problem; hence it was assumed that this was indeed the reason of the distorted display. But clearly this assumption does not quite add up, as it was proved during testing that the PIC had excess processing power (e.g. near 3-times faster they required, 147Hz refresh rate), hence even if the inefficient program was used the PIC should have been capable of running smoothly. After some reflection, it was recalled that a bug in the program was spotted and fixed at the time the optimisation of program was taking place. This bug was a while loop counting one too many, the array that was used to store the last 10 seconds of ECG data only had 10 locations and the program was writing to an eleventh (oh dear, what eleventh location?). Clearly this is a more likely reason of the distorted image as the PIC would have written the eleventh location somewhere in RAM hence corrupting an important variable.

Clearly the ECG signal was successfully displayed on the PC using the final year project scope program (see appendix 2 for a summary of final year project), but allow the ECG signal was displayed correctly and scrolled across the screen, it was discovered that the voltage level of the waveform was incorrect for example the input triangle wave to the PICs ADC had a peak-peak voltage of 4V, but the scope program waveform had a peak-to-peak voltage of 1V (hence scope displayed image at reduced resolution). Note the scope program has been proven under testing for the final year project that it is actuate, hence something is wrong. Clearly there are two possible reasons; the first is that the PIC ADC is damaged, for example the analogue circuitry was not constructed, hence the PIC had no protection against negative or high voltages therefore it is possible that the signal generator voltage offset was accidentally touched creating a voltage level that damaged the PIC. The second and more likely reason is that when converting the 10-bit ADC reading to 8-bit for the DAC, the original 10-bit variable was also modified (CCS complier fault) to 8-bits, hence only a 8-bit ADC reading was being sent to the scope program, which clearly would reduce waveform resolution and voltage level (e.g. scope assumes reading 1024 = 5V, but for 8-bit 255 is the large possible reading that's 1.25V).

The buzzer worked, but it became clear that there was a problem. The buzzer used in the design was a Piezo alarm which oscillated at an extremely high frequency, with an extremely load volume (e.g. during testing a lecturer complained about the noise, even though the laboratory door was closed and he was in a room half way down the corridor, that's some buzzer!!!). The problem is that there is no way of adjusting the volume level of this buzzer as it has a wide operating voltage range (between 3 and 12 voltages). Clearly the system needs to be resigned so that the volume of the buzzer can be modified; this could be achieved by building a variable audio amplifier and using a speaker as the output. Note the solution to this problem during testing was to push paper into the hole off buzzer.

Obviously some additional development work is required to detect the peaks of a real ECG signal. Clearly the current design would not work, as the maximum amplitude of the ECG signal could vary dramatically, hence most of the peaks may not be detected. This simple method used this design clearly demonstrates the principle, and maked it possible to demonstrate how to calculate the beats per minute.

The Nyquist sampling theorem states that a minimum sample rate of twice the maximum signal is required to represent an analogue signal in digital memory (else aliasing occurs). The system is sampling at 128Hz, a low pass filter should be designed to cut off analogue frequencies above 64Hz (128/2), hence the system has an analogue bandwidth of 64Hz as the low frequency cut off is 0Hz (i.e. ECG flat lined, DC signal).

Clearly a dedicated ECG application is required for the PC, this application should be able to display the ECG signal in real-time, scream real-time data to disk, and could have internet TCP/IP communications so that a doctor can remotely view the ECG single in real-time from anywhere in the world.

The push button successfully paused the ECG display, clearly it makes sense to take this one stage further and have two ECG outputs, having the option of pausing one while the other continues to move. Clearly this is currently possible (crude technique) as the PC display can be paused while the CRT display continues to scroll, this was achieved by simply disabling RS232 communications on the scope program, hence no new data was received and the waveform is paused.

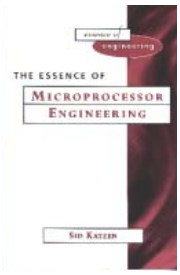
It is clear that the PIC is producing a refresh rate nearly 3-time faster (142 Hz) than minimally required (50 Hz), hence to reduce power consumption (useful if using battery power supply), and reduce ADC noise, the PIC can be ran at a slower frequency, for example 10 MHz should produce a refresh rate above 70Hz.

Evidently the project was a success, a simple low-cost microcontroller based ECG monitor was design, constructed and proven to work. The system successfully displayed an ECG signal (trianglewave) on a CRT display, transmitted the ECG signal in real-time to the PC, correctly displayed bpm on the 7-segment displays, the push button paused the display and the buzzer beeped on every ECG peak.

8.0. REFERENCES / BIBLIOGRAPHY

Text Books

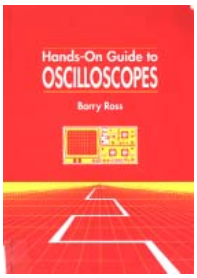
[B1]



The Essences of Microprocessor Engineering

By Sid Katzen
Publisher: Prentice Hall
ISBN: 0-13-244708-8
1998

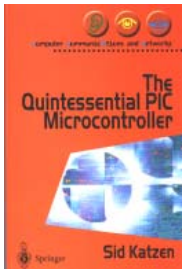
[B2]



Hands-On Guide to Oscilloscopes

By Barry Ross
Publisher: McGraw – Hill
ISBN: 0-07-707818-7
UUJLIB:621.3815483ROS
1994

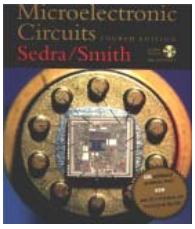
[B3]



The Quintessential PIC Microcontroller

by Dr Sid Katzen
published by: Springer
ISBN: 1-85233- 309-X.
2001

[B4]



Microelectronic Circuits – Fourth Edition.

By S. Sedra and Kenneth C. Smith
Publisher: Oxford university press
ISBN: 0-19-511690-9
1998

Websites

[W1] http://www.laurushealth.com

[W2] http://baserv.uci.kun.nl/~smientki/Lego_Knex/Lego_electronica/BioSensors/ECG_sensor.htm (Mindstorms ECG Sensor by Stef Mientki, august 2001)

[W3] http://www.oucom.ohiou.edu/CVPhysiology/A013.htm (An overview of the different standard electrode placements is given by Richard E. Klabunde, Ph.D).

[W4] http://www.oucom.ohiou.edu/CVPhysiology/A009.htm (An overview of electrocardiogram by Richard E. Klabunde, Ph.D).

[W5] http://www.consultrsr.com/resources/agcl.htm (The Ag-AgCl reference electrode, © Copyright 2000 research solutions and resources).

[W6] http://www.healthsci.utas.edu.au/physiol/tute2/rm11.html (The Electrical Conduction System of the Heart)

[W7] http://www.microchip.com - Microchip Website (PIC datasheets and application notes).

[W8] http://www.farnel.com/uk (Farnell Online catalogue).

[W9] http://rswww.com (RS Online Catalogue).

[W10] http://www.engj.ulst.ac.uk/sidk/PIC/index.html - PIC Resource Site (An assorted set of data sheets, instruction sets and web sites, by Dr S Katzen).

[W11] http://www.ecglibrary.com/ecghist.html (A not so brief history of electrocardiography).

[W12] http://www.analog.com (Analog Devices official website)

A1. PIC SOURCE CODE

A1.1. mark1.c

```
/* -------------------------------------------------------
   | FILE    : mark1.c                                    |
   | PROJECT : Real-Time PIC Based ECG Monitor            |
   | DESC    : Test RS232                                 |
   | ================================================== |
   | DATE    : 19/03/2002                                 |
   | BY      : Colin K McCord                             |
   | VERSION : 1.0                                        |
   ------------------------------------------------------- */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock = 20000000)  // 20MHz clock, change this value if using different clock speed.
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)


main()
{

        while(TRUE)
        {
                /* Note prinf could be used but this function is wasteful and will not be used */
                putc('T');     // Transmit T
                putc('e');     // Transmit e
                putc('s');     // Transmit s
                putc('t');     // Transmit t
                putc('i');     // Transmit i
                putc('n');     // Transmit n
                putc('g');     // Transmit g
                putc('.');     // Transmit .
                putc('.');     // Transmit .
                putc('.');     // Transmit .

                delay_ms(1000);        // Present delay, repeat every second
        }
}
```

A1.2. mark2.c

```
/* -------------------------------------------------------
   | FILE    : mark2.c                                    |
   | PROJECT : Real-Time PIC Based ECG Monitor            |
   | DESC    : Read ADC, transmit result through          |
   |         : RS232 using the final year project         |
   |         : (PC based Oscilloscope) real-time frame    |
   |         : format.                                    |
   | ================================================== |
   | DATE    : 19/03/2002                                 |
   | BY      : Colin K McCord                             |
   | VERSION : 1.1                                        |
   ------------------------------------------------------- */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.


#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock = 4000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)
```

```
main()
{
        // NOTE: by default in CCS all var's are unsigned
        long int adcValue;            // 16-bit storage for ADC reading
        char adcHI,adcLO;             // 8-bit storage for real-time frames.

        setup_adc_ports(RA0_ANALOG);  // RA0  Analogue, rest digital
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);

        delay_us(20);  // Delay for sampling cap to charge

        while(TRUE)
        {
                adcValue = read_adc(); // Get ADC reading

                /* Convert 16-bit adcValue to Real-time frame structure, CH1 */
                adcHI = (char)((adcValue >> 5)& 0x1f);       // 0|0|0|d9|d8|d7|d6|d5
                adcLO = (char)((adcValue & 0x1f)|0x80);      // 1|0|0|d4|d3|d2|d1|d0

                putc(adcHI);   // Transmit Byte 1 (d9...d5)
                putc(adcLO);   // Transmit Byte 2 (d4...d0)

                delay_ms(8);   // Preset delay, repeat every 8ms, that's a sample rate of 125Hz.
        }
}
```

A1.3. mark3.c

```
/* -------------------------------------------------------
   | FILE    : mark3.c                                    |
   | PROJECT : Real-Time PIC Based ECG Monitor            |
   | DESC    : Test RS232, baud rate set by dip switches. |
   | ================================================== |
   | DATE    : 01/04/2002                                 |
   | BY      : Colin K McCord                             |
   | VERSION : 1.0                                        |
   ------------------------------------------------------- */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

#fuses HS,NOWDT,NOPROTECT,NOLVP

#byte   PORTE = 0x09   // PortE lives in File 9

/* Note 20MHz clock must be used for 115,000 bps, the % error is to large at slower speeds */
#use delay(clock = 20000000)  // 20MHz clock, change this value if using different clock speed.
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)


#use    fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

/* Forward declaration of functions */
void SetBaudRate();


main()
{
        set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 TTL Inputs

        SetBaudRate();

        while(TRUE)
        {
                /* Note prinf could be used but this function is wasteful and will not be used */
                putc('T');     // Transmit T
                putc('e');     // Transmit e
                putc('s');     // Transmit s
                putc('t');     // Transmit t
                putc('i');     // Transmit i
                putc('n');     // Transmit n
```

```
                putc('g');     // Transmit g
                putc('.');     // Transmit .
                putc('.');     // Transmit .
                putc('.');     // Transmit .

                delay_ms(1000);       // Present delay, repeat every second
        }
}

void SetBaudRate()
{
        switch(PORTE & 0x07)  // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);   break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }
}
```

A1.4. mark4.c

```
/* -------------------------------------------------------
   | FILE    : mark4.c                                   |
   | PROJECT : Real-Time PIC Based ECG Monitor           |
   | DESC    : Read ADC, transmit result through         |
   |         : RS232 using the final year project        |
   |         : (PC based Oscilloscope) real-time frame   |
   |         : format. Baud rate set by dip-switches.    |
   | ================================================== |
   | DATE    : 01/04/2002                                |
   | BY      : Colin K McCord                            |
   | VERSION : 1.2                                       |
   ------------------------------------------------------- */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.


#fuses HS,NOWDT,NOPROTECT,NOLVP

#byte   PORTE = 0x09   // PortE lives in File 9

/* Note 20MHz clock must be used for 115,000 bps, the % error is to large at slower speeds */
#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)

#use   fast_io(E)            // Fast access to PortE (don't fiddle with TRISE)

/* Forward declaration of functions */
void SetBaudRate();

main()
{
        // NOTE: by default in CCS all var's are unsigned
        long int adcValue;            // 16-bit storage for ADC reading
        char adcHI,adcLO;             // 8-bit storage for real-time frames.

        set_tris_e(0x17);             // TRISE = 00010111; RE2,RE1 and RE0 TTL Inputs
        SetBaudRate();

        setup_adc_ports(RA0_ANALOG);  // RA0  Analogue, rest digital
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);

        delay_us(20);  // Delay for sampling cap to charge
```

```
        while(TRUE)
        {
                adcValue = read_adc(); // Get ADC reading

                /* Convert 16-bit adcValue to Real-time frame structure, CH1 */
                adcHI = (char)((adcValue >> 5)& 0x1f);       // 0|0|0|d9|d8|d7|d6|d5
                adcLO = (char)((adcValue & 0x1f)|0x80);      // 1|0|0|d4|d3|d2|d1|d0

                putc(adcHI);   // Transmit Byte 1 (d9...d5)
                putc(adcLO);   // Transmit Byte 2 (d4...d0)

                delay_ms(8);   // Preset delay, repeat every 8ms, thats 125 Hz
        }
}

void SetBaudRate()
{
        switch(PORTE & 0x07)  // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }
}
```

A1.5. mark5.c

```
/* -------------------------------------------------------
   | FILE    : Mark 5.c                                  |
   | PROJECT : Mini Project: ECG                         |
   | DESC    : Test 7-Seg display and buzzer             |
   | ================================================== |
   | DATE    : 14/04/2002                                |
   | BY      : Colin K McCord                            |
   | VERSION : 1.2                                       |
   ------------------------------------------------------- */

#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.

#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT

#byte   PORTA = 0x05          // PortA lives in File 5 (ADC input and Control Lines)
#byte   DATA_BUS = 0x06       // PortB lives in File 6 (data bus)
#byte   PORTC = 0x07          // PortC lives in File 7 (UART and Control Lines)
#byte   ADDRESS_BUS = 0x08    // PortD lives in File 8 (address bus A0 to A7)
#byte   PORTE = 0x09          // PortE lives in File 9 (used for DIP switches)

/** Setup control lines with user-friendly names */
#bit    DAC_ENABLE = PORTA.2  // C0
#bit    DAC_SELECT = PORTA.3  // C1
#bit    RAM_OE     = PORTA.4  // C2
#bit    RAM_WRITE  = PORTA.5  // C3
#bit    BUZZER     = PORTC.0  // C4
#bit    SEG1_EN    = PORTC.1  // C5
#bit    SEG_CLOCK  = PORTC.2  // C6
#bit    SEG2_EN    = PORTC.5  // C7
#bit    SEG3_EN    = PORTC.4  // C8

#bit    PUSH_BUTT  = PORTA.1

#byte   PORTE = 0x09   // PortE lives in File 9

#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)
```

```
#use   fast_io(A)     // Fast access to PortA (don't fiddle with TRISA)
#use   fast_io(B)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(C)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(D)     // Fast access to PortD (don't fiddle with TRISD)
#use   fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

/** Seven Segment display lookup table */
const char SevenSeg[10] ={0b01000000, // 0
                          0b01111001,  // 1
                          0b00100100,  // 2
                          0b00110000,  // 3
                          0b00011001,  // 4
                          0b00010010,  // 5
                          0b00000010,  // 6
                          0b01111000,  // 7
                          0b00000000,  // 8
                          0b00010000}; // 9


/* Forward declaration of functions */
void SetBaudRate();

main()
{
       int b1=0,b2=0,b3=0;

       set_tris_a(0x02);             // TRISA = 00000011; RA1 to RA7 TTL Outputs
       set_tris_c(0xC0);             // TRISC = 11001000; RC0 to RC2, RC4,RC5 TTL Outputs (bits 7&6
                                     // must be set for UART to work)
       set_tris_d(0x00);             // TRISD = 00000000; RD0 to RD7 TTL Outputs
       port_b_pullups(FALSE);        // Don't use internal pull up resistors

       SetBaudRate();

       set_tris_b(0x00); // Data bus output
       while(true)
       {
              DAC_ENABLE = 1;        //Disable DAC
              DAC_SELECT = 0;

              RAM_OE     = 1; // Disable RAM
              RAM_WRITE  = 1;

              if (b2 == 0 && b1 == 0)
              {
                     BUZZER = 1;    // Buzzer ON
              }
              else
              {
                     BUZZER = 0;    // Buzzer OFF
              }


              // ENABLE RIGHT DISPLAY
              SEG1_EN    = 1;
              SEG2_EN    = 1;
              SEG3_EN    = 0;

              DATA_BUS   = SevenSeg[b1++];
              if(b1 == 10)
              {
                     b1 = 0;
                     b2++;
              }
              if(b2 == 10)
              {
                     b2 = 0;
                     b3++;
              }

              if(b3 == 10)
              {
                     b3 = 0;
              }

              SEG_CLOCK = 0;
```

```
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;


                // Enable MIDDLE DISPLAY
                SEG1_EN    = 1;
                SEG2_EN    = 0;
                SEG3_EN    = 1;

                DATA_BUS   = SevenSeg[b2];

                SEG_CLOCK = 0;
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;

                // ENABLE LEFT DISPLAY
                SEG1_EN    = 0;
                SEG2_EN    = 1;
                SEG3_EN    = 1;

                DATA_BUS   = SevenSeg[b3];;

                SEG_CLOCK = 0;
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;


                SEG1_EN = 1;   // Disable display

                delay_ms(250);

        }

}

void SetBaudRate()
{
        // Verison 1.1, 22/3/2002, by CKM

        set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 Inputs
                               // Parallel slave mode is ON, e.g. TTL Inputs, on Port E and D

        switch(PORTE & 0x07)   // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }

        set_tris_e(0x07);  // Turn OFF Parallel Slave Mode as this affects port D (Address Bus)

}
```

A1.6. mark6.c

```
/* -------------------------------------------------------
   | FILE    : mark6.c                                    |
   | PROJECT : Mini Project: ECG                          |
   | DESC    : Test timer interrupts                      |
   | ==================================================== |
   | DATE    : 05/04/2002                                 |
   | BY      : Colin K McCord                             |
   | VERSION : 1.1                                        |
   ------------------------------------------------------- */
```

```
#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.


#fuses HS,NOWDT,NOPROTECT,NOLVP

#byte   PORTE = 0x09   // PortE lives in File 9

/* Note 20MHz clock must be used for 115,000 bps, the % error is to large at slower speeds */
#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)

#use   fast_io(E)            // Fast access to PortE (don't fiddle with TRISE)

#define T0_INTS_PER_SEC 76    // (20,000,000/(4*256*256))
#define T1_INTS_PER_SEC 76    // (20,000,000/(4*1*65536))

byte int_count0;              // Number of T0 interrupts left before a 0.5s has elapsed.
byte int_count1;              // Number of T1 interrupts left before a 0.25s has elapsed.
byte int_count2;              // Number of T2 interrupts left before a 100 msec has elapsed.

#int_rtcc             // RTCC (timer0) interrupt, called every time RTCC overflows (255->0)
Timer0_ISR()
{
       if(--int_count0==0)
       {
              printf("Interrupt_T0...");            // Every 0.5 seconds.
              int_count0 = T0_INTS_PER_SEC/2;
       }
}

#INT_TIMER1           // timer1 interrupt, called every time timer1 overflows (65536->0)
Timer1_ISR()
{
       if(--int_count1==0)
       {
              printf("Interrupt_T1...");            // Every 0.25 seconds.
              int_count1 = T1_INTS_PER_SEC/4;
       }
}

#INT_TIMER2           // timer2 interrupt
Timer2_ISR()
{
       if(--int_count2==0)
       {
              printf("Interrupt_T2...");            // Every 0.1 seconds.
              int_count2 = 100;
       }
}


/* Forward declaration of functions */
void SetBaudRate();

main()
{
       set_tris_e(0x17);             // TRISE = 00010111; RE2,RE1 and RE0 TTL Inputs
       SetBaudRate();

       /** Setup timer0 (RTCC) **/
       int_count0 = T0_INTS_PER_SEC/2;              // 0.5 seconds
       set_rtcc(0);
       setup_counters(RTCC_INTERNAL,RTCC_DIV_256);
       enable_interrupts(RTCC_ZERO);

       /** Setup timer1 **/
       int_count1 = T1_INTS_PER_SEC/4;              //0.25 second
       setup_timer_1(T1_DIV_BY_1 | T1_INTERNAL);
       set_timer1(0);
       enable_interrupts(INT_TIMER1);


       /** Setup timer2 **/
```

```
        int_count2 = 100;                       // 0.1 second
        setup_timer_2 (T2_DIV_BY_4,125,9);      // interrupt every 1ms
        set_timer2(0);
        enable_interrupts(INT_TIMER2);

        enable_interrupts(GLOBAL);

        while(TRUE)
        {
                printf("Main...");
                delay_ms(1000);                 // 1 second delay
        }
}

void SetBaudRate()
{
        switch(PORTE & 0x07)   // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }
}
```

A1.7. mark7.c

```
/* ------------------------------------------------------
   | FILE    : mark7.c                                  |
   | PROJECT : Mini Project: ECG                        |
   | DESC    : Test Dual DAC                            |
   | ================================================== |
   | DATE    : 14/04/2002                               |
   | BY      : Colin K McCord                           |
   | VERSION : 1.1                                      |
   ------------------------------------------------------ */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.


#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT

#byte   PORTA = 0x05          // PortA lives in File 5 (ADC input and Control Lines)
#byte   DATA_BUS = 0x06       // PortB lives in File 6 (data bus)
#byte   PORTC = 0x07          // PortC lives in File 7 (UART and Control Lines)
#byte   ADDRESS_BUS = 0x08    // PortD lives in File 8 (address bus A0 to A7)
#byte   PORTE = 0x09          // PortE lives in File 9 (used for DIP switches)

/** Setup control lines with userfriendly names */
#bit    DAC_ENABLE = PORTA.2  // C0
#bit    DAC_SELECT = PORTA.3  // C1
#bit    RAM_OE     = PORTA.4  // C2
#bit    RAM_WRITE  = PORTA.5  // C3
#bit    BUZZER     = PORTC.0  // C4
#bit    SEG1_EN    = PORTC.1  // C5
#bit    SEG_CLOCK  = PORTC.2  // C6
#bit    SEG2_EN    = PORTC.5  // C7
#bit    SEG3_EN    = PORTC.4  // C8

#bit    PUSH_BUTT  = PORTA.1

#byte   PORTE = 0x09   // PortE lives in File 9

#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)
```

```
#use   fast_io(A)     // Fast access to PortA (don't fiddle with TRISA)
#use   fast_io(B)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(C)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(D)     // Fast access to PortD (don't fiddle with TRISD)
#use   fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

/** Seven Seg display lookup table */
const char SevenSeg[10] ={0b01000000, // 0
                          0b01111001,  // 1
                          0b00100100,  // 2
                          0b00110000,  // 3
                          0b00011001,  // 4
                          0b00010010,  // 5
                          0b00000010,  // 6
                          0b01111000,  // 7
                          0b00000000,  // 8
                          0b00010000}; // 9

// Global Variables
char block1[64];      // Address 0 to 63
char block2[64];      // Address 64 to 127
char block3[64];      // Address 128 to 191
char block4[64];      // Address 192 to 255
char pStartOfArray = 0; // Incoming Sampled Character stored here then inc the pointer
char pScanPOS =0;

/* Forward declaration of functions */
void SetBaudRate();
int  ReadArray(char address);
void WriteArray(char address, char data);
void WriteToDAC(char address, char data);
void SetDAC(char X, char Y);


main()
{
       char DAC_X = 0,DAC_Y=0xFF;
       set_tris_a(0x02);      // TRISA = 00000011; RA2 to RA7 TTL Outputs
       set_tris_c(0xC0);      // TRISC = 11001000; RC0 to RC2, RC4,RC5 TTL Outputs (bits 7&6 must be
                              // set for UART to work)
       set_tris_d(0x00);      // TRISD = 00000000; RD0 to RD7 TTL Outputs
       port_b_pullups(FALSE); // Don't use internal pull up resistors

       RAM_OE     = 1;        // Disable RAM
       RAM_WRITE  = 1;

       SEG1_EN    = 1;        // Disable SEG 1
       SEG2_EN    = 1;        // Disable SEG 2
       SEG3_EN    = 1;        // Disable SEG 3

       DAC_ENABLE = 1;        // Disable DAC
       DAC_SELECT = 0;

       SetBaudRate();

       while(true)
       {
              SetDAC(DAC_X, DAC_Y);  // Send data to DAC
              DAC_X++;
              DAC_Y--;

              delay_us(78); // 50 x 255 = 12,750Hz for 50Hz display (approximately 78uS)
       }
}


void SetBaudRate()
{
       // Verison 1.1, 22/3/2002, by CKM

       set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 Inputs
                              // Parallel slave mode is ON, e.g. TTL Inputs, on Port E and D

       switch(PORTE & 0x07)   // Read dip switches and setup baud rate
       {
              case 0: set_uart_speed(4800);   break;
```

```
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }

        set_tris_e(0x07);  // Turn OFF Parallel Slave Mode as this affects port D (Address Bus)

}

void SetDAC(char X, char Y)
{
        // by CKM 14/04/2002
        set_tris_b(0x00); // Data bus output

        DATA_BUS = X;
        DAC_SELECT = 0;        // Selected DAC A
        DAC_ENABLE = 0;        // Enable DAC

        delay_cycles(1);       // 100nS delay

        DAC_ENABLE = 1;        // Disable DAC

        delay_cycles(1);
        DATA_BUS = Y;
        DAC_SELECT = 1;        // Select DAC B
        DAC_ENABLE = 0;        // Enable DAC

        delay_cycles(1);       // 100nS delay
        DAC_ENABLE = 1;        // Disable DAC
}
```

A1.8. mark8.c

```
/* -------------------------------------------------------
   | FILE    : mark8.c                                   |
   | PROJECT : Mini Project: ECG                         |
   | DESC    : Test RAM Chip                             |
   | ================================================== |
   | DATE    : 14/04/2002                                |
   | BY      : Colin K McCord                            |
   | VERSION : 1.3                                       |
   ------------------------------------------------------- */

#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.

#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT

#byte   PORTA = 0x05          // PortA lives in File 5 (ADC input and Control Lines)
#byte   DATA_BUS = 0x06       // PortB lives in File 6 (data bus)
#byte   PORTC = 0x07          // PortC lives in File 7 (UART and Control Lines)
#byte   ADDRESS_BUS = 0x08    // PortD lives in File 8 (address bus A0 to A7)
#byte   PORTE = 0x09          // PortE lives in File 9 (used for DIP switches)

/** Setup control lines with user-friendly names */
#bit    DAC_ENABLE = PORTA.2  // C0
#bit    DAC_SELECT = PORTA.3  // C1
#bit    RAM_OE     = PORTA.4  // C2
#bit    RAM_WRITE  = PORTA.5  // C3
#bit    BUZZER     = PORTC.0  // C4
#bit    SEG1_EN    = PORTC.1  // C5
#bit    SEG_CLOCK  = PORTC.2  // C6
#bit    SEG2_EN    = PORTC.5  // C7
#bit    SEG3_EN    = PORTC.4  // C8

#bit    PUSH_BUTT  = PORTA.1
```

```
#byte   PORTE = 0x09   // PortE lives in File 9

#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)

#use   fast_io(A)     // Fast access to PortA (don't fiddle with TRISA)
#use   fast_io(B)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(C)     // Fast access to PortB (don't fiddle with TRISB)
#use   fast_io(D)     // Fast access to PortD (don't fiddle with TRISD)
#use   fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

/* Forward declaration of functions */
void SetBaudRate();
void WriteRAM(char address, char data);
char ReadRAM(char address);

main()
{
        long int i;
        char pass, fail;
        set_tris_a(0x02);      // TRISA = 00000011; RA2 to RA7 TTL Outputs
        set_tris_c(0xC0);      // TRISC = 11001000; RC0 to RC2, RC4, RC5 TTL Outputs (bits 7&6 must
                               // be set for UART to work)
        set_tris_d(0x00);      // TRISD = 00000000; RD0 to RD7 TTL Outputs
        port_b_pullups(FALSE); // Don't use internal pull up resistors

        SetBaudRate();

        while(TRUE)
        {
                delay_ms(5000); // 5 Sec delay

                printf("TEST RAM CHIP...");

                /** RAM Test 1 write 0xFF into all locations and check **/
                printf("\n\nTest 1 - Fill RAM with 0xFF --->");
                pass = 0;      // Reset Pass counter
                fail = 0;      // Reset Fail counter

                for (i=0;i<256;i++)    // Write 0xFF to all locations
                {
                        WriteRAM((char)i,0xFF);
                }

                for (i=0;i<256;i++)
                {
                        if(ReadRAM((char)i) == 0xFF) pass++;
                        else fail++;
                }

                printf(" Passes = %u, Fails = %u", pass,fail);


                /** RAM Test 2 write 0x00 into all locations and check **/
                printf("\n\nTest 2 - Fill RAM with 0x00 --->");
                pass = 0;      // Reset Pass counter
                fail = 0;      // Reset Fail counter

                for (i=0;i<256;i++)    // Write 0x00 to all locations
                {
                        WriteRAM((char)i,0x00);
                }

                for (i=0;i<256;i++)
                {
                        if(ReadRAM((char)i) == 0x00) pass++;
                        else fail++;
                }

                printf(" Passes = %u, Fails = %u", pass,fail);


                /** RAM Test 3  address into all locations and check **/
                printf("\n\nTest 3 - Fill RAM with Address --->");
                pass = 0;      // Reset Pass counter
                fail = 0;      // Reset Fail counter
```

```
                for (i=0;i<256;i++)    // Write address to all locations
                {
                        WriteRAM((char)i,(char)i);
                }

                for (i=0;i<256;i++)
                {
                        if(ReadRAM((char)i) == (char)i) pass++;
                        else fail++;
                }

                printf(" Passes = %u, Fails = %u", pass,fail);

                printf("\n\nEnd of RAMTEST\n\n");

        }
        sleep();
}


void SetBaudRate()
{
        // Verison 1.1, 22/3/2002, by CKM

        set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 Inputs
                               // Parallel slave mode is ON, e.g. TTL Inputs, on Port E and D

        switch(PORTE & 0x07)   // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }

        set_tris_e(0x07);  // Turn OFF Parallel Slave Mode as this affects port D (Address Bus)
}

void WriteRAM(char address, char data)
{
        // 20/03/2002, version 1.0 by Colin McCord

        RAM_OE     = 0;        // Enable RAM Chip

        set_tris_b(0x00);      // Databus set for output

        RAM_WRITE = 1;         // Don't Write until address and data bus is setup.

        // Set Address
        ADDRESS_BUS = address; // Put address on the address bus
        DATA_BUS = data;       // Put data on the data bus


        delay_cycles(1);       // 200nS delay, same as NOP, (1/20,000,000)*4*1) = 200nS
                               // make sure address & data is valid before write

        RAM_WRITE = 0;         // Write data to specified address

        delay_cycles(1);       // 200nS delay (same as NOP), make sure write is finished

        RAM_WRITE = 1;         // Back to Read Mode
        RAM_OE    = 1;         // Disable RAM Chip
}

char ReadRAM(char address)
{
        // 20/03/2002, version 1.0 by Colin McCord
        char m_data;

        set_tris_b(0xFF);      // Put databus in read mode

        RAM_OE     = 0;        // Enable RAM Chip
        RAM_WRITE = 1;         // Put RAM chip in Read Mode
```

```
        // Set Address
        ADDRESS_BUS = address; // Put address on address bus

        delay_cycles(1);       // 200nS delay, same as NOP, (1/20,000,000)*4*1) = 200nS
                               // make sure data is valid before reading
        m_data = DATA_BUS;     // Get data from the data bus
        RAM_OE   = 1;          // Disable RAM Chip

        return m_data;
}
```

A1.9. mark9.c

```
/* -------------------------------------------------------
   | FILE    : mark9.c                                    |
   | PROJECT : Mini Project: ECG                          |
   | DESC    : CRT ECG NONITOR using Interrupt Sampling.  |
   |         : ECG output to DAC, drawn bidirectional.    |
   |         : ECG also outputted through RS232 using     |
   |         : final year project real-time frame         |
   |         : structure.                                 |
   | ================================================== |
   | DATE    : 22/04/2002                                 |
   | BY      : Colin K McCord                             |
   | VERSION : 1.4                                        |
   ------------------------------------------------------- */


#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.

#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT

#byte   PORTA = 0x05          // PortA lives in File 5 (ADC input and Control Lines)
#byte    DATA_BUS = 0x06      // PortB lives in File 6 (data bus)
#byte    PORTC = 0x07         // PortC lives in File 7 (UART and Control Lines)
#byte   ADDRESS_BUS = 0x08    // PortD lives in File 8 (address bus A0 to A7)
#byte   PORTE = 0x09          // PortE lives in File 9 (used for DIP switches)

/** Setup control lines with userfriendly names */
#bit    DAC_ENABLE = PORTA.2  // C0
#bit    DAC_SELECT = PORTA.3  // C1
#bit    RAM_OE     = PORTA.4  // C2
#bit    RAM_WRITE  = PORTA.5  // C3
#bit    BUZZER     = PORTC.0  // C4
#bit    SEG1_EN    = PORTC.1  // C5
#bit    SEG_CLOCK  = PORTC.2  // C6
#bit    SEG2_EN    = PORTC.5  // C7
#bit     SEG3_EN    = PORTC.4  // C8

#bit    PUSH_BUTT  = PORTA.1

#byte   PORTE = 0x09   // PortE lives in File 9

#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)

#use    fast_io(A)     // Fast access to PortA (don't fiddle with TRISA)
#use    fast_io(B)     // Fast access to PortB (don't fiddle with TRISB)
#use    fast_io(C)     // Fast access to PortB (don't fiddle with TRISB)
#use    fast_io(D)     // Fast access to PortD (don't fiddle with TRISD)
#use    fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

/** Seven Seg display lookup table */
const char SevenSeg[10] ={0b01000000, // 0
                          0b01111001,  // 1
                          0b00100100,  // 2
                          0b00110000,  // 3
```

```
                        0b00011001,  // 4
                        0b00010010,  // 5
                        0b00000010,  // 6
                        0b01111000,  // 7
                        0b00000000,  // 8
                        0b00010000}; // 9

// Global Variables
char block1[64];      // Address 0 to 63
char block2[64];      // Address 64 to 127
char block3[64];      // Address 128 to 191
char block4[64];      // Address 192 to 255
char pStartOfArray = 0; // Incoming Sampled Character stored here then inc the pointer
char pScanPOS =0;

/* Forward declaration of functions */
void SetBaudRate();
int  ReadArray(char address);
void WriteArray(char address, char data);
void SetDAC(char X, char Y);

/** Sampling Interrupt routine **/
#INT_TIMER2           // timer2 interrupt, called 128.48 times per second
Timer2_ISR()
{
        long int adcValue16;
        char adcHI, adcLO, adcValue8;

        if (PUSH_BUTT == 1)   // If PUSB_BUTT = 0 screen is paused
        {
                adcValue16 = read_adc();              // Get ADC reading
                adcValue8 = (char)(adcValue16>>2);    // Converted to 8-bit as DAC is only 8-bit

                // simple trick for putting the new adc reading at the end of the array, e.g. put the
                // new reading at the start of the array and increment the start of array pointer
                // hence the new value is actually placed at the end of the array, overriding the
                // oldest value.
                WriteArray(pStartOfArray, adcValue8);
                pStartOfArray++;

                /** transmit reading through RS232 using Final Year Project Real-Time Frame Structure
                    (CH1) **/
                adcHI = (char)((adcValue16 >> 5)& 0x1f);     // Byte 1 (d9...d5) 0|0|0|d9|d8|d7|d6|d5
                adcLO = (char)((adcValue16 & 0x1f)|0x80);    // Byte 2 (d4...d0) 1|0|0|d4|d3|d2|d1|d0
                putc(adcHI);   // Transmit Byte 1
                putc(adcLO);   // Transmit Byte 2
        }
}


main()
{
        char i, data, address, bforward = TRUE;

        set_tris_a(0x02);      // TRISA = 00000011; RA2 to RA7 TTL Outputs
        set_tris_c(0xC0);      // TRISC = 11001000; RC0 to RC2, RC4, RC5 TTL Outputs (bits 7&6 must
                               // be set for UART to work)
        set_tris_d(0x00);      // TRISD = 00000000; RD0 to RD7 TTL Outputs
        port_b_pullups(FALSE); // Don't use internal pull up resistors

        SetBaudRate();

        /** Setup ADC **/
        setup_adc_ports(RA0_ANALOG);
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);
        delay_us(20);  // Delay for sampling cap to charge

        /** Setup timer2 **/
        setup_timer_2 (T2_DIV_BY_16,152,16); //(20,000,000/(4*16*152*16)) = 128.49 Hz
        set_timer2(0);
        enable_interrupts(INT_TIMER2);
        enable_interrupts(GLOBAL);

        while(TRUE) // Refreash Display
        {
                address = pStartOfArray + pScanPOS;
```

```
                data = ReadArray(address);
                SetDAC(pScanPOS,data);

                Switch (bforward)
                {
                        /** forward draw */
                        case TRUE:     if (pScanPOS == 0xFF)  bforward = FALSE;
                                       else pScanPOS++;
                                       break;

                        /** backward draw */
                        case FALSE:    if (pScanPOS == 0x00) bforward = TRUE;
                                       else pScanPOS--;
                                       break;

                        default:       bforward = TRUE; break;
                }
        }
}


void SetBaudRate()
{
        // Verison 1.1, 22/3/2002, by CKM

        set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 Inputs
                               // Parrell slave mode is ON, e.g. TTL Inputs, on Port E and D

        switch(PORTE & 0x07)   // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200); break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }

        set_tris_e(0x07);  // Turn OFF Parral Slave Mode as this affects port D (Address Bus)
}

int  ReadArray(char address)
{
        char data;

        if (address <64) data = block1[address];
        else if(address <128) data = block2[address-64];
        else if(address <192) data = block3[address-128];
        else data = block4[address-192];

        return data;
}

void WriteArray(char address, char data)
{
        if (address <64) block1[address] = data;
        else if(address <128) block2[address-64] = data;
        else if(address <192) block3[address-128] = data;
        else block4[address-192] = data;
}


void SetDAC(char X, char Y)
{
        // by CKM 14/04/2002
        set_tris_b(0x00); // Data bus output

        DATA_BUS = X;
        DAC_SELECT = 0;        // Selected DAC A
        DAC_ENABLE = 0;        // Enable DAC

        delay_cycles(1);       // 200nS delay

        DAC_ENABLE = 1;        // Disable DAC
        DATA_BUS = Y;
```

```
        DAC_SELECT = 1;        // Select DAC B
        DAC_ENABLE = 0;        // Enable DAC

        delay_cycles(1);       // 200nS delay
        DAC_ENABLE = 1;        // Disable DAC
}
```

A1.10. mark9.ls

```
MPASM

CCS PCM C Compiler, Version 2.707, 8851

               Filename: C:\WORK\COLIN\MPLAB\MINIPR~1\MARK9~1\MARK9.LST

               ROM used: 662 (8%)
                         Largest free fragment is 2048
               RAM used: 275 (75%) at main() level
                         287 (78%) worst case
               Stack:    3 worst case (2 in main + 1 for interrupts)

0000 3000           00001 MOVLW  00
0001 008A           00002 MOVWF  0A
0002 2A09           00003 GOTO   209
0003 0000           00004 NOP
0004 00FF           00005 MOVWF  7F
0005 0E03           00006 SWAPF  03,W
0006 1283           00007 BCF    03,5
0007 1303           00008 BCF    03,6
0008 00A1           00009 MOVWF  21
0009 080A           00010 MOVF   0A,W
000A 00A0           00011 MOVWF  20
000B 018A           00012 CLRF   0A
000C 0804           00013 MOVF   04,W
000D 00A2           00014 MOVWF  22
000E 0877           00015 MOVF   77,W
000F 00A3           00016 MOVWF  23
0010 0878           00017 MOVF   78,W
0011 00A4           00018 MOVWF  24
0012 0879           00019 MOVF   79,W
0013 00A5           00020 MOVWF  25
0014 087A           00021 MOVF   7A,W
0015 00A6           00022 MOVWF  26
0016 087B           00023 MOVF   7B,W
0017 00A7           00024 MOVWF  27
0018 1383           00025 BCF    03,7
0019 1283           00026 BCF    03,5
001A 308C           00027 MOVLW  8C
001B 0084           00028 MOVWF  04
001C 1C80           00029 BTFSS  00,1
001D 2820           00030 GOTO   020
001E 188C           00031 BTFSC  0C,1
001F 2833           00032 GOTO   033
0020 0822           00033 MOVF   22,W
0021 0084           00034 MOVWF  04
0022 0823           00035 MOVF   23,W
0023 00F7           00036 MOVWF  77
0024 0824           00037 MOVF   24,W
0025 00F8           00038 MOVWF  78
0026 0825           00039 MOVF   25,W
0027 00F9           00040 MOVWF  79
0028 0826           00041 MOVF   26,W
0029 00FA           00042 MOVWF  7A
002A 0827           00043 MOVF   27,W
002B 00FB           00044 MOVWF  7B
002C 0820           00045 MOVF   20,W
002D 008A           00046 MOVWF  0A
002E 0E21           00047 SWAPF  21,W
002F 0083           00048 MOVWF  03
0030 0EFF           00049 SWAPF  7F,F
0031 0E7F           00050 SWAPF  7F,W
0032 0009           00051 RETFIE
0033 118A           00052 BCF    0A,3
0034 120A           00053 BCF    0A,4
0035 2836           00054 GOTO   036
0000                00055 ....................  /* -------------------------------------------------------
0000                00056 ....................    | FILE    : mark9.c                                    |
0000                00057 ....................    | PROJECT : Mini Project: ECG                          |
0000                00058 ....................    | DESC    : CRT ECG NONITOR using Interrupt Sampling.  |
0000                00059 ....................    |         : ECG output to DAC, drawn bidirectional.    |
0000                00060 ....................    |         : ECG also outputted through RS232 using     |
0000                00061 ....................    |         : final year project real-time frame         |
0000                00062 ....................    |         : structure.                                 |
0000                00063 ....................    | ================================================== |
0000                00064 ....................    | DATE    : 22/04/2002                                 |
0000                00065 ....................    | BY      : Colin K McCord                             |
0000                00066 ....................    | VERSION : 1.4                                        |
0000                00067 ....................    -------------------------------------------------------- */
0000                00068 ....................
0000                00069 ....................
0000                00070 .................... #include <16F877.h>
0000                00071 ....................  //////// Standard Header file for the PIC16F877 device ////////
0000                00072 .................... #device PIC16F877
0000                00264 .................... #list
0000                00265 ....................
```

```
0000                ..................... #device PIC16F877 *=16 ADC=10
0000                ..................... 
0000                ..................... // use #device adc = 10 to implement a 10-bit conversion,
0000                ..................... // otherwise the default is 8-bits.
0000                ..................... 
0000                ..................... #fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT
0000                ..................... 
0000                ..................... #byte   PORTA = 0x05   // PortA lives in File 5 (ADC input and Control Lines)
0000                ..................... #byte   DATA_BUS = 0x06          // PortB lives in File 6 (data bus)
0000                ..................... #byte   PORTC = 0x07             // PortC lives in File 7 (UART and Control Lines)
0000                ..................... #byte   ADDRESS_BUS = 0x08       // PortD lives in File 8 (address bus A0 to A7)
0000                ..................... #byte   PORTE = 0x09             // PortE lives in File 9 (used for DIP switches)
0000                ..................... 
0000                ..................... /** Setup control lines with userfriendly names */
0000                ..................... #bit         DAC_ENABLE = PORTA.2          // C0
0000                ..................... #bit         DAC_SELECT = PORTA.3          // C1
0000                ..................... #bit         RAM_OE       = PORTA.4        // C2
0000                ..................... #bit         RAM_WRITE  = PORTA.5          // C3
0000                ..................... #bit         BUZZER     = PORTC.0          // C4
0000                ..................... #bit         SEG1_EN    = PORTC.1          // C5
0000                ..................... #bit         SEG_CLOCK  = PORTC.2          // C6
0000                ..................... #bit         SEG2_EN    = PORTC.5          // C7
0000                ..................... #bit         SEG3_EN    = PORTC.4          // C8
0000                ..................... 
0000                ..................... #bit         PUSH_BUTT  = PORTA.1
0000                ..................... 
0000                ..................... #byte        PORTE = 0x09        // PortE lives in File 9
0000                ..................... 
0000                ..................... #use delay(clock = 20000000)
0000                ..................... #use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)
0000                ..................... 
0000                ..................... #use         fast_io(A)          // Fast access to PortA (don't fiddle with TRISA)
0000                ..................... #use         fast_io(B)          // Fast access to PortB (don't fiddle with TRISB)
0000                ..................... #use         fast_io(C)          // Fast access to PortB (don't fiddle with TRISB)
0000                ..................... #use         fast_io(D)          // Fast access to PortD (don't fiddle with TRISD)
0000                ..................... #use         fast_io(E)          // Fast access to PortE (don't fiddle with TRISE)
0000                ..................... 
0000                ..................... /** Seven Seg display lookup table */
0000                ..................... const char SevenSeg[10] ={0b01000000,      // 0
0000                .....................                          0b01111001,       // 1
0000                .....................                          0b00100100,       // 2
0000                .....................                          0b00110000,       // 3
0000                .....................                          0b00011001,       // 4
0000                .....................                          0b00010010,       // 5
0000                .....................                          0b00000010,       // 6
0000                .....................                          0b01111000,       // 7
0000                .....................                          0b00000000,       // 8
0000                .....................                          0b00010000};      // 9
0000                ..................... 
0000                ..................... // Global Variables
0000                ..................... char block1[64];       // Address 0 to 63
0000                ..................... char block2[64];       // Address 64 to 127
0000                ..................... char block3[64];       // Address 128 to 191
0000                ..................... char block4[64];       // Address 192 to 255
0000                ..................... char pStartOfArray = 0; // Incoming Sampled Character stored here then inc the pointer
0000                ..................... char pScanPOS =0;
0000                ..................... 
0000                ..................... /* Forward declaration of functions */
0000                ..................... void SetBaudRate();
0000                ..................... int  ReadArray(char address);
0000                ..................... void WriteArray(char address, char data);
0000                ..................... void SetDAC(char X, char Y);
0000                ..................... 
0000                ..................... /** Sampling Interrupt routine **/
0000                ..................... #INT_TIMER2            // timer2 interrupt, called 128.48 times per second
0000                ..................... Timer2_ISR()
0000                ..................... {
0000                ..................... .   long int adcValue16;
0000                ..................... .   char adcHI, adcLO, adcValue8;
0000                ..................... 
0000                ..................... .   if (PUSH_BUTT == 1) // If PUSB_BUTT = 0 screen is paused
0036 3000           MOVLW  00
0037 1885           BTFSC  05,1
0038 3001           MOVLW  01
0039 00F7           MOVWF  77
003A 3001           MOVLW  01
003B 0277           SUBWF  77,W
003C 1D03           BTFSS  03,2
003D 2918           GOTO   118
0000                ..................... .   {
0000                ..................... .             adcValue16 = read_adc();                // Get ADC reading
003E 151F           BSF    1F,2
003F 191F           BTFSC  1F,2
0040 283F           GOTO   03F
0041 081E           MOVF   1E,W
0042 00FA           MOVWF  7A
0043 1683           BSF    03,5
0044 081E           MOVF   1E,W
0045 1703           BSF    03,6
0046 00D7           MOVWF  57
0047 1283           BCF    03,5
0048 1303           BCF    03,6
0049 087A           MOVF   7A,W
004A 1683           BSF    03,5
004B 1703           BSF    03,6
004C 00D8           MOVWF  58
0000                ..................... .             adcValue8 = (char)(adcValue16>>2);      // Converted to 8-bit as DAC is
only 8-bit
004D 0857           MOVF   57,W
004E 1283           BCF    03,5
004F 1303           BCF    03,6
0050 00F7           MOVWF  77
```

```
0051 1683           00367 BSF    03,5
0052 1703           00368 BSF    03,6
0053 0858           00369 MOVF   58,W
0054 1283           00370 BCF    03,5
0055 1303           00371 BCF    03,6
0056 00F9           00372 MOVWF  79
0057 0CF9           00373 RRF    79,F
0058 0CF7           00374 RRF    77,F
0059 0CF9           00375 RRF    79,F
005A 0CF7           00376 RRF    77,F
005B 303F           00377 MOVLW  3F
005C 05F9           00378 ANDWF  79,F
005D 0879           00379 MOVF   79,W
005E 00FA           00380 MOVWF  7A
005F 0877           00381 MOVF   77,W
0060 1683           00382 BSF    03,5
0061 1703           00383 BSF    03,6
0062 00DB           00384 MOVWF  5B
0000                00385 ....................
0000                00386 ....................              // simple trick for putting the new adc reading at the end of the array,
e.g. put the new
0000                00387 ....................              // reading at the start of the array and increment the start of array
pointer
0000                00388 ....................              // hence the new value is actually placed at the end of the array,
overriding the oldest value.
0000                00389 ....................              WriteArray(pStartOfArray, adcValue8);
0063 1283           00390 BCF    03,5
0064 1303           00391 BCF    03,6
0065 0868           00392 MOVF   68,W
0066 1683           00393 BSF    03,5
0067 1703           00394 BSF    03,6
0068 00DC           00395 MOVWF  5C
0069 085B           00396 MOVF   5B,W
006A 00DD           00397 MOVWF  5D
0000                00398 ....................              pStartOfArray++;
00DE 0AE8           00399 INCF   68,F
0000                00400 ....................
0000                00401 ....................              /** Transmit reading thorugh RS232 using Final Year Project Real-Time
Frame Struture (CH1) **/
0000                00402 ....................              adcHI = (char)((adcValue16 >> 5)& 0x1f);          // Byte 1 (d9...d5)
0|0|0|d9|d8|d7|d6|d5
00DF 1683           00403 BSF    03,5
00E0 1703           00404 BSF    03,6
00E1 0857           00405 MOVF   57,W
00E2 00DC           00406 MOVWF  5C
00E3 0858           00407 MOVF   58,W
00E4 00DD           00408 MOVWF  5D
00E5 0CDD           00409 RRF    5D,F
00E6 0CDC           00410 RRF    5C,F
00E7 0CDD           00411 RRF    5D,F
00E8 0CDC           00412 RRF    5C,F
00E9 0CDD           00413 RRF    5D,F
00EA 0CDC           00414 RRF    5C,F
00EB 0CDD           00415 RRF    5D,F
00EC 0CDC           00416 RRF    5C,F
00ED 0CDD           00417 RRF    5D,F
00EE 0CDC           00418 RRF    5C,F
00EF 3007           00419 MOVLW  07
00F0 05DD           00420 ANDWF  5D,F
00F1 085D           00421 MOVF   5D,W
00F2 3900           00422 ANDLW  00
00F3 1283           00423 BCF    03,5
00F4 1303           00424 BCF    03,6
00F5 00FA           00425 MOVWF  7A
00F6 1683           00426 BSF    03,5
00F7 1703           00427 BSF    03,6
00F8 085C           00428 MOVF   5C,W
00F9 391F           00429 ANDLW  1F
00FA 00D9           00430 MOVWF  59
0000                00431 ....................              adcLO = (char)((adcValue16 & 0x1f)|0x80);         // Byte 2 (d4...d0)
1|0|0|d4|d3|d2|d1|d0
00FB 0858           00432 MOVF   58,W
00FC 3900           00433 ANDLW  00
00FD 00DD           00434 MOVWF  5D
00FE 0857           00435 MOVF   57,W
00FF 391F           00436 ANDLW  1F
0100 00DC           00437 MOVWF  5C
0101 085D           00438 MOVF   5D,W
0102 1283           00439 BCF    03,5
0103 1303           00440 BCF    03,6
0104 00FA           00441 MOVWF  7A
0105 1683           00442 BSF    03,5
0106 1703           00443 BSF    03,6
0107 085C           00444 MOVF   5C,W
0108 3880           00445 IORLW  80
0109 00DA           00446 MOVWF  5A
0000                00447 ....................              putc(adcHI);        // Transmit Byte 1
010A 0859           00448 MOVF   59,W
010B 1283           00449 BCF    03,5
010C 1303           00450 BCF    03,6
010D 1E0C           00451 BTFSS  0C,4
010E 290D           00452 GOTO   10D
010F 0099           00453 MOVWF  19
0000                00454 ....................              putc(adcLO);        // Transmit Byte 2
0110 1683           00455 BSF    03,5
0111 1703           00456 BSF    03,6
0112 085A           00457 MOVF   5A,W
0113 1283           00458 BCF    03,5
0114 1303           00459 BCF    03,6
0115 1E0C           00460 BTFSS  0C,4
0116 2915           00461 GOTO   115
0117 0099           00462 MOVWF  19
```

```
0000                00463 ....................    }
0118 108C           00464 BCF    0C,1
0119 118A           00465 BCF    0A,3
011A 120A           00466 BCF    0A,4
011B 2820           00467 GOTO   020
0000                00468 .................... }
0000                00469 ....................
0000                00470 ....................
0000                00471 .................... main()
0000                00472 .................... {
021C 3001           00473 MOVLW  01
021D 1683           00474 BSF    03,5
021E 1703           00475 BSF    03,6
021F 00D3           00476 MOVWF  53
0000                00477 ....................    char i, data, address, bforward = TRUE;
0209 0184           00478 CLRF   04
020A 1383           00479 BCF    03,7
020B 301F           00480 MOVLW  1F
020C 0583           00481 ANDWF  03,F
020D 309F           00482 MOVLW  9F
020E 0084           00483 MOVWF  04
020F 1383           00484 BCF    03,7
0210 3007           00485 MOVLW  07
0211 0080           00486 MOVWF  00
0212 3081           00487 MOVLW  81
0213 1683           00488 BSF    03,5
0214 0099           00489 MOVWF  19
0215 3026           00490 MOVLW  26
0216 0098           00491 MOVWF  18
0217 3090           00492 MOVLW  90
0218 1283           00493 BCF    03,5
0219 0098           00494 MOVWF  18
021A 01E8           00495 CLRF   68
021B 01E9           00496 CLRF   69
0000                00497 ....................
0000                00498 ....................    set_tris_a(0x02);   // TRISA = 00000011; RA2 to RA7 TTL Outputs
0220 3002           00499 MOVLW  02
0221 0065           00500 TRIS   5
0000                00501 ....................    set_tris_c(0xC0);   // TRISC = 11001000; RC0 to RC2, RC4, RC5 TTL Outputs (bits 7&6
must be set for UART to work)
0222 30C0           00502 MOVLW  C0
0223 0067           00503 TRIS   7
0000                00504 ....................    set_tris_d(0x00);   // TRISD = 00000000; RD0 to RD7 TTL Outputs
0224 3000           00505 MOVLW  00
0225 1303           00506 BCF    03,6
0226 0088           00507 MOVWF  08
0000                00508 ....................    port_b_pullups(FALSE); // Don't use internal pull up resistors
0227 3081           00509 MOVLW  81
0228 0084           00510 MOVWF  04
0229 1383           00511 BCF    03,7
022A 1780           00512 BSF    00,7
0000                00513 ....................
0000                00514 ....................    SetBaudRate();
022B 1283           00515 BCF    03,5
022C 291C           00516 GOTO   11C
0000                00517 ....................
0000                00518 ....................    /** Setup ADC **/
0000                00519 ....................    setup_adc_ports(RA0_ANALOG);
022D 308E           00520 MOVLW  8E
022E 1683           00521 BSF    03,5
022F 009F           00522 MOVWF  1F
0000                00523 ....................    setup_adc(ADC_CLOCK_DIV_32);
0230 1283           00524 BCF    03,5
0231 081F           00525 MOVF   1F,W
0232 3938           00526 ANDLW  38
0233 3881           00527 IORLW  81
0234 009F           00528 MOVWF  1F
0000                00529 ....................    set_adc_channel(0);
0235 3000           00530 MOVLW  00
0236 00F8           00531 MOVWF  78
0237 081F           00532 MOVF   1F,W
0238 39C7           00533 ANDLW  C7
0239 0478           00534 IORWF  78,W
023A 009F           00535 MOVWF  1F
0000                00536 ....................    delay_us(20);       // Delay for sampling cap to charge
023B 3021           00537 MOVLW  21
023C 00F7           00538 MOVWF  77
023D 0BF7           00539 DECFSZ 77,F
023E 2A3D           00540 GOTO   23D
0000                00541 ....................
0000                00542 ....................    /** Setup timer2 **/
0000                00543 ....................    setup_timer_2 (T2_DIV_BY_16,152,16); //(20,000,000/(4*16*152*16)) = 128.49 Hz
023F 3078           00544 MOVLW  78
0240 00F8           00545 MOVWF  78
0241 0878           00546 MOVF   78,W
0242 3806           00547 IORLW  06
0243 0092           00548 MOVWF  12
0244 3098           00549 MOVLW  98
0245 1683           00550 BSF    03,5
0246 0092           00551 MOVWF  12
0000                00552 ....................    set_timer2(0);
0247 1283           00553 BCF    03,5
0248 0191           00554 CLRF   11
0000                00555 ....................    enable_interrupts(INT_TIMER2);
0249 308C           00556 MOVLW  8C
024A 0084           00557 MOVWF  04
024B 1383           00558 BCF    03,7
024C 1480           00559 BSF    00,1
0000                00560 ....................    enable_interrupts(GLOBAL);
024D 30C0           00561 MOVLW  C0
024E 048B           00562 IORWF  0B,F
0000                00563 ....................
```

```
0000                00564 ....................    while(TRUE) // Refreash Display
0000                00565 ....................    {
0000                00566 ....................              address = pStartOfArray + pScanPOS;
024F 0868           00567 MOVF   68,W
0250 0769           00568 ADDWF  69,W
0251 1683           00569 BSF    03,5
0252 1703           00570 BSF    03,6
0253 00D2           00571 MOVWF  52
0000                00572 ....................              data = ReadArray(address);
0254 0852           00573 MOVF   52,W
0255 00D4           00574 MOVWF  54
0256 1283           00575 BCF    03,5
0257 1303           00576 BCF    03,6
0258 2972           00577 GOTO   172
0259 0878           00578 MOVF   78,W
025A 1683           00579 BSF    03,5
025B 1703           00580 BSF    03,6
025C 00D1           00581 MOVWF  51
0000                00582 ....................              SetDAC(pScanPOS,data);
025D 1283           00583 BCF    03,5
025E 1303           00584 BCF    03,6
025F 0869           00585 MOVF   69,W
0260 1683           00586 BSF    03,5
0261 1703           00587 BSF    03,6
0262 00D4           00588 MOVWF  54
0263 0851           00589 MOVF   51,W
0264 00D5           00590 MOVWF  55
0265 1283           00591 BCF    03,5
0266 1303           00592 BCF    03,6
0267 29F0           00593 GOTO   1F0
0000                00594 ....................
0000                00595 ....................              Switch (bforward)
0268 1683           00596 BSF    03,5
0269 1703           00597 BSF    03,6
026A 0853           00598 MOVF   53,W
026B 1283           00599 BCF    03,5
026C 1303           00600 BCF    03,6
026D 00F7           00601 MOVWF  77
026E 3001           00602 MOVLW  01
026F 0277           00603 SUBWF  77,W
0270 1903           00604 BTFSC  03,2
0271 2A76           00605 GOTO   276
0272 0877           00606 MOVF   77,W
0273 1903           00607 BTFSC  03,2
0274 2A81           00608 GOTO   281
0275 2A8D           00609 GOTO   28D
0000                00610 ....................              {
0000                00611 ....................                        /** forward draw */
0000                00612 ....................                        case TRUE:          if (pScanPOS == 0xFF)         bforward =
FALSE;
0276 0A69           00613 INCF   69,W
0277 1D03           00614 BTFSS  03,2
0278 2A7F           00615 GOTO   27F
0279 1683           00616 BSF    03,5
027A 1703           00617 BSF    03,6
027B 01D3           00618 CLRF   53
0000                00619 ....................                                            else pScanPOS++;
027C 1283           00620 BCF    03,5
027D 1303           00621 BCF    03,6
027E 2A80           00622 GOTO   280
027F 0AE9           00623 INCF   69,F
0000                00624 ....................                                            break;
0280 2A94           00625 GOTO   294
0000                00626 ....................
0000                00627 ....................                        /** backward draw */
0000                00628 ....................                        case FALSE:         if (pScanPOS == 0x00) bforward = TRUE;
0281 08E9           00629 MOVF   69,F
0282 1D03           00630 BTFSS  03,2
0283 2A8B           00631 GOTO   28B
0284 3001           00632 MOVLW  01
0285 1683           00633 BSF    03,5
0286 1703           00634 BSF    03,6
0287 00D3           00635 MOVWF  53
0000                00636 ....................                                            else pScanPOS--;
0288 1283           00637 BCF    03,5
0289 1303           00638 BCF    03,6
028A 2A8C           00639 GOTO   28C
028B 03E9           00640 DECF   69,F
0000                00641 ....................                                            break;
028C 2A94           00642 GOTO   294
0000                00643 ....................
0000                00644 ....................                        default:  bforward = TRUE; break;
028D 3001           00645 MOVLW  01
028E 1683           00646 BSF    03,5
028F 1703           00647 BSF    03,6
0290 00D3           00648 MOVWF  53
0291 1283           00649 BCF    03,5
0292 1303           00650 BCF    03,6
0293 2A94           00651 GOTO   294
0000                00652 ....................              }
0000                00653 ....................    }
0294 2A4F           00654 GOTO   24F
0000                00655 .................... }
0000                00656 ....................
0295 0063           00657 SLEEP
0000                00658 ....................
0000                00659 .................... void SetBaudRate()
0000                00660 .................... {
0000                00661 ....................    // Verison 1.1, 22/3/2002, by CKM
0000                00662 ....................
0000                00663 ....................    set_tris_e(0x17);   // TRISE = 00010111; RE2,RE1 and RE0 Inputs
011C 3017           00664 MOVLW  17
```

```
011D 1683           00665 BSF    03,5
011E 0089           00666 MOVWF  09
0000                00667 ....................                                // Parrell slave mode is ON, e.g. TTL Inputs, on Port
E and D
0000                00668 ....................
0000                00669 ....................    switch(PORTE & 0x07)          // Read dip switches and setup baud rate
011F 1283           00670 BCF    03,5
0120 0809           00671 MOVF   09,W
0121 3907           00672 ANDLW  07
0122 3EF8           00673 ADDLW  F8
0123 1803           00674 BTFSC  03,0
0124 295F           00675 GOTO   15F
0125 3E08           00676 ADDLW  08
0126 2966           00677 GOTO   166
0000                00678 ....................    {
0000                00679 ....................              case 0: set_uart_speed(4800);   break;
0127 3040           00680 MOVLW  40
0128 1683           00681 BSF    03,5
0129 0099           00682 MOVWF  19
012A 3022           00683 MOVLW  22
012B 0098           00684 MOVWF  18
012C 1283           00685 BCF    03,5
012D 295F           00686 GOTO   15F
0000                00687 ....................              case 1: set_uart_speed(9600);   break;
012E 3081           00688 MOVLW  81
012F 1683           00689 BSF    03,5
0130 0099           00690 MOVWF  19
0131 3026           00691 MOVLW  26
0132 0098           00692 MOVWF  18
0133 1283           00693 BCF    03,5
0134 295F           00694 GOTO   15F
0000                00695 ....................              case 2: set_uart_speed(14400);  break;
0135 3056           00696 MOVLW  56
0136 1683           00697 BSF    03,5
0137 0099           00698 MOVWF  19
0138 3026           00699 MOVLW  26
0139 0098           00700 MOVWF  18
013A 1283           00701 BCF    03,5
013B 295F           00702 GOTO   15F
0000                00703 ....................              case 3:   set_uart_speed(19200);        break;
013C 3040           00704 MOVLW  40
013D 1683           00705 BSF    03,5
013E 0099           00706 MOVWF  19
013F 3026           00707 MOVLW  26
0140 0098           00708 MOVWF  18
0141 1283           00709 BCF    03,5
0142 295F           00710 GOTO   15F
0000                00711 ....................              case 4: set_uart_speed(32768);  break;
0143 3025           00712 MOVLW  25
0144 1683           00713 BSF    03,5
0145 0099           00714 MOVWF  19
0146 3026           00715 MOVLW  26
0147 0098           00716 MOVWF  18
0148 1283           00717 BCF    03,5
0149 295F           00718 GOTO   15F
0000                00719 ....................              case 5: set_uart_speed(38400);  break;
014A 3020           00720 MOVLW  20
014B 1683           00721 BSF    03,5
014C 0099           00722 MOVWF  19
014D 3026           00723 MOVLW  26
014E 0098           00724 MOVWF  18
014F 1283           00725 BCF    03,5
0150 295F           00726 GOTO   15F
0000                00727 ....................              case 6: set_uart_speed(57600);  break;
0151 3015           00728 MOVLW  15
0152 1683           00729 BSF    03,5
0153 0099           00730 MOVWF  19
0154 3026           00731 MOVLW  26
0155 0098           00732 MOVWF  18
0156 1283           00733 BCF    03,5
0157 295F           00734 GOTO   15F
0000                00735 ....................              case 7: set_uart_speed(115200); break;
0158 300A           00736 MOVLW  0A
0159 1683           00737 BSF    03,5
015A 0099           00738 MOVWF  19
015B 3026           00739 MOVLW  26
015C 0098           00740 MOVWF  18
015D 1283           00741 BCF    03,5
015E 295F           00742 GOTO   15F
0000                00743 ....................    }
0166 140A           00744 BSF    0A,0
0167 108A           00745 BCF    0A,1
0168 110A           00746 BCF    0A,2
0169 0782           00747 ADDWF  02,F
016A 2927           00748 GOTO   127
016B 292E           00749 GOTO   12E
016C 2935           00750 GOTO   135
016D 293C           00751 GOTO   13C
016E 2943           00752 GOTO   143
016F 294A           00753 GOTO   14A
0170 2951           00754 GOTO   151
0171 2958           00755 GOTO   158
0000                00756 ....................
0000                00757 ....................    set_tris_e(0x07);  // Turn OFF Parral Slave Mode as this affects port D (Address
Bus)
015F 3007           00758 MOVLW  07
0160 1683           00759 BSF    03,5
0161 0089           00760 MOVWF  09
0162 1283           00761 BCF    03,5
0163 118A           00762 BCF    0A,3
0164 120A           00763 BCF    0A,4
0165 2A2D           00764 GOTO   22D
```

```
0000                00765 ..................... }
0000                00766 .....................
0000                00767 ..................... int  ReadArray(char address)
0000                00768 ..................... {
0000                00769 .....................    char data;
0000                00770 .....................
0000                00771 .....................    if (address <64) data = block1[address];
0172 3040           00772 MOVLW  40
0173 1683           00773 BSF    03,5
0174 1703           00774 BSF    03,6
0175 0254           00775 SUBWF  54,W
0176 1C03           00776 BTFSS  03,0
0177 297B           00777 GOTO   17B
0178 1283           00778 BCF    03,5
0179 1303           00779 BCF    03,6
017A 298C           00780 GOTO   18C
017B 1283           00781 BCF    03,5
017C 1303           00782 BCF    03,6
017D 3028           00783 MOVLW  28
017E 1683           00784 BSF    03,5
017F 1703           00785 BSF    03,6
0180 0754           00786 ADDWF  54,W
0181 1283           00787 BCF    03,5
0182 1303           00788 BCF    03,6
0183 0084           00789 MOVWF  04
0184 1383           00790 BCF    03,7
0185 0800           00791 MOVF   00,W
0186 1683           00792 BSF    03,5
0187 1703           00793 BSF    03,6
0188 00D5           00794 MOVWF  55
0000                00795 .....................    else if(address <128) data = block2[address-64];
0189 1283           00796 BCF    03,5
018A 1303           00797 BCF    03,6
018B 29E7           00798 GOTO   1E7
018C 3080           00799 MOVLW  80
018D 1683           00800 BSF    03,5
018E 1703           00801 BSF    03,6
018F 0254           00802 SUBWF  54,W
0190 1C03           00803 BTFSS  03,0
0191 2995           00804 GOTO   195
0192 1283           00805 BCF    03,5
0193 1303           00806 BCF    03,6
0194 29A9           00807 GOTO   1A9
0195 1283           00808 BCF    03,5
0196 1303           00809 BCF    03,6
0197 3040           00810 MOVLW  40
0198 1683           00811 BSF    03,5
0199 1703           00812 BSF    03,6
019A 0254           00813 SUBWF  54,W
019B 1283           00814 BCF    03,5
019C 1303           00815 BCF    03,6
019D 00F7           00816 MOVWF  77
019E 30A0           00817 MOVLW  A0
019F 0777           00818 ADDWF  77,W
01A0 0084           00819 MOVWF  04
01A1 1383           00820 BCF    03,7
01A2 0800           00821 MOVF   00,W
01A3 1683           00822 BSF    03,5
01A4 1703           00823 BSF    03,6
01A5 00D5           00824 MOVWF  55
0000                00825 .....................    else if(address <192) data = block3[address-128];
01A6 1283           00826 BCF    03,5
01A7 1303           00827 BCF    03,6
01A8 29E7           00828 GOTO   1E7
01A9 30C0           00829 MOVLW  C0
01AA 1683           00830 BSF    03,5
01AB 1703           00831 BSF    03,6
01AC 0254           00832 SUBWF  54,W
01AD 1C03           00833 BTFSS  03,0
01AE 29B2           00834 GOTO   1B2
01AF 1283           00835 BCF    03,5
01B0 1303           00836 BCF    03,6
01B1 29CE           00837 GOTO   1CE
01B2 1283           00838 BCF    03,5
01B3 1303           00839 BCF    03,6
01B4 3080           00840 MOVLW  80
01B5 1683           00841 BSF    03,5
01B6 1703           00842 BSF    03,6
01B7 0254           00843 SUBWF  54,W
01B8 1283           00844 BCF    03,5
01B9 1303           00845 BCF    03,6
01BA 00F7           00846 MOVWF  77
01BB 3010           00847 MOVLW  10
01BC 0777           00848 ADDWF  77,W
01BD 00F9           00849 MOVWF  79
01BE 3001           00850 MOVLW  01
01BF 00FA           00851 MOVWF  7A
01C0 0879           00852 MOVF   79,W
01C1 1803           00853 BTFSC  03,0
01C2 0AFA           00854 INCF   7A,F
01C3 0084           00855 MOVWF  04
01C4 1383           00856 BCF    03,7
01C5 187A           00857 BTFSC  7A,0
01C6 1783           00858 BSF    03,7
01C7 0800           00859 MOVF   00,W
01C8 1683           00860 BSF    03,5
01C9 1703           00861 BSF    03,6
01CA 00D5           00862 MOVWF  55
0000                00863 .....................    else data = block4[address-192];
01CB 1283           00864 BCF    03,5
01CC 1303           00865 BCF    03,6
01CD 29E7           00866 GOTO   1E7
```

```
01CE 30C0          00867 MOVLW  C0
01CF 1683          00868 BSF    03,5
01D0 1703          00869 BSF    03,6
01D1 0254          00870 SUBWF  54,W
01D2 1283          00871 BCF    03,5
01D3 1303          00872 BCF    03,6
01D4 00F7          00873 MOVWF  77
01D5 3090          00874 MOVLW  90
01D6 0777          00875 ADDWF  77,W
01D7 00F9          00876 MOVWF  79
01D8 3001          00877 MOVLW  01
01D9 00FA          00878 MOVWF  7A
01DA 0879          00879 MOVF   79,W
01DB 1803          00880 BTFSC  03,0
01DC 0AFA          00881 INCF   7A,F
01DD 0084          00882 MOVWF  04
01DE 1383          00883 BCF    03,7
01DF 187A          00884 BTFSC  7A,0
01E0 1783          00885 BSF    03,7
01E1 0800          00886 MOVF   00,W
01E2 1683          00887 BSF    03,5
01E3 1703          00888 BSF    03,6
01E4 00D5          00889 MOVWF  55
01E5 1283          00890 BCF    03,5
01E6 1303          00891 BCF    03,6
0000               00892 ....................
0000               00893 ....................    return data;
01E7 1683          00894 BSF    03,5
01E8 1703          00895 BSF    03,6
01E9 0855          00896 MOVF   55,W
01EA 1283          00897 BCF    03,5
01EB 1303          00898 BCF    03,6
01EC 00F8          00899 MOVWF  78
01ED 118A          00900 BCF    0A,3
01EE 120A          00901 BCF    0A,4
01EF 2A59          00902 GOTO   259
0000               00903 .................... }
0000               00904 ....................
0000               00905 .................... void WriteArray(char address, char data)
0000               00906 .................... {
0000               00907 ....................    if (address <64) block1[address] = data;
006B 3040          00908 MOVLW  40
006C 025C          00909 SUBWF  5C,W
006D 1C03          00910 BTFSS  03,0
006E 2872          00911 GOTO   072
006F 1283          00912 BCF    03,5
0070 1303          00913 BCF    03,6
0071 2883          00914 GOTO   083
0072 1283          00915 BCF    03,5
0073 1303          00916 BCF    03,6
0074 3028          00917 MOVLW  28
0075 1683          00918 BSF    03,5
0076 1703          00919 BSF    03,6
0077 075C          00920 ADDWF  5C,W
0078 1283          00921 BCF    03,5
0079 1303          00922 BCF    03,6
007A 0084          00923 MOVWF  04
007B 1383          00924 BCF    03,7
007C 1683          00925 BSF    03,5
007D 1703          00926 BSF    03,6
007E 085D          00927 MOVF   5D,W
007F 0080          00928 MOVWF  00
0000               00929 ....................    else if(address <128) block2[address-64] = data;
0080 1283          00930 BCF    03,5
0081 1303          00931 BCF    03,6
0082 28DE          00932 GOTO   0DE
0083 3080          00933 MOVLW  80
0084 1683          00934 BSF    03,5
0085 1703          00935 BSF    03,6
0086 025C          00936 SUBWF  5C,W
0087 1C03          00937 BTFSS  03,0
0088 288C          00938 GOTO   08C
0089 1283          00939 BCF    03,5
008A 1303          00940 BCF    03,6
008B 28A0          00941 GOTO   0A0
008C 1283          00942 BCF    03,5
008D 1303          00943 BCF    03,6
008E 3040          00944 MOVLW  40
008F 1683          00945 BSF    03,5
0090 1703          00946 BSF    03,6
0091 025C          00947 SUBWF  5C,W
0092 1283          00948 BCF    03,5
0093 1303          00949 BCF    03,6
0094 00F7          00950 MOVWF  77
0095 30A0          00951 MOVLW  A0
0096 0777          00952 ADDWF  77,W
0097 0084          00953 MOVWF  04
0098 1383          00954 BCF    03,7
0099 1683          00955 BSF    03,5
009A 1703          00956 BSF    03,6
009B 085D          00957 MOVF   5D,W
009C 0080          00958 MOVWF  00
0000               00959 ....................    else if(address <192) block3[address-128] = data;
009D 1283          00960 BCF    03,5
009E 1303          00961 BCF    03,6
009F 28DE          00962 GOTO   0DE
00A0 30C0          00963 MOVLW  C0
00A1 1683          00964 BSF    03,5
00A2 1703          00965 BSF    03,6
00A3 025C          00966 SUBWF  5C,W
00A4 1C03          00967 BTFSS  03,0
00A5 28A9          00968 GOTO   0A9
```

```
00A6 1283          00969 BCF    03,5
00A7 1303          00970 BCF    03,6
00A8 28C5          00971 GOTO   0C5
00A9 1283          00972 BCF    03,5
00AA 1303          00973 BCF    03,6
00AB 3080          00974 MOVLW  80
00AC 1683          00975 BSF    03,5
00AD 1703          00976 BSF    03,6
00AE 025C          00977 SUBWF  5C,W
00AF 1283          00978 BCF    03,5
00B0 1303          00979 BCF    03,6
00B1 00F7          00980 MOVWF  77
00B2 3010          00981 MOVLW  10
00B3 0777          00982 ADDWF  77,W
00B4 00F9          00983 MOVWF  79
00B5 3001          00984 MOVLW  01
00B6 00FA          00985 MOVWF  7A
00B7 0879          00986 MOVF   79,W
00B8 1803          00987 BTFSC  03,0
00B9 0AFA          00988 INCF   7A,F
00BA 0084          00989 MOVWF  04
00BB 1383          00990 BCF    03,7
00BC 187A          00991 BTFSC  7A,0
00BD 1783          00992 BSF    03,7
00BE 1683          00993 BSF    03,5
00BF 1703          00994 BSF    03,6
00C0 085D          00995 MOVF   5D,W
00C1 0080          00996 MOVWF  00
0000               00997 ....................    else block4[address-192] = data;
00C2 1283          00998 BCF    03,5
00C3 1303          00999 BCF    03,6
00C4 28DE          01000 GOTO   0DE
00C5 30C0          01001 MOVLW  C0
00C6 1683          01002 BSF    03,5
00C7 1703          01003 BSF    03,6
00C8 025C          01004 SUBWF  5C,W
00C9 1283          01005 BCF    03,5
00CA 1303          01006 BCF    03,6
00CB 00F7          01007 MOVWF  77
00CC 3090          01008 MOVLW  90
00CD 0777          01009 ADDWF  77,W
00CE 00F9          01010 MOVWF  79
00CF 3001          01011 MOVLW  01
00D0 00FA          01012 MOVWF  7A
00D1 0879          01013 MOVF   79,W
00D2 1803          01014 BTFSC  03,0
00D3 0AFA          01015 INCF   7A,F
00D4 0084          01016 MOVWF  04
00D5 1383          01017 BCF    03,7
00D6 187A          01018 BTFSC  7A,0
00D7 1783          01019 BSF    03,7
00D8 1683          01020 BSF    03,5
00D9 1703          01021 BSF    03,6
00DA 085D          01022 MOVF   5D,W
00DB 0080          01023 MOVWF  00
00DC 1283          01024 BCF    03,5
00DD 1303          01025 BCF    03,6
0000               01026 .................... }
0000               01027 ....................
0000               01028 ....................
0000               01029 .................... void SetDAC(char X, char Y)
0000               01030 .................... {
0000               01031 ....................    // by CKM 14/04/2002
0000               01032 ....................    set_tris_b(0x00); // Data bus output
01F0 3000          01033 MOVLW  00
01F1 0066          01034 TRIS   6
0000               01035 ....................
0000               01036 ....................    DATA_BUS = X;
01F2 1683          01037 BSF    03,5
01F3 1703          01038 BSF    03,6
01F4 0854          01039 MOVF   54,W
01F5 1283          01040 BCF    03,5
01F6 1303          01041 BCF    03,6
01F7 0086          01042 MOVWF  06
0000               01043 ....................    DAC_SELECT = 0;     // Selected DAC A
01F8 1185          01044 BCF    05,3
0000               01045 ....................    DAC_ENABLE = 0;     // Enable DAC
01F9 1105          01046 BCF    05,2
0000               01047 ....................
0000               01048 ....................    delay_cycles(1);    // 200nS delay
01FA 0000          01049 NOP
0000               01050 ....................
0000               01051 ....................    DAC_ENABLE = 1;     // Disable DAC
01FB 1505          01052 BSF    05,2
0000               01053 ....................    DATA_BUS = Y;
01FC 1683          01054 BSF    03,5
01FD 1703          01055 BSF    03,6
01FE 0855          01056 MOVF   55,W
01FF 1283          01057 BCF    03,5
0200 1303          01058 BCF    03,6
0201 0086          01059 MOVWF  06
0000               01060 ....................    DAC_SELECT = 1;     // Select DAC B
0202 1585          01061 BSF    05,3
0000               01062 ....................    DAC_ENABLE = 0;     // Enable DAC
0203 1105          01063 BCF    05,2
0000               01064 ....................
0000               01065 ....................    delay_cycles(1);    // 200nS delay
0204 0000          01066 NOP
0000               01067 ....................    DAC_ENABLE = 1;     // Disable DAC
0205 1505          01068 BSF    05,2
0206 118A          01069 BCF    0A,3
0207 120A          01070 BCF    0A,4
```

```
0208 2A68          01071 GOTO   268
0000               01072 ..................... }
0000               01073 .....................
0000               01074 .....................

SYMBOL TABLE
  LABEL                           VALUE

PORTA                            00000005
DAC_ENABLE                       00000005
DAC_SELECT                       00000005
RAM_OE                           00000005
RAM_WRITE                        00000005
PUSH_BUTT                        00000005
DATA_BUS                         00000006
PORTC                            00000007
BUZZER                           00000007
SEG1_EN                          00000007
SEG_CLOCK                        00000007
SEG2_EN                          00000007
SEG3_EN                          00000007
PSP_DATA                         00000008
ADDRESS_BUS                      00000008
PORTE                            00000009
TIMER_1_LOW                      0000000E
TIMER_1_HIGH                     0000000F
TIMER_2                          00000011
CCP_1                            00000015
CCP_1_LOW                        00000015
CCP_1_HIGH                       00000016
CCP_2                            0000001B
CCP_2_LOW                        0000001B
CCP_2_HIGH                       0000001C
BLOCK1                           00000028
PSTARTOFARRAY                    00000068
PSCANPOS                         00000069
_RETURN_                         00000078
BLOCK2                           000000A0
BLOCK3                           00000110
BLOCK4                           00000190
MAIN.I                           000001D0
MAIN.DATA                        000001D1
MAIN.ADDRESS                     000001D2
MAIN.BFORWARD                    000001D3
READARRAY.ADDRESS                000001D4
SETDAC.X                         000001D4
SETDAC.Y                         000001D5
READARRAY.DATA                   000001D5
TIMER2_ISR.ADCVALUE16            000001D7
TIMER2_ISR.ADCHI                 000001D9
TIMER2_ISR.ADCLO                 000001DA
TIMER2_ISR.ADCVALUE8             000001DB
WRITEARRAY.ADDRESS               000001DC
WRITEARRAY.DATA                  000001DD
TIMER2_ISR                       00000036
MAIN                             00000209
SETBAUDRATE                      0000011C
READARRAY                        00000172
SETDAC                           000001F0

MEMORY USAGE
```

A1.11. mark10.c

```
/* -------------------------------------------------------
   | FILE    : mark10.c                                  |
   | PROJECT : Mini Project: ECG                         |
   | DESC    : CRT ECG NONITOR using Interrupt Sampling. |
   |         : ECG output to DAC, drawn bidirectional.   |
   |         : ECG also outputted through RS232 using    |
   |         : final year project real-time frame        |
   |         : structure. BPM displayed on 7-Segment     |
   |         : and buzzer beeps.                         |
   | ================================================== |
   | DATE    : 21/04/2002                                |
   | BY      : Colin K McCord                            |
   | VERSION : 1.5                                       |
   ------------------------------------------------------- */
#include <16F877.h>
#device PIC16F877 *=16 ADC=10

// use #device adc = 10 to implement a 10-bit conversion,
// otherwise the default is 8-bits.

#fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT

#byte   PORTA = 0x05          // PortA lives in File 5 (ADC input and Control Lines)
#byte   DATA_BUS = 0x06       // PortB lives in File 6 (data bus)
#byte   PORTC = 0x07          // PortC lives in File 7 (UART and Control Lines)
```

```
#byte   ADDRESS_BUS = 0x08     // PortD lives in File 8 (address bus A0 to A7)
#byte   PORTE = 0x09           // PortE lives in File 9 (used for DIP switches)

/** Setup control lines with user-friendly names */
#bit    DAC_ENABLE = PORTA.2   // C0
#bit    DAC_SELECT = PORTA.3   // C1
#bit    RAM_OE     = PORTA.4   // C2
#bit    RAM_WRITE  = PORTA.5   // C3
#bit    BUZZER     = PORTC.0   // C4
#bit    SEG1_EN    = PORTC.1   // C5
#bit    SEG_CLOCK  = PORTC.2   // C6
#bit    SEG2_EN    = PORTC.5   // C7
#bit     SEG3_EN    = PORTC.4  // C8

#bit    PUSH_BUTT  = PORTA.1

#byte   PORTE = 0x09   // PortE lives in File 9

#use delay(clock = 20000000)
#use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)

#use    fast_io(A)     // Fast access to PortA (don't fiddle with TRISA)
#use    fast_io(B)     // Fast access to PortB (don't fiddle with TRISB)
#use    fast_io(C)     // Fast access to PortB (don't fiddle with TRISB)
#use    fast_io(D)     // Fast access to PortD (don't fiddle with TRISD)
#use    fast_io(E)     // Fast access to PortE (don't fiddle with TRISE)

#define T0_INTS_PER_SEC 76    // (20,000,000/4*256*256) = 76.29 Hz

/** Seven Seg display lookup table */
const char SevenSeg[10] ={0b01000000, // 0
                          0b01111001,  // 1
                          0b00100100,  // 2
                          0b00110000,  // 3
                          0b00011001,  // 4
                          0b00010010,  // 5
                          0b00000010,  // 6
                          0b01111000,  // 7
                          0b00000000,  // 8
                          0b00010000}; // 9

// Global Variables
char block1[64];       // Address 0 to 63
char block2[64];       // Address 64 to 127
char block3[64];       // Address 128 to 191
char block4[64];       // Address 192 to 255
char pStartOfArray = 0; // Incoming Sampled Character stored here then inc the pointer
char pScanPOS =0;

char m_BuzzerOnTime = 0;
long ECG_MAX = 0;

char ECG_Peak_Counter = 0;
char ECG_Peak_CountDelay = 0;
byte int_count0;       // Timer 0 counter
byte SEG_Update = 0;
byte SEG_Average = 0;
char m_bpm = 0;        // Beats per min
char bpsArray[10];     // Store 10 seconds of ECG and calculate average
char SEG_State=0;
char Unit;
char Ten;
char Hundred;
char UpdateBPM = TRUE;
char BeatsPer10Sec = 0;

/* Forward declaration of functions */
void SetBaudRate();
int  ReadArray(char address);
void WriteArray(char address, char data);
void SetDAC(char X, char Y);
void Update7SEG();


/** Updated LEDs Interrupt Routine **/
#int_rtcc      // RTCC (timer0) interrupt, called every time RTCC overflows (255->0)
Timer0_ISR()
```

```
{
        if (--int_count0==0)   // Wait for 1 second.
        {
                int_count0 = T0_INTS_PER_SEC;
                SEG_Update++;
                SEG_Average++;

                bpsArray[SEG_Average-1] = ECG_Peak_Counter; // Convert bps to bpm
                ECG_Peak_Counter = 0;  //Reset Counter

                if (SEG_Update >= 4)   // Update LED display every four seconds
                {
                        UpdateBPM = TRUE;
                        SEG_Update = 0;
                }

                if (SEG_Average >= 10) // Reset average pointer
                {
                        SEG_Average = 0;
                }
        }

}


/** Sampling Interrupt routine **/
#INT_TIMER2            // timer2 interrupt, called 128.48 times per second
Timer2_ISR()
{
        long int adcValue16;
        char adcHI, adcLO, adcValue8;

        if (PUSH_BUTT == 1)    // If PUSB_BUTT = 0 screen is paused
        {
                adcValue16 = read_adc();              // Get ADC reading
                adcValue8 = (char)(adcValue16>>2);    // Converted to 8-bit as DAC is only 8-bit

                // simple trick for putting the new adc reading at the end of the array, e.g. put the
                // new reading at the start of the array and increment the start of array pointer
                // hence the new value is actually placed at the end of the array, overriding the
                // oldest value.
                WriteArray(pStartOfArray, adcValue8);
                pStartOfArray++;

                // Transmit reading through RS232 using Final Year Project Real-Time Frame Structure
                adcHI = (char)((adcValue16 >> 5)& 0x1f);     // Byte 1 (d9...d5) 0|0|0|d9|d8|d7|d6|d5
                adcLO = (char)((adcValue16 & 0x1f)|0x80);    // Byte 2 (d4...d0) 1|0|0|d4|d3|d2|d1|d0
                putc(adcHI);   // Transmit Byte 1
                putc(adcLO);   // Transmit Byte 2


                if (ECG_MAX < adcValue16) ECG_MAX = adcValue16;

                if (ECG_MAX-50 <= adcValue16)
                {
                        BUZZER = 1;            // Turn on Buzzer
                        m_BuzzerOnTime = 0;    // Reset Buzzer On Timer

                        /** 25 * 7.8uS = 195mS,  1/195ms*60 = 307 bpm max **/
                        if(ECG_Peak_CountDelay >= 25) // Min Delay make sure not to detect peak twice
                        {
                                ECG_Peak_Counter++;
                                ECG_Peak_CountDelay = 0;
                        }
                }

                if (m_BuzzerOnTime >= 15)
                {
                        BUZZER = 0;            // Turn off buzzer after 15 interrupts, that’s 117mS
                }
                else
                {
                        m_BuzzerOnTime++;      // Increment buzzer timer.
                }

                if(ECG_Peak_CountDelay < 25) ECG_Peak_CountDelay++;
        }
```

```
}


main()
{
        char i, data, address, bforward = TRUE;

        set_tris_a(0x02);      // TRISA = 00000011; RA1 to RA7 TTL Outputs
        set_tris_c(0xC0);      // TRISC = 11001000; (bits 7&6 must be set for UART to work)
        set_tris_d(0x00);      // TRISD = 00000000; RD0 to RD7 TTL Outputs
        port_b_pullups(FALSE); // Don't use internal pull up resistors

        SetBaudRate();

        /** Setup ADC **/
        setup_adc_ports(RA0_ANALOG);
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);
        delay_us(20);  // Delay for sampling cap to charge

        /** Setup timer0, Update 7-segment display **/
        int_count0 = T0_INTS_PER_SEC;
        set_rtcc(0);
        setup_counters(RTCC_INTERNAL,RTCC_DIV_256);  // (20,000,000/4*256*256) = 76.29 Hz
        enable_interrupts (RTCC_ZERO);

        /** Setup timer2 , Sampling, RS232, Buzzer **/
        setup_timer_2 (T2_DIV_BY_16,152,16); //(20,000,000/(4*16*152*16)) = 128.49 Hz
        set_timer2(0);
        enable_interrupts(INT_TIMER2);
        enable_interrupts(GLOBAL);

        while(TRUE) // Refreash Display
        {
                address = pStartOfArray + pScanPOS;
                data = ReadArray(address);
                SetDAC(pScanPOS,data);

                Switch (bforward)
                {
                        /** forward draw */
                        case TRUE:     if (pScanPOS == 0xFF)  bforward = FALSE;
                                       else pScanPOS++;
                                       break;

                        /** backward draw */
                        case FALSE:    if (pScanPOS == 0x00) bforward = TRUE;
                                       else pScanPOS--;
                                       break;

                        default:       bforward = TRUE; break;
                }

                /** Update 7-segment Display, approx. every 4 seconds **/
                If (UpdateBPM == TRUE && pScanPOS == 0) Update7SEG();
        }
}

void SetBaudRate()
{
        // Verison 1.1, 22/3/2002, by CKM

        set_tris_e(0x17);      // TRISE = 00010111; RE2,RE1 and RE0 Inputs
                               // Parallel slave mode is ON, e.g. TTL Inputs, on Port E and D

        switch(PORTE & 0x07)   // Read dip switches and setup baud rate
        {
                case 0: set_uart_speed(4800);   break;
                case 1: set_uart_speed(9600);   break;
                case 2: set_uart_speed(14400);  break;
                case 3: set_uart_speed(19200);  break;
                case 4: set_uart_speed(32768);  break;
                case 5: set_uart_speed(38400);  break;
                case 6: set_uart_speed(57600);  break;
                case 7: set_uart_speed(115200); break;
        }
```

```
        set_tris_e(0x07);  // Turn OFF Parallel Slave Mode as this affects port D (Address Bus)
}

int  ReadArray(char address)
{
        char data;

        if (address <64) data = block1[address];
        else if(address <128) data = block2[address-64];
        else if(address <192) data = block3[address-128];
        else data = block4[address-192];

        return data;
}

void WriteArray(char address, char data)
{
        if (address <64) block1[address] = data;
        else if(address <128) block2[address-64] = data;
        else if(address <192) block3[address-128] = data;
        else block4[address-192] = data;
}


void SetDAC(char X, char Y)
{
        // by CKM 14/04/2002
        set_tris_b(0x00); // Data bus output

        DATA_BUS = X;
        DAC_SELECT = 0;                // Selected DAC A
        DAC_ENABLE = 0;                // Enable DAC

        delay_cycles(1);       // 200nS delay

        DAC_ENABLE = 1;        // Disable DAC
        DATA_BUS = Y;
        DAC_SELECT = 1;        // Select DAC B
        DAC_ENABLE = 0;                // Enable DAC

        delay_cycles(1);       // 200nS delay
        DAC_ENABLE = 1;                // Disable DAC
}

Update7SEG()
{
        int i;
        char temp1;

        /* carried out in 5 steps, to reduce the affect caused to the waveform, e.g. between
           each step the waveform is redraw, for 50Hz each step must execute in less than
           78uS */

        switch(SEG_State)
        {

                case 0: // Cal bpm

                        /** ECG bpm calculated over 10 seconds, hence on power up it will take
                            10 seconds to display correct bpm value                            */
                        BeatsPer10Sec = 0;
                        for (i=0; i<10; i++) BeatsPer10Sec = BeatsPer10Sec + bpsArray[i];

                        /** BeatsPer10Sec * 6 for bps, add 6 times is much faster **/
                        m_bpm = BeatsPer10Sec + BeatsPer10Sec + BeatsPer10Sec + BeatsPer10Sec +
                        BeatsPer10Sec + BeatsPer10Sec;

                        unit = 0;
                        ten = 0;
                        hundred = 0;

                        SEG_State = 1;
                        break;

                // Steps 1 to 5 Convert Binary number into it hundreds, tens, units */
                case 1:
                        temp1 = m_bpm;
```

```
                m_bpm = m_bpm - 100;
                if(m_bpm > temp1)
                {
                        SEG_State = 2;
                        m_bpm = m_bpm +100;
                }
                else hundred++;

                break;

        case 2:
                temp1 = m_bpm;
                m_bpm = m_bpm - 10;
                if(m_bpm > temp1)
                {
                        SEG_state = 3;
                        m_bpm = m_bpm +10;
                }
                else ten++;

                break;

        case 3:
                temp1 = m_bpm;
                m_bpm--;
                if(m_bpm > temp1)
                {
                        SEG_state = 4;
                }
                else unit++;
                break;

        case 4: // Update Display
                DAC_ENABLE = 1;        // Disable DAC
                DAC_SELECT = 0;
                RAM_OE     = 1; // Disable RAM
                RAM_WRITE  = 1;

                // ENABLE UNITS DISPLAY
                SEG1_EN    = 1;
                SEG2_EN    = 1;
                SEG3_EN    = 0;

                DATA_BUS   = SevenSeg[Unit];

                SEG_CLOCK = 0;
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;

                // Enable TENS DISPLAY
                SEG1_EN    = 1;
                SEG2_EN    = 0;
                SEG3_EN    = 1;

                DATA_BUS   = SevenSeg[Ten];

                SEG_CLOCK = 0;
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;

                // ENABLE Hundred DISPLAY
                SEG1_EN    = 0;
                SEG2_EN    = 1;
                SEG3_EN    = 1;

                DATA_BUS   = SevenSeg[Hundred];

                SEG_CLOCK = 0;
                SEG_CLOCK = 1;
                SEG_CLOCK = 0;
                SEG_CLOCK = 1;

                SEG1_EN = 1;   // Disable display

                SEG_State = 0;
```

```
                        UpdateBPM = FALSE;
                        break;

                default:

                        SEG_State = 0;
                        break;
        }
}
```

A1.12. mark10.ls

```
MPASM

CCS PCM C Compiler, Version 2.707, 8851

               Filename: C:\WORK\COLIN\MPLAB\MINIPR~1\MARK10~1\MARK10B.LST

               ROM used: 1056 (13%)
                         Largest free fragment is 2048
               RAM used: 300 (82%) at main() level
                         314 (86%) worst case
               Stack:    3 worst case (2 in main + 1 for interrupts)

0000 3000           00001 MOVLW  00
0001 008A           00002 MOVWF  0A
0002 2B68           00003 GOTO   368
0003 0000           00004 NOP
0004 00FF           00005 MOVWF  7F
0005 0E03           00006 SWAPF  03,W
0006 1283           00007 BCF    03,5
0007 1303           00008 BCF    03,6
0008 00A1           00009 MOVWF  21
0009 080A           00010 MOVF   0A,W
000A 00A0           00011 MOVWF  20
000B 018A           00012 CLRF   0A
000C 0804           00013 MOVF   04,W
000D 00A2           00014 MOVWF  22
000E 0877           00015 MOVF   77,W
000F 00A3           00016 MOVWF  23
0010 0878           00017 MOVF   78,W
0011 00A4           00018 MOVWF  24
0012 0879           00019 MOVF   79,W
0013 00A5           00020 MOVWF  25
0014 087A           00021 MOVF   7A,W
0015 00A6           00022 MOVWF  26
0016 087B           00023 MOVF   7B,W
0017 00A7           00024 MOVWF  27
0018 1383           00025 BCF    03,7
0019 1283           00026 BCF    03,5
001A 1E8B           00027 BTFSS  0B,5
001B 281E           00028 GOTO   01E
001C 190B           00029 BTFSC  0B,2
001D 2837           00030 GOTO   037
001E 308C           00031 MOVLW  8C
001F 0084           00032 MOVWF  04
0020 1C80           00033 BTFSS  00,1
0021 2824           00034 GOTO   024
0022 188C           00035 BTFSC  0C,1
0023 283A           00036 GOTO   03A
0024 0822           00037 MOVF   22,W
0025 0084           00038 MOVWF  04
0026 0823           00039 MOVF   23,W
0027 00F7           00040 MOVWF  77
0028 0824           00041 MOVF   24,W
0029 00F8           00042 MOVWF  78
002A 0825           00043 MOVF   25,W
002B 00F9           00044 MOVWF  79
002C 0826           00045 MOVF   26,W
002D 00FA           00046 MOVWF  7A
002E 0827           00047 MOVF   27,W
002F 00FB           00048 MOVWF  7B
0030 0820           00049 MOVF   20,W
0031 008A           00050 MOVWF  0A
0032 0E21           00051 SWAPF  21,W
0033 0083           00052 MOVWF  03
0034 0EFF           00053 SWAPF  7F,F
0035 0E7F           00054 SWAPF  7F,W
0036 0009           00055 RETFIE
0037 118A           00056 BCF    0A,3
0038 120A           00057 BCF    0A,4
0039 284B           00058 GOTO   04B
003A 118A           00059 BCF    0A,3
003B 120A           00060 BCF    0A,4
003C 286A           00061 GOTO   06A
0000                00062 ....................  /* ------------------------------------------------------
0000                00063 ....................    | FILE    : mark10.c                                  |
0000                00064 ....................    | PROJECT : Mini Project: ECG                         |
0000                00065 ....................    | DESC    : CRT ECG NONITOR using Interrupt Sampling.  |
0000                00066 ....................    |         : ECG output to DAC, drawn bidirectional.    |
0000                00067 ....................    |         : ECG also outputted through RS232 using     |
0000                00068 ....................    |         : final year project real-time frame         |
0000                00069 ....................    |         : structure. BPM displayed on 7-Segment      |
0000                00070 ....................    |         : and buzzer beeps.                         |
0000                00071 ....................    | ================================================== |
```

```
0000                 ..................    | DATE    : 21/04/2002                                |
0000                 ..................    | BY      : Colin K McCord                            |
0000                 ..................    | VERSION : 1.5                                       |
0000                 ..................    ---------------------------------------------------- */
0000                 ..................
0000                 ..................
0000                 .................. #include <16F877.h>
0000                 ..................  //////// Standard Header file for the PIC16F877 device ////////
0000                 .................. #device PIC16F877
0000                 .................. #list
0000                 ..................
0000                 .................. #device PIC16F877 *=16 ADC=10
0000                 ..................
0000                 .................. // use #device adc = 10 to implement a 10-bit conversion,
0000                 .................. // otherwise the default is 8-bits.
0000                 ..................
0000                 .................. #fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT
0000                 ..................
0000                 .................. #byte   PORTA = 0x05   // PortA lives in File 5 (ADC input and Control Lines)
0000                 .................. #byte   DATA_BUS = 0x06          // PortB lives in File 6 (data bus)
0000                 .................. #byte   PORTC = 0x07             // PortC lives in File 7 (UART and Control Lines)
0000                 .................. #byte   ADDRESS_BUS = 0x08       // PortD lives in File 8 (address bus A0 to A7)
0000                 .................. #byte   PORTE = 0x09             // PortE lives in File 9 (used for DIP switches)
0000                 ..................
0000                 .................. /** Setup control lines with user-friendly names */
0000                 .................. #bit         DAC_ENABLE = PORTA.2          // C0
0000                 .................. #bit         DAC_SELECT = PORTA.3          // C1
0000                 .................. #bit         RAM_OE       = PORTA.4        // C2
0000                 .................. #bit         RAM_WRITE  = PORTA.5          // C3
0000                 .................. #bit         BUZZER     = PORTC.0          // C4
0000                 .................. #bit         SEG1_EN    = PORTC.1          // C5
0000                 .................. #bit         SEG_CLOCK  = PORTC.2          // C6
0000                 .................. #bit         SEG2_EN    = PORTC.5          // C7
0000                 .................. #bit         SEG3_EN    = PORTC.4          // C8
0000                 ..................
0000                 .................. #bit         PUSH_BUTT  = PORTA.1
0000                 ..................
0000                 .................. #byte        PORTE = 0x09        // PortE lives in File 9
0000                 ..................
0000                 .................. #use delay(clock = 20000000)
0000                 .................. #use rs232(baud=9600, xmit = PIN_C6, rcv = PIN_C7, parity = N, bits = 8)
0000                 ..................
0000                 .................. #use         fast_io(A)          // Fast access to PortA (don't fiddle with TRISA)
0000                 .................. #use         fast_io(B)          // Fast access to PortB (don't fiddle with TRISB)
0000                 .................. #use         fast_io(C)          // Fast access to PortB (don't fiddle with TRISB)
0000                 .................. #use         fast_io(D)          // Fast access to PortD (don't fiddle with TRISD)
0000                 .................. #use         fast_io(E)          // Fast access to PortE (don't fiddle with TRISE)
0000                 ..................
0000                 .................. #define T0_INTS_PER_SEC 76       // (20,000,000/4*256*256) = 76.29 Hz
0000                 ..................
0000                 .................. /** Seven Seg display lookup table */
0000                 .................. const char SevenSeg[10] ={0b01000000,      // 0
0000                 ..................                          0b01111001,       // 1
0000                 ..................                          0b00100100,       // 2
0000                 ..................                          0b00110000,       // 3
0000                 ..................                          0b00011001,       // 4
0000                 ..................                          0b00010010,       // 5
0000                 ..................                          0b00000010,       // 6
0000                 ..................                          0b01111000,       // 7
0000                 ..................                          0b00000000,       // 8
0000                 ..................                          0b00010000};      // 9
003D 100A            00324 BCF    0A,0
003E 108A            00325 BCF    0A,1
003F 110A            00326 BCF    0A,2
0040 0782            00327 ADDWF  02,F
0041 3440            00328 RETLW  40
0042 3479            00329 RETLW  79
0043 3424            00330 RETLW  24
0044 3430            00331 RETLW  30
0045 3419            00332 RETLW  19
0046 3412            00333 RETLW  12
0047 3402            00334 RETLW  02
0048 3478            00335 RETLW  78
0049 3400            00336 RETLW  00
004A 3410            00337 RETLW  10
0000                 ..................
0000                 .................. // Global Variables
0000                 .................. char block1[64];       // Address 0 to 63
0000                 .................. char block2[64];       // Address 64 to 127
0000                 .................. char block3[64];       // Address 128 to 191
0000                 .................. char block4[64];       // Address 192 to 255
0000                 .................. char pStartOfArray = 0; // Incoming Sampled Character stored here then inc the pointer
0000                 .................. char pScanPOS =0;
0000                 ..................
0000                 .................. char m_BuzzerOnTime = 0;
0000                 .................. long ECG_MAX = 0;
0000                 ..................
0000                 .................. char ECG_Peak_Counter = 0;
0000                 .................. char ECG_Peak_CountDelay = 0;
0000                 .................. byte int_count0;       // Timer 0 counter
0000                 .................. byte SEG_Update = 0;
0000                 .................. byte SEG_Average = 0;
0000                 .................. char m_bpm = 0;        // Beats per min
0000                 .................. char bpsArray[10];     // Store 10 seconds of ECG and calculate average
0000                 .................. char SEG_State=0;
0000                 .................. char Unit;
0000                 .................. char Ten;
0000                 .................. char Hundred;
0000                 .................. char UpdateBPM = TRUE;
0000                 .................. char BeatsPer10Sec = 0;
0000                 ..................
0000                 .................. /* Forward declaration of functions */
```

```
0000                 00365 ..................... void SetBaudRate();
0000                 00366 ..................... int  ReadArray(char address);
0000                 00367 ..................... void WriteArray(char address, char data);
0000                 00368 ..................... void SetDAC(char X, char Y);
0000                 00369 ..................... void Update7SEG();
0000                 00370 .....................
0000                 00371 .....................
0000                 00372 ..................... /** Updated LEDs Interrupt Rountine **/
0000                 00373 ..................... #int_rtcc    // RTCC (timer0) interrupt, called every time RTCC overflows (255->0)
0000                 00374 ..................... Timer0_ISR()
0000                 00375 ..................... {
0000                 00376 .....................    if (--int_count0==0)          // Wait for 1 second.
004B 0BEF            00377 DECFSZ 6F,F
004C 2866            00378 GOTO   066
0000                 00379 .....................    {
0000                 00380 .....................              int_count0 = T0_INTS_PER_SEC;
004D 304C            00381 MOVLW  4C
004E 00EF            00382 MOVWF  6F
0000                 00383 .....................              SEG_Update++;
004F 0AF0            00384 INCF   70,F
0000                 00385 .....................              SEG_Average++;
0050 0AF1            00386 INCF   71,F
0000                 00387 .....................
0000                 00388 .....................              bpsArray[SEG_Average-1] = ECG_Peak_Counter; // Convert bps to bpm
0051 3001            00389 MOVLW  01
0052 0271            00390 SUBWF  71,W
0053 00F7            00391 MOVWF  77
0054 30E0            00392 MOVLW  E0
0055 0777            00393 ADDWF  77,W
0056 0084            00394 MOVWF  04
0057 1383            00395 BCF    03,7
0058 086D            00396 MOVF   6D,W
0059 0080            00397 MOVWF  00
0000                 00398 .....................              ECG_Peak_Counter = 0;         //Reset Counter
005A 01ED            00399 CLRF   6D
0000                 00400 .....................
0000                 00401 .....................              if (SEG_Update >= 4)          // Update LED display every four seconds
005B 3004            00402 MOVLW  04
005C 0270            00403 SUBWF  70,W
005D 1C03            00404 BTFSS  03,0
005E 2862            00405 GOTO   062
0000                 00406 .....................              {
0000                 00407 .....................                        UpdateBPM = TRUE;
005F 3001            00408 MOVLW  01
0060 00FC            00409 MOVWF  7C
0000                 00410 .....................                        SEG_Update = 0;
0061 01F0            00411 CLRF   70
0000                 00412 .....................              }
0000                 00413 .....................
0000                 00414 .....................              if (SEG_Average >= 10) // Reset average pointer
0062 300A            00415 MOVLW  0A
0063 0271            00416 SUBWF  71,W
0064 1803            00417 BTFSC  03,0
0000                 00418 .....................              {
0000                 00419 .....................                        SEG_Average = 0;
0065 01F1            00420 CLRF   71
0000                 00421 .....................              }
0000                 00422 .....................    }
0000                 00423 .....................
0000                 00424 ..................... }
0000                 00425 .....................
0000                 00426 .....................
0000                 00427 ..................... /** Sampling Interrupt routine **/
0066 110B            00428 BCF    0B,2
0067 118A            00429 BCF    0A,3
0068 120A            00430 BCF    0A,4
0069 2824            00431 GOTO   024
0000                 00432 ..................... #INT_TIMER2            // timer2 interrupt, called 128.48 times per second
0000                 00433 ..................... Timer2_ISR()
0000                 00434 ..................... {
0000                 00435 .....................    long int adcValue16;
0000                 00436 .....................    char adcHI, adcLO, adcValue8;
0000                 00437 .....................
0000                 00438 .....................    if (PUSH_BUTT == 1) // If PUSB_BUTT = 0 screen is paused
006A 3000            00439 MOVLW  00
006B 1885            00440 BTFSC  05,1
006C 3001            00441 MOVLW  01
006D 00F7            00442 MOVWF  77
006E 3001            00443 MOVLW  01
006F 0277            00444 SUBWF  77,W
0070 1D03            00445 BTFSS  03,2
0071 29B7            00446 GOTO   1B7
0000                 00447 .....................    {
0000                 00448 .....................              adcValue16 = read_adc();                // Get ADC reading
0072 151F            00449 BSF    1F,2
0073 191F            00450 BTFSC  1F,2
0074 2873            00451 GOTO   073
0075 081E            00452 MOVF   1E,W
0076 00FA            00453 MOVWF  7A
0077 1683            00454 BSF    03,5
0078 081E            00455 MOVF   1E,W
0079 1703            00456 BSF    03,6
007A 00D9            00457 MOVWF  59
007B 1283            00458 BCF    03,5
007C 1303            00459 BCF    03,6
007D 087A            00460 MOVF   7A,W
007E 1683            00461 BSF    03,5
007F 1703            00462 BSF    03,6
0080 00DA            00463 MOVWF  5A
0000                 00464 .....................              adcValue8 = (char)(adcValue16>>2);      // Converted to 8-bit as DAC is
only 8-bit
0081 0859            00465 MOVF   59,W
```

```
0082 1283           00466 BCF    03,5
0083 1303           00467 BCF    03,6
0084 00F7           00468 MOVWF  77
0085 1683           00469 BSF    03,5
0086 1703           00470 BSF    03,6
0087 085A           00471 MOVF   5A,W
0088 1283           00472 BCF    03,5
0089 1303           00473 BCF    03,6
008A 00F9           00474 MOVWF  79
008B 0CF9           00475 RRF    79,F
008C 0CF7           00476 RRF    77,F
008D 0CF9           00477 RRF    79,F
008E 0CF7           00478 RRF    77,F
008F 303F           00479 MOVLW  3F
0090 05F9           00480 ANDWF  79,F
0091 0879           00481 MOVF   79,W
0092 00FA           00482 MOVWF  7A
0093 0877           00483 MOVF   77,W
0094 1683           00484 BSF    03,5
0095 1703           00485 BSF    03,6
0096 00DD           00486 MOVWF  5D
0000                00487 ....................
0000                00488 ....................              // simple trick for putting the new adc reading at the end of the array,
e.g. put the new
0000                00489 ....................              // reading at the start of the array and increment the start of array
pointer
0000                00490 ....................              // hence the new value is actually placed at the end of the array,
overriding the oldest value.
0000                00491 ....................              WriteArray(pStartOfArray, adcValue8);
0097 1283           00492 BCF    03,5
0098 1303           00493 BCF    03,6
0099 0868           00494 MOVF   68,W
009A 1683           00495 BSF    03,5
009B 1703           00496 BSF    03,6
009C 00DE           00497 MOVWF  5E
009D 085D           00498 MOVF   5D,W
009E 00DF           00499 MOVWF  5F
0000                00500 ....................              pStartOfArray++;
0112 0AE8           00501 INCF   68,F
0000                00502 ....................
0000                00503 ....................              /** Transmit reading thorugh RS232 using Final Year Project Real-Time
Frame Struture (CH1) **/
0000                00504 ....................              adcHI = (char)((adcValue16 >> 5)& 0x1f);          // Byte 1 (d9...d5)
0|0|0|d9|d8|d7|d6|d5
0113 1683           00505 BSF    03,5
0114 1703           00506 BSF    03,6
0115 0859           00507 MOVF   59,W
0116 00DE           00508 MOVWF  5E
0117 085A           00509 MOVF   5A,W
0118 00DF           00510 MOVWF  5F
0119 0CDF           00511 RRF    5F,F
011A 0CDE           00512 RRF    5E,F
011B 0CDF           00513 RRF    5F,F
011C 0CDE           00514 RRF    5E,F
011D 0CDF           00515 RRF    5F,F
011E 0CDE           00516 RRF    5E,F
011F 0CDF           00517 RRF    5F,F
0120 0CDE           00518 RRF    5E,F
0121 0CDF           00519 RRF    5F,F
0122 0CDE           00520 RRF    5E,F
0123 3007           00521 MOVLW  07
0124 05DF           00522 ANDWF  5F,F
0125 085F           00523 MOVF   5F,W
0126 3900           00524 ANDLW  00
0127 1283           00525 BCF    03,5
0128 1303           00526 BCF    03,6
0129 00FA           00527 MOVWF  7A
012A 1683           00528 BSF    03,5
012B 1703           00529 BSF    03,6
012C 085E           00530 MOVF   5E,W
012D 391F           00531 ANDLW  1F
012E 00DB           00532 MOVWF  5B
0000                00533 ....................              adcLO = (char)((adcValue16 & 0x1f)|0x80);         // Byte 2 (d4...d0)
1|0|0|d4|d3|d2|d1|d0
012F 085A           00534 MOVF   5A,W
0130 3900           00535 ANDLW  00
0131 00DF           00536 MOVWF  5F
0132 0859           00537 MOVF   59,W
0133 391F           00538 ANDLW  1F
0134 00DE           00539 MOVWF  5E
0135 085F           00540 MOVF   5F,W
0136 1283           00541 BCF    03,5
0137 1303           00542 BCF    03,6
0138 00FA           00543 MOVWF  7A
0139 1683           00544 BSF    03,5
013A 1703           00545 BSF    03,6
013B 085E           00546 MOVF   5E,W
013C 3880           00547 IORLW  80
013D 00DC           00548 MOVWF  5C
0000                00549 ....................              putc(adcHI);        // Transmit Byte 1
013E 085B           00550 MOVF   5B,W
013F 1283           00551 BCF    03,5
0140 1303           00552 BCF    03,6
0141 1E0C           00553 BTFSS  0C,4
0142 2941           00554 GOTO   141
0143 0099           00555 MOVWF  19
0000                00556 ....................              putc(adcLO);        // Transmit Byte 2
0144 1683           00557 BSF    03,5
0145 1703           00558 BSF    03,6
0146 085C           00559 MOVF   5C,W
0147 1283           00560 BCF    03,5
0148 1303           00561 BCF    03,6
```

```
0149 1E0C          00562 BTFSS  0C,4
014A 2949          00563 GOTO   149
014B 0099          00564 MOVWF  19
0000               00565 ....................
0000               00566 ....................
0000               00567 ....................              if (ECG_MAX < adcValue16) ECG_MAX = adcValue16;
014C 1683          00568 BSF    03,5
014D 1703          00569 BSF    03,6
014E 085A          00570 MOVF   5A,W
014F 1283          00571 BCF    03,5
0150 1303          00572 BCF    03,6
0151 026C          00573 SUBWF  6C,W
0152 1C03          00574 BTFSS  03,0
0153 2964          00575 GOTO   164
0154 1683          00576 BSF    03,5
0155 1703          00577 BSF    03,6
0156 085A          00578 MOVF   5A,W
0157 1283          00579 BCF    03,5
0158 1303          00580 BCF    03,6
0159 026C          00581 SUBWF  6C,W
015A 1D03          00582 BTFSS  03,2
015B 2970          00583 GOTO   170
015C 1683          00584 BSF    03,5
015D 1703          00585 BSF    03,6
015E 0859          00586 MOVF   59,W
015F 1283          00587 BCF    03,5
0160 1303          00588 BCF    03,6
0161 026B          00589 SUBWF  6B,W
0162 1803          00590 BTFSC  03,0
0163 2970          00591 GOTO   170
0164 1683          00592 BSF    03,5
0165 1703          00593 BSF    03,6
0166 085A          00594 MOVF   5A,W
0167 1283          00595 BCF    03,5
0168 1303          00596 BCF    03,6
0169 00EC          00597 MOVWF  6C
016A 1683          00598 BSF    03,5
016B 1703          00599 BSF    03,6
016C 0859          00600 MOVF   59,W
016D 1283          00601 BCF    03,5
016E 1303          00602 BCF    03,6
016F 00EB          00603 MOVWF  6B
0000               00604 ....................
0000               00605 ....................              if (ECG_MAX-50 <= adcValue16)
0170 086B          00606 MOVF   6B,W
0171 1683          00607 BSF    03,5
0172 1703          00608 BSF    03,6
0173 00DE          00609 MOVWF  5E
0174 3032          00610 MOVLW  32
0175 02DE          00611 SUBWF  5E,F
0176 1283          00612 BCF    03,5
0177 1303          00613 BCF    03,6
0178 086C          00614 MOVF   6C,W
0179 1683          00615 BSF    03,5
017A 1703          00616 BSF    03,6
017B 00DF          00617 MOVWF  5F
017C 3000          00618 MOVLW  00
017D 1C03          00619 BTFSS  03,0
017E 03DF          00620 DECF   5F,F
017F 02DF          00621 SUBWF  5F,F
0180 085A          00622 MOVF   5A,W
0181 025F          00623 SUBWF  5F,W
0182 1803          00624 BTFSC  03,0
0183 2987          00625 GOTO   187
0184 1283          00626 BCF    03,5
0185 1303          00627 BCF    03,6
0186 29A4          00628 GOTO   1A4
0187 1283          00629 BCF    03,5
0188 1303          00630 BCF    03,6
0189 1683          00631 BSF    03,5
018A 1703          00632 BSF    03,6
018B 085A          00633 MOVF   5A,W
018C 025F          00634 SUBWF  5F,W
018D 1903          00635 BTFSC  03,2
018E 2992          00636 GOTO   192
018F 1283          00637 BCF    03,5
0190 1303          00638 BCF    03,6
0191 29AC          00639 GOTO   1AC
0192 1283          00640 BCF    03,5
0193 1303          00641 BCF    03,6
0194 1683          00642 BSF    03,5
0195 1703          00643 BSF    03,6
0196 0859          00644 MOVF   59,W
0197 025E          00645 SUBWF  5E,W
0198 1D03          00646 BTFSS  03,2
0199 299D          00647 GOTO   19D
019A 1283          00648 BCF    03,5
019B 1303          00649 BCF    03,6
019C 29A4          00650 GOTO   1A4
019D 1C03          00651 BTFSS  03,0
019E 29A2          00652 GOTO   1A2
019F 1283          00653 BCF    03,5
01A0 1303          00654 BCF    03,6
01A1 29AC          00655 GOTO   1AC
01A2 1283          00656 BCF    03,5
01A3 1303          00657 BCF    03,6
0000               00658 ....................              {
0000               00659 ....................                        BUZZER = 1;                    // Turn on Buzzer
01A4 1407          00660 BSF    07,0
0000               00661 ....................                        m_BuzzerOnTime = 0; // Reset Buzzer On Timer
01A5 01EA          00662 CLRF   6A
0000               00663 ....................
```

```
0000                00664 ....................                        /** 25 * 7.8uS = 195mS,  1/195ms*60 = 307 bpm max **/
0000                00665 ....................                        if(ECG_Peak_CountDelay >= 25) // Min Delay make sure not to
detect peak twice
01A6 3019           00666 MOVLW  19
01A7 026E           00667 SUBWF  6E,W
01A8 1C03           00668 BTFSS  03,0
01A9 29AC           00669 GOTO   1AC
0000                00670 ....................                        {
0000                00671 ....................                                  ECG_Peak_Counter++;
01AA 0AED           00672 INCF   6D,F
0000                00673 ....................                                  ECG_Peak_CountDelay = 0;
01AB 01EE           00674 CLRF   6E
0000                00675 ....................                        }
0000                00676 ....................              }
0000                00677 ....................
0000                00678 ....................              if (m_BuzzerOnTime >= 15)
01AC 300F           00679 MOVLW  0F
01AD 026A           00680 SUBWF  6A,W
01AE 1C03           00681 BTFSS  03,0
01AF 29B2           00682 GOTO   1B2
0000                00683 ....................              {
0000                00684 ....................                        BUZZER = 0;                   // Turn off buzzer after 15
interrupts, thats 117mS
01B0 1007           00685 BCF    07,0
0000                00686 ....................              }
0000                00687 ....................              else
01B1 29B3           00688 GOTO   1B3
0000                00689 ....................              {
0000                00690 ....................                        m_BuzzerOnTime++;   // Increment buzzer timer.
01B2 0AEA           00691 INCF   6A,F
0000                00692 ....................              }
0000                00693 ....................
0000                00694 ....................              if(ECG_Peak_CountDelay < 25) ECG_Peak_CountDelay++;
01B3 3019           00695 MOVLW  19
01B4 026E           00696 SUBWF  6E,W
01B5 1C03           00697 BTFSS  03,0
01B6 0AEE           00698 INCF   6E,F
0000                00699 ....................    }
01B7 108C           00700 BCF    0C,1
01B8 118A           00701 BCF    0A,3
01B9 120A           00702 BCF    0A,4
01BA 2824           00703 GOTO   024
0000                00704 .................... }
0000                00705 ....................
0000                00706 ....................
0000                00707 .................... main()
0000                00708 .................... {
0387 3001           00709 MOVLW  01
0388 1683           00710 BSF    03,5
0389 1703           00711 BSF    03,6
038A 00D3           00712 MOVWF  53
0000                00713 ....................    char i, data, address, bforward = TRUE;
0368 0184           00714 CLRF   04
0369 1383           00715 BCF    03,7
036A 301F           00716 MOVLW  1F
036B 0583           00717 ANDWF  03,F
036C 309F           00718 MOVLW  9F
036D 0084           00719 MOVWF  04
036E 1383           00720 BCF    03,7
036F 3007           00721 MOVLW  07
0370 0080           00722 MOVWF  00
0371 3081           00723 MOVLW  81
0372 1683           00724 BSF    03,5
0373 0099           00725 MOVWF  19
0374 3026           00726 MOVLW  26
0375 0098           00727 MOVWF  18
0376 3090           00728 MOVLW  90
0377 1283           00729 BCF    03,5
0378 0098           00730 MOVWF  18
0379 01E8           00731 CLRF   68
037A 01E9           00732 CLRF   69
037B 01EA           00733 CLRF   6A
037C 01EB           00734 CLRF   6B
037D 01EC           00735 CLRF   6C
037E 01ED           00736 CLRF   6D
037F 01EE           00737 CLRF   6E
0380 01F0           00738 CLRF   70
0381 01F1           00739 CLRF   71
0382 01F2           00740 CLRF   72
0383 01F3           00741 CLRF   73
0384 3001           00742 MOVLW  01
0385 00FC           00743 MOVWF  7C
0386 01FD           00744 CLRF   7D
0000                00745 ....................
0000                00746 ....................    set_tris_a(0x02);   // TRISA = 00000001; RA1 to RA7 TTL Outputs
038B 3002           00747 MOVLW  02
038C 0065           00748 TRIS   5
0000                00749 ....................    set_tris_c(0xC0);   // TRISC = 11001000; (bits 7&6 must be set for UART to work)

038D 30C0           00750 MOVLW  C0
038E 0067           00751 TRIS   7
0000                00752 ....................    set_tris_d(0x00);   // TRISD = 00000000; RD0 to RD7 TTL Outputs
038F 3000           00753 MOVLW  00
0390 1303           00754 BCF    03,6
0391 0088           00755 MOVWF  08
0000                00756 ....................    port_b_pullups(FALSE);        // Don't use internal pull up resistors
0392 3081           00757 MOVLW  81
0393 0084           00758 MOVWF  04
0394 1383           00759 BCF    03,7
0395 1780           00760 BSF    00,7
0000                00761 ....................
0000                00762 ....................    SetBaudRate();
```

```
0396 1283          00763 BCF    03,5
0397 29BB          00764 GOTO   1BB
0000               00765 ....................
0000               00766 ....................    /** Setup ADC **/
0000               00767 ....................    setup_adc_ports(RA0_ANALOG);
0398 308E          00768 MOVLW  8E
0399 1683          00769 BSF    03,5
039A 009F          00770 MOVWF  1F
0000               00771 ....................    setup_adc(ADC_CLOCK_DIV_32);
039B 1283          00772 BCF    03,5
039C 081F          00773 MOVF   1F,W
039D 3938          00774 ANDLW  38
039E 3881          00775 IORLW  81
039F 009F          00776 MOVWF  1F
0000               00777 ....................    set_adc_channel(0);
03A0 3000          00778 MOVLW  00
03A1 00F8          00779 MOVWF  78
03A2 081F          00780 MOVF   1F,W
03A3 39C7          00781 ANDLW  C7
03A4 0478          00782 IORWF  78,W
03A5 009F          00783 MOVWF  1F
0000               00784 ....................    delay_us(20);       // Delay for sampling cap to charge
03A6 3021          00785 MOVLW  21
03A7 00F7          00786 MOVWF  77
03A8 0BF7          00787 DECFSZ 77,F
03A9 2BA8          00788 GOTO   3A8
0000               00789 ....................
0000               00790 ....................    /** Setup timer0, Update 7-segment display **/
0000               00791 ....................    int_count0 = T0_INTS_PER_SEC;
03AA 304C          00792 MOVLW  4C
03AB 00EF          00793 MOVWF  6F
0000               00794 ....................    set_rtcc(0);
03AC 0181          00795 CLRF   01
0000               00796 ....................    setup_counters(RTCC_INTERNAL,RTCC_DIV_256);       // (20,000,000/4*256*256) = 76.29
Hz
03AD 3007          00797 MOVLW  07
03AE 00F7          00798 MOVWF  77
03AF 1DF7          00799 BTFSS  77,3
03B0 2BBA          00800 GOTO   3BA
03B1 3007          00801 MOVLW  07
03B2 0181          00802 CLRF   01
03B3 3081          00803 MOVLW  81
03B4 0084          00804 MOVWF  04
03B5 1383          00805 BCF    03,7
03B6 0800          00806 MOVF   00,W
03B7 39C0          00807 ANDLW  C0
03B8 380F          00808 IORLW  0F
03B9 0080          00809 MOVWF  00
03BA 0064          00810 CLRWDT
03BB 3081          00811 MOVLW  81
03BC 0084          00812 MOVWF  04
03BD 0800          00813 MOVF   00,W
03BE 39C0          00814 ANDLW  C0
03BF 0477          00815 IORWF  77,W
03C0 0080          00816 MOVWF  00
0000               00817 ....................    enable_interrupts (RTCC_ZERO);
03C1 168B          00818 BSF    0B,5
0000               00819 ....................
0000               00820 ....................    /** Setup timer2 , Sampling, RS232, Buzzer **/
0000               00821 ....................    setup_timer_2 (T2_DIV_BY_16,152,16); //(20,000,000/(4*16*152*16)) = 128.49 Hz
03C2 3078          00822 MOVLW  78
03C3 00F8          00823 MOVWF  78
03C4 0878          00824 MOVF   78,W
03C5 3806          00825 IORLW  06
03C6 0092          00826 MOVWF  12
03C7 3098          00827 MOVLW  98
03C8 1683          00828 BSF    03,5
03C9 0092          00829 MOVWF  12
0000               00830 ....................    set_timer2(0);
03CA 1283          00831 BCF    03,5
03CB 0191          00832 CLRF   11
0000               00833 ....................    enable_interrupts(INT_TIMER2);
03CC 308C          00834 MOVLW  8C
03CD 0084          00835 MOVWF  04
03CE 1383          00836 BCF    03,7
03CF 1480          00837 BSF    00,1
0000               00838 ....................    enable_interrupts(GLOBAL);
03D0 30C0          00839 MOVLW  C0
03D1 048B          00840 IORWF  0B,F
0000               00841 ....................
0000               00842 ....................    while(TRUE) // Refreash Display
0000               00843 ....................    {
0000               00844 ....................              address = pStartOfArray + pScanPOS;
03D2 0868          00845 MOVF   68,W
03D3 0769          00846 ADDWF  69,W
03D4 1683          00847 BSF    03,5
03D5 1703          00848 BSF    03,6
03D6 00D2          00849 MOVWF  52
0000               00850 ....................              data = ReadArray(address);
03D7 0852          00851 MOVF   52,W
03D8 00D4          00852 MOVWF  54
03D9 1283          00853 BCF    03,5
03DA 1303          00854 BCF    03,6
03DB 2A11          00855 GOTO   211
03DC 0878          00856 MOVF   78,W
03DD 1683          00857 BSF    03,5
03DE 1703          00858 BSF    03,6
03DF 00D1          00859 MOVWF  51
0000               00860 ....................              SetDAC(pScanPOS,data);
03E0 1283          00861 BCF    03,5
03E1 1303          00862 BCF    03,6
03E2 0869          00863 MOVF   69,W
```

```
03E3 1683          00864 BSF    03,5
03E4 1703          00865 BSF    03,6
03E5 00D4          00866 MOVWF  54
03E6 0851          00867 MOVF   51,W
03E7 00D5          00868 MOVWF  55
03E8 1283          00869 BCF    03,5
03E9 1303          00870 BCF    03,6
03EA 2A8F          00871 GOTO   28F
0000               00872 ....................
0000               00873 ....................             Switch (bforward)
03EB 1683          00874 BSF    03,5
03EC 1703          00875 BSF    03,6
03ED 0853          00876 MOVF   53,W
03EE 1283          00877 BCF    03,5
03EF 1303          00878 BCF    03,6
03F0 00F7          00879 MOVWF  77
03F1 3001          00880 MOVLW  01
03F2 0277          00881 SUBWF  77,W
03F3 1903          00882 BTFSC  03,2
03F4 2BF9          00883 GOTO   3F9
03F5 0877          00884 MOVF   77,W
03F6 1903          00885 BTFSC  03,2
03F7 2C04          00886 GOTO   404
03F8 2C10          00887 GOTO   410
0000               00888 ....................             {
0000               00889 ....................                       /** forward draw */
0000               00890 ....................                       case TRUE:          if (pScanPOS == 0xFF)         bforward =
FALSE;
03F9 0A69          00891 INCF   69,W
03FA 1D03          00892 BTFSS  03,2
03FB 2C02          00893 GOTO   402
03FC 1683          00894 BSF    03,5
03FD 1703          00895 BSF    03,6
03FE 01D3          00896 CLRF   53
0000               00897 ....................                                            else pScanPOS++;
03FF 1283          00898 BCF    03,5
0400 1303          00899 BCF    03,6
0401 2C03          00900 GOTO   403
0402 0AE9          00901 INCF   69,F
0000               00902 ....................                                            break;
0403 2C17          00903 GOTO   417
0000               00904 ....................
0000               00905 ....................                       /** backward draw */
0000               00906 ....................                       case FALSE:         if (pScanPOS == 0x00) bforward = TRUE;
0404 08E9          00907 MOVF   69,F
0405 1D03          00908 BTFSS  03,2
0406 2C0E          00909 GOTO   40E
0407 3001          00910 MOVLW  01
0408 1683          00911 BSF    03,5
0409 1703          00912 BSF    03,6
040A 00D3          00913 MOVWF  53
0000               00914 ....................                                            else pScanPOS--;
040B 1283          00915 BCF    03,5
040C 1303          00916 BCF    03,6
040D 2C0F          00917 GOTO   40F
040E 03E9          00918 DECF   69,F
0000               00919 ....................                                            break;
040F 2C17          00920 GOTO   417
0000               00921 ....................
0000               00922 ....................                       default:  bforward = TRUE; break;
0410 3001          00923 MOVLW  01
0411 1683          00924 BSF    03,5
0412 1703          00925 BSF    03,6
0413 00D3          00926 MOVWF  53
0414 1283          00927 BCF    03,5
0415 1303          00928 BCF    03,6
0416 2C17          00929 GOTO   417
0000               00930 ....................             }
0000               00931 ....................
0000               00932 ....................             /** Update 7-segment Display, approx. every 4 seconds **/
0000               00933 ....................             If (UpdateBPM == TRUE && pScanPOS == 0) Update7SEG();
0417 3001          00934 MOVLW  01
0418 027C          00935 SUBWF  7C,W
0419 1D03          00936 BTFSS  03,2
041A 2C1E          00937 GOTO   41E
041B 08E9          00938 MOVF   69,F
041C 1903          00939 BTFSC  03,2
041D 2AA8          00940 GOTO   2A8
0000               00941 ....................    }
041E 2BD2          00942 GOTO   3D2
0000               00943 .................... }
0000               00944 ....................
041F 0063          00945 SLEEP
0000               00946 .................... void SetBaudRate()
0000               00947 .................... {
0000               00948 ....................    // Verison 1.1, 22/3/2002, by CKM
0000               00949 ....................
0000               00950 ....................    set_tris_e(0x17);   // TRISE = 00010111; RE2,RE1 and RE0 Inputs
01BB 3017          00951 MOVLW  17
01BC 1683          00952 BSF    03,5
01BD 0089          00953 MOVWF  09
0000               00954 ....................                                 // Parrell slave mode is ON, e.g. TTL Inputs, on Port
E and D
0000               00955 ....................
0000               00956 ....................    switch(PORTE & 0x07)          // Read dip switches and setup baud rate
01BE 1283          00957 BCF    03,5
01BF 0809          00958 MOVF   09,W
01C0 3907          00959 ANDLW  07
01C1 3EF8          00960 ADDLW  F8
01C2 1803          00961 BTFSC  03,0
01C3 29FE          00962 GOTO   1FE
01C4 3E08          00963 ADDLW  08
```

```
01C5 2A05          00964 GOTO   205
0000               00965 ....................    {
0000               00966 ....................              case 0: set_uart_speed(4800);   break;
01C6 3040          00967 MOVLW  40
01C7 1683          00968 BSF    03,5
01C8 0099          00969 MOVWF  19
01C9 3022          00970 MOVLW  22
01CA 0098          00971 MOVWF  18
01CB 1283          00972 BCF    03,5
01CC 29FE          00973 GOTO   1FE
0000               00974 ....................              case 1: set_uart_speed(9600);   break;
01CD 3081          00975 MOVLW  81
01CE 1683          00976 BSF    03,5
01CF 0099          00977 MOVWF  19
01D0 3026          00978 MOVLW  26
01D1 0098          00979 MOVWF  18
01D2 1283          00980 BCF    03,5
01D3 29FE          00981 GOTO   1FE
0000               00982 ....................              case 2: set_uart_speed(14400);  break;
01D4 3056          00983 MOVLW  56
01D5 1683          00984 BSF    03,5
01D6 0099          00985 MOVWF  19
01D7 3026          00986 MOVLW  26
01D8 0098          00987 MOVWF  18
01D9 1283          00988 BCF    03,5
01DA 29FE          00989 GOTO   1FE
0000               00990 ....................              case 3:   set_uart_speed(19200);        break;
01DB 3040          00991 MOVLW  40
01DC 1683          00992 BSF    03,5
01DD 0099          00993 MOVWF  19
01DE 3026          00994 MOVLW  26
01DF 0098          00995 MOVWF  18
01E0 1283          00996 BCF    03,5
01E1 29FE          00997 GOTO   1FE
0000               00998 ....................              case 4: set_uart_speed(32768);  break;
01E2 3025          00999 MOVLW  25
01E3 1683          01000 BSF    03,5
01E4 0099          01001 MOVWF  19
01E5 3026          01002 MOVLW  26
01E6 0098          01003 MOVWF  18
01E7 1283          01004 BCF    03,5
01E8 29FE          01005 GOTO   1FE
0000               01006 ....................              case 5: set_uart_speed(38400);  break;
01E9 3020          01007 MOVLW  20
01EA 1683          01008 BSF    03,5
01EB 0099          01009 MOVWF  19
01EC 3026          01010 MOVLW  26
01ED 0098          01011 MOVWF  18
01EE 1283          01012 BCF    03,5
01EF 29FE          01013 GOTO   1FE
0000               01014 ....................              case 6: set_uart_speed(57600);  break;
01F0 3015          01015 MOVLW  15
01F1 1683          01016 BSF    03,5
01F2 0099          01017 MOVWF  19
01F3 3026          01018 MOVLW  26
01F4 0098          01019 MOVWF  18
01F5 1283          01020 BCF    03,5
01F6 29FE          01021 GOTO   1FE
0000               01022 ....................              case 7: set_uart_speed(115200); break;
01F7 300A          01023 MOVLW  0A
01F8 1683          01024 BSF    03,5
01F9 0099          01025 MOVWF  19
01FA 3026          01026 MOVLW  26
01FB 0098          01027 MOVWF  18
01FC 1283          01028 BCF    03,5
01FD 29FE          01029 GOTO   1FE
0000               01030 ....................    }
0205 100A          01031 BCF    0A,0
0206 148A          01032 BSF    0A,1
0207 110A          01033 BCF    0A,2
0208 0782          01034 ADDWF  02,F
0209 29C6          01035 GOTO   1C6
020A 29CD          01036 GOTO   1CD
020B 29D4          01037 GOTO   1D4
020C 29DB          01038 GOTO   1DB
020D 29E2          01039 GOTO   1E2
020E 29E9          01040 GOTO   1E9
020F 29F0          01041 GOTO   1F0
0210 29F7          01042 GOTO   1F7
0000               01043 ....................
0000               01044 ....................    set_tris_e(0x07);  // Turn OFF Parral Slave Mode as this affects port D (Address
Bus)
01FE 3007          01045 MOVLW  07
01FF 1683          01046 BSF    03,5
0200 0089          01047 MOVWF  09
0201 1283          01048 BCF    03,5
0202 118A          01049 BCF    0A,3
0203 120A          01050 BCF    0A,4
0204 2B98          01051 GOTO   398
0000               01052 .................... }
0000               01053 ....................
0000               01054 .................... int  ReadArray(char address)
0000               01055 .................... {
0000               01056 ....................    char data;
0000               01057 ....................
0000               01058 ....................    if (address <64) data = block1[address];
0211 3040          01059 MOVLW  40
0212 1683          01060 BSF    03,5
0213 1703          01061 BSF    03,6
0214 0254          01062 SUBWF  54,W
0215 1C03          01063 BTFSS  03,0
0216 2A1A          01064 GOTO   21A
```

```
0217 1283          01065 BCF    03,5
0218 1303          01066 BCF    03,6
0219 2A2B          01067 GOTO   22B
021A 1283          01068 BCF    03,5
021B 1303          01069 BCF    03,6
021C 3028          01070 MOVLW  28
021D 1683          01071 BSF    03,5
021E 1703          01072 BSF    03,6
021F 0754          01073 ADDWF  54,W
0220 1283          01074 BCF    03,5
0221 1303          01075 BCF    03,6
0222 0084          01076 MOVWF  04
0223 1383          01077 BCF    03,7
0224 0800          01078 MOVF   00,W
0225 1683          01079 BSF    03,5
0226 1703          01080 BSF    03,6
0227 00D5          01081 MOVWF  55
0000               01082 ....................    else if(address <128) data = block2[address-64];
0228 1283          01083 BCF    03,5
0229 1303          01084 BCF    03,6
022A 2A86          01085 GOTO   286
022B 3080          01086 MOVLW  80
022C 1683          01087 BSF    03,5
022D 1703          01088 BSF    03,6
022E 0254          01089 SUBWF  54,W
022F 1C03          01090 BTFSS  03,0
0230 2A34          01091 GOTO   234
0231 1283          01092 BCF    03,5
0232 1303          01093 BCF    03,6
0233 2A48          01094 GOTO   248
0234 1283          01095 BCF    03,5
0235 1303          01096 BCF    03,6
0236 3040          01097 MOVLW  40
0237 1683          01098 BSF    03,5
0238 1703          01099 BSF    03,6
0239 0254          01100 SUBWF  54,W
023A 1283          01101 BCF    03,5
023B 1303          01102 BCF    03,6
023C 00F7          01103 MOVWF  77
023D 30A0          01104 MOVLW  A0
023E 0777          01105 ADDWF  77,W
023F 0084          01106 MOVWF  04
0240 1383          01107 BCF    03,7
0241 0800          01108 MOVF   00,W
0242 1683          01109 BSF    03,5
0243 1703          01110 BSF    03,6
0244 00D5          01111 MOVWF  55
0000               01112 ....................    else if(address <192) data = block3[address-128];
0245 1283          01113 BCF    03,5
0246 1303          01114 BCF    03,6
0247 2A86          01115 GOTO   286
0248 30C0          01116 MOVLW  C0
0249 1683          01117 BSF    03,5
024A 1703          01118 BSF    03,6
024B 0254          01119 SUBWF  54,W
024C 1C03          01120 BTFSS  03,0
024D 2A51          01121 GOTO   251
024E 1283          01122 BCF    03,5
024F 1303          01123 BCF    03,6
0250 2A6D          01124 GOTO   26D
0251 1283          01125 BCF    03,5
0252 1303          01126 BCF    03,6
0253 3080          01127 MOVLW  80
0254 1683          01128 BSF    03,5
0255 1703          01129 BSF    03,6
0256 0254          01130 SUBWF  54,W
0257 1283          01131 BCF    03,5
0258 1303          01132 BCF    03,6
0259 00F7          01133 MOVWF  77
025A 3010          01134 MOVLW  10
025B 0777          01135 ADDWF  77,W
025C 00F9          01136 MOVWF  79
025D 3001          01137 MOVLW  01
025E 00FA          01138 MOVWF  7A
025F 0879          01139 MOVF   79,W
0260 1803          01140 BTFSC  03,0
0261 0AFA          01141 INCF   7A,F
0262 0084          01142 MOVWF  04
0263 1383          01143 BCF    03,7
0264 187A          01144 BTFSC  7A,0
0265 1783          01145 BSF    03,7
0266 0800          01146 MOVF   00,W
0267 1683          01147 BSF    03,5
0268 1703          01148 BSF    03,6
0269 00D5          01149 MOVWF  55
0000               01150 ....................    else data = block4[address-192];
026A 1283          01151 BCF    03,5
026B 1303          01152 BCF    03,6
026C 2A86          01153 GOTO   286
026D 30C0          01154 MOVLW  C0
026E 1683          01155 BSF    03,5
026F 1703          01156 BSF    03,6
0270 0254          01157 SUBWF  54,W
0271 1283          01158 BCF    03,5
0272 1303          01159 BCF    03,6
0273 00F7          01160 MOVWF  77
0274 3090          01161 MOVLW  90
0275 0777          01162 ADDWF  77,W
0276 00F9          01163 MOVWF  79
0277 3001          01164 MOVLW  01
0278 00FA          01165 MOVWF  7A
0279 0879          01166 MOVF   79,W
```

```
027A 1803          01167 BTFSC  03,0
027B 0AFA          01168 INCF   7A,F
027C 0084          01169 MOVWF  04
027D 1383          01170 BCF    03,7
027E 187A          01171 BTFSC  7A,0
027F 1783          01172 BSF    03,7
0280 0800          01173 MOVF   00,W
0281 1683          01174 BSF    03,5
0282 1703          01175 BSF    03,6
0283 00D5          01176 MOVWF  55
0284 1283          01177 BCF    03,5
0285 1303          01178 BCF    03,6
0000               01179 ....................
0000               01180 ....................    return data;
0286 1683          01181 BSF    03,5
0287 1703          01182 BSF    03,6
0288 0855          01183 MOVF   55,W
0289 1283          01184 BCF    03,5
028A 1303          01185 BCF    03,6
028B 00F8          01186 MOVWF  78
028C 118A          01187 BCF    0A,3
028D 120A          01188 BCF    0A,4
028E 2BDC          01189 GOTO   3DC
0000               01190 .................... }
0000               01191 ....................
0000               01192 .................... void WriteArray(char address, char data)
0000               01193 .................... {
0000               01194 ....................    if (address <64) block1[address] = data;
009F 3040          01195 MOVLW  40
00A0 025E          01196 SUBWF  5E,W
00A1 1C03          01197 BTFSS  03,0
00A2 28A6          01198 GOTO   0A6
00A3 1283          01199 BCF    03,5
00A4 1303          01200 BCF    03,6
00A5 28B7          01201 GOTO   0B7
00A6 1283          01202 BCF    03,5
00A7 1303          01203 BCF    03,6
00A8 3028          01204 MOVLW  28
00A9 1683          01205 BSF    03,5
00AA 1703          01206 BSF    03,6
00AB 075E          01207 ADDWF  5E,W
00AC 1283          01208 BCF    03,5
00AD 1303          01209 BCF    03,6
00AE 0084          01210 MOVWF  04
00AF 1383          01211 BCF    03,7
00B0 1683          01212 BSF    03,5
00B1 1703          01213 BSF    03,6
00B2 085F          01214 MOVF   5F,W
00B3 0080          01215 MOVWF  00
0000               01216 ....................    else if(address <128) block2[address-64] = data;
00B4 1283          01217 BCF    03,5
00B5 1303          01218 BCF    03,6
00B6 2912          01219 GOTO   112
00B7 3080          01220 MOVLW  80
00B8 1683          01221 BSF    03,5
00B9 1703          01222 BSF    03,6
00BA 025E          01223 SUBWF  5E,W
00BB 1C03          01224 BTFSS  03,0
00BC 28C0          01225 GOTO   0C0
00BD 1283          01226 BCF    03,5
00BE 1303          01227 BCF    03,6
00BF 28D4          01228 GOTO   0D4
00C0 1283          01229 BCF    03,5
00C1 1303          01230 BCF    03,6
00C2 3040          01231 MOVLW  40
00C3 1683          01232 BSF    03,5
00C4 1703          01233 BSF    03,6
00C5 025E          01234 SUBWF  5E,W
00C6 1283          01235 BCF    03,5
00C7 1303          01236 BCF    03,6
00C8 00F7          01237 MOVWF  77
00C9 30A0          01238 MOVLW  A0
00CA 0777          01239 ADDWF  77,W
00CB 0084          01240 MOVWF  04
00CC 1383          01241 BCF    03,7
00CD 1683          01242 BSF    03,5
00CE 1703          01243 BSF    03,6
00CF 085F          01244 MOVF   5F,W
00D0 0080          01245 MOVWF  00
0000               01246 ....................    else if(address <192) block3[address-128] = data;
00D1 1283          01247 BCF    03,5
00D2 1303          01248 BCF    03,6
00D3 2912          01249 GOTO   112
00D4 30C0          01250 MOVLW  C0
00D5 1683          01251 BSF    03,5
00D6 1703          01252 BSF    03,6
00D7 025E          01253 SUBWF  5E,W
00D8 1C03          01254 BTFSS  03,0
00D9 28DD          01255 GOTO   0DD
00DA 1283          01256 BCF    03,5
00DB 1303          01257 BCF    03,6
00DC 28F9          01258 GOTO   0F9
00DD 1283          01259 BCF    03,5
00DE 1303          01260 BCF    03,6
00DF 3080          01261 MOVLW  80
00E0 1683          01262 BSF    03,5
00E1 1703          01263 BSF    03,6
00E2 025E          01264 SUBWF  5E,W
00E3 1283          01265 BCF    03,5
00E4 1303          01266 BCF    03,6
00E5 00F7          01267 MOVWF  77
00E6 3010          01268 MOVLW  10
```

```
00E7 0777          01269 ADDWF  77,W
00E8 00F9          01270 MOVWF  79
00E9 3001          01271 MOVLW  01
00EA 00FA          01272 MOVWF  7A
00EB 0879          01273 MOVF   79,W
00EC 1803          01274 BTFSC  03,0
00ED 0AFA          01275 INCF   7A,F
00EE 0084          01276 MOVWF  04
00EF 1383          01277 BCF    03,7
00F0 187A          01278 BTFSC  7A,0
00F1 1783          01279 BSF    03,7
00F2 1683          01280 BSF    03,5
00F3 1703          01281 BSF    03,6
00F4 085F          01282 MOVF   5F,W
00F5 0080          01283 MOVWF  00
0000               01284 ....................    else block4[address-192] = data;
00F6 1283          01285 BCF    03,5
00F7 1303          01286 BCF    03,6
00F8 2912          01287 GOTO   112
00F9 30C0          01288 MOVLW  C0
00FA 1683          01289 BSF    03,5
00FB 1703          01290 BSF    03,6
00FC 025E          01291 SUBWF  5E,W
00FD 1283          01292 BCF    03,5
00FE 1303          01293 BCF    03,6
00FF 00F7          01294 MOVWF  77
0100 3090          01295 MOVLW  90
0101 0777          01296 ADDWF  77,W
0102 00F9          01297 MOVWF  79
0103 3001          01298 MOVLW  01
0104 00FA          01299 MOVWF  7A
0105 0879          01300 MOVF   79,W
0106 1803          01301 BTFSC  03,0
0107 0AFA          01302 INCF   7A,F
0108 0084          01303 MOVWF  04
0109 1383          01304 BCF    03,7
010A 187A          01305 BTFSC  7A,0
010B 1783          01306 BSF    03,7
010C 1683          01307 BSF    03,5
010D 1703          01308 BSF    03,6
010E 085F          01309 MOVF   5F,W
010F 0080          01310 MOVWF  00
0110 1283          01311 BCF    03,5
0111 1303          01312 BCF    03,6
0000               01313 ..................... }
0000               01314 ....................
0000               01315 ....................
0000               01316 ..................... void SetDAC(char X, char Y)
0000               01317 ..................... {
0000               01318 ....................    // by CKM 14/04/2002
0000               01319 ....................    set_tris_b(0x00); // Data bus output
028F 3000          01320 MOVLW  00
0290 0066          01321 TRIS   6
0000               01322 ....................
0000               01323 ....................    DATA_BUS = X;
0291 1683          01324 BSF    03,5
0292 1703          01325 BSF    03,6
0293 0854          01326 MOVF   54,W
0294 1283          01327 BCF    03,5
0295 1303          01328 BCF    03,6
0296 0086          01329 MOVWF  06
0000               01330 ....................    DAC_SELECT = 0;               // Selected DAC A
0297 1185          01331 BCF    05,3
0000               01332 ....................    DAC_ENABLE = 0;               // Enable DAC
0298 1105          01333 BCF    05,2
0000               01334 ....................
0000               01335 ....................    delay_cycles(1);    // 200nS delay
0299 0000          01336 NOP
0000               01337 ....................
0000               01338 ....................    DAC_ENABLE = 1;     // Disable DAC
029A 1505          01339 BSF    05,2
0000               01340 ....................    DATA_BUS = Y;
029B 1683          01341 BSF    03,5
029C 1703          01342 BSF    03,6
029D 0855          01343 MOVF   55,W
029E 1283          01344 BCF    03,5
029F 1303          01345 BCF    03,6
02A0 0086          01346 MOVWF  06
0000               01347 ....................    DAC_SELECT = 1;     // Select DAC B
02A1 1585          01348 BSF    05,3
0000               01349 ....................    DAC_ENABLE = 0;               // Enable DAC
02A2 1105          01350 BCF    05,2
0000               01351 ....................
0000               01352 ....................    delay_cycles(1);    // 200nS delay
02A3 0000          01353 NOP
0000               01354 ....................    DAC_ENABLE = 1;               // Disable DAC
02A4 1505          01355 BSF    05,2
02A5 118A          01356 BCF    0A,3
02A6 120A          01357 BCF    0A,4
02A7 2BEB          01358 GOTO   3EB
0000               01359 ..................... }
0000               01360 ....................
0000               01361 ..................... Update7SEG()
0000               01362 ..................... {
0000               01363 ....................    int i;
0000               01364 ....................    char temp1;
0000               01365 ....................
0000               01366 ....................    /* Carried out in 5 stepss, to reduce the affect caused to the waveform, e.g.
between
0000               01367 ....................            each step the waveform is redraw, for 50Hz each step must execute in less
than
0000               01368 ....................            78uS */
```

```
0000                01369 ....................
0000                01370 ....................    switch(SEG_State)
02A8 0873           01371 MOVF   73,W
02A9 00F7           01372 MOVWF  77
02AA 0877           01373 MOVF   77,W
02AB 1903           01374 BTFSC  03,2
02AC 2ABE           01375 GOTO   2BE
02AD 3001           01376 MOVLW  01
02AE 0277           01377 SUBWF  77,W
02AF 1903           01378 BTFSC  03,2
02B0 2AEC           01379 GOTO   2EC
02B1 3002           01380 MOVLW  02
02B2 0277           01381 SUBWF  77,W
02B3 1903           01382 BTFSC  03,2
02B4 2B06           01383 GOTO   306
02B5 3003           01384 MOVLW  03
02B6 0277           01385 SUBWF  77,W
02B7 1903           01386 BTFSC  03,2
02B8 2B20           01387 GOTO   320
02B9 3004           01388 MOVLW  04
02BA 0277           01389 SUBWF  77,W
02BB 1903           01390 BTFSC  03,2
02BC 2B37           01391 GOTO   337
02BD 2B63           01392 GOTO   363
0000                01393 ....................    {
0000                01394 ....................
0000                01395 ....................              case 0:   // Cal bpm
0000                01396 ....................
0000                01397 ....................                        /** ECG bpm cacluated over 10 seconds, hence on power up it
will take
0000                01398 ....................                            10 seonds to display correct bpm value
*/
0000                01399 ....................                        BeatsPer10Sec = 0;
02BE 01FD           01400 CLRF   7D
0000                01401 ....................                        for (i=0; i<10; i++) BeatsPer10Sec = BeatsPer10Sec +
bpsArray[i];
02BF 1683           01402 BSF    03,5
02C0 1703           01403 BSF    03,6
02C1 01D4           01404 CLRF   54
02C2 1283           01405 BCF    03,5
02C3 1303           01406 BCF    03,6
02C4 300A           01407 MOVLW  0A
02C5 1683           01408 BSF    03,5
02C6 1703           01409 BSF    03,6
02C7 0254           01410 SUBWF  54,W
02C8 1C03           01411 BTFSS  03,0
02C9 2ACD           01412 GOTO   2CD
02CA 1283           01413 BCF    03,5
02CB 1303           01414 BCF    03,6
02CC 2ADF           01415 GOTO   2DF
02CD 1283           01416 BCF    03,5
02CE 1303           01417 BCF    03,6
02CF 30E0           01418 MOVLW  E0
02D0 1683           01419 BSF    03,5
02D1 1703           01420 BSF    03,6
02D2 0754           01421 ADDWF  54,W
02D3 1283           01422 BCF    03,5
02D4 1303           01423 BCF    03,6
02D5 0084           01424 MOVWF  04
02D6 1383           01425 BCF    03,7
02D7 0800           01426 MOVF   00,W
02D8 07FD           01427 ADDWF  7D,F
02D9 1683           01428 BSF    03,5
02DA 1703           01429 BSF    03,6
02DB 0AD4           01430 INCF   54,F
02DC 1283           01431 BCF    03,5
02DD 1303           01432 BCF    03,6
02DE 2AC4           01433 GOTO   2C4
0000                01434 ....................
0000                01435 ....................                        /** BeatsPer10Sec * 6 for bps, add 6 times is much faster **/
0000                01436 ....................                        m_bpm = BeatsPer10Sec + BeatsPer10Sec + BeatsPer10Sec +
BeatsPer10Sec + BeatsPer10Sec + BeatsPer10Sec;
02DF 087D           01437 MOVF   7D,W
02E0 077D           01438 ADDWF  7D,W
02E1 077D           01439 ADDWF  7D,W
02E2 077D           01440 ADDWF  7D,W
02E3 077D           01441 ADDWF  7D,W
02E4 077D           01442 ADDWF  7D,W
02E5 00F2           01443 MOVWF  72
0000                01444 ....................
0000                01445 ....................                        unit = 0;
02E6 01F4           01446 CLRF   74
0000                01447 ....................                        ten = 0;
02E7 01F5           01448 CLRF   75
0000                01449 ....................                        hundred = 0;
02E8 01F6           01450 CLRF   76
0000                01451 ....................
0000                01452 ....................                        SEG_State = 1;
02E9 3001           01453 MOVLW  01
02EA 00F3           01454 MOVWF  73
0000                01455 ....................                        break;
02EB 2B65           01456 GOTO   365
0000                01457 ....................
0000                01458 ....................              // Steps 1 to 5 Convert Binary number into it hunderds, tens, units */
0000                01459 ....................              case 1:
0000                01460 ....................                        temp1 = m_bpm;
02EC 0872           01461 MOVF   72,W
02ED 1683           01462 BSF    03,5
02EE 1703           01463 BSF    03,6
02EF 00D5           01464 MOVWF  55
0000                01465 ....................                        m_bpm = m_bpm - 100;
02F0 3064           01466 MOVLW  64
```

```
02F1 1283          01467 BCF    03,5
02F2 1303          01468 BCF    03,6
02F3 02F2          01469 SUBWF  72,F
0000               01470 ....................                        if(m_bpm > temp1)
02F4 0872          01471 MOVF   72,W
02F5 1683          01472 BSF    03,5
02F6 1703          01473 BSF    03,6
02F7 0255          01474 SUBWF  55,W
02F8 1C03          01475 BTFSS  03,0
02F9 2AFD          01476 GOTO   2FD
02FA 1283          01477 BCF    03,5
02FB 1303          01478 BCF    03,6
02FC 2B04          01479 GOTO   304
02FD 1283          01480 BCF    03,5
02FE 1303          01481 BCF    03,6
0000               01482 ....................                        {
0000               01483 ....................                                  SEG_State = 2;
02FF 3002          01484 MOVLW  02
0300 00F3          01485 MOVWF  73
0000               01486 ....................                                  m_bpm = m_bpm +100;
0301 3064          01487 MOVLW  64
0302 07F2          01488 ADDWF  72,F
0000               01489 ....................                        }
0000               01490 ....................                        else hundred++;
0303 2B05          01491 GOTO   305
0304 0AF6          01492 INCF   76,F
0000               01493 ....................
0000               01494 ....................                        break;
0305 2B65          01495 GOTO   365
0000               01496 ....................
0000               01497 ....................              case 2:
0000               01498 ....................                        temp1 = m_bpm;
0306 0872          01499 MOVF   72,W
0307 1683          01500 BSF    03,5
0308 1703          01501 BSF    03,6
0309 00D5          01502 MOVWF  55
0000               01503 ....................                        m_bpm = m_bpm - 10;
030A 300A          01504 MOVLW  0A
030B 1283          01505 BCF    03,5
030C 1303          01506 BCF    03,6
030D 02F2          01507 SUBWF  72,F
0000               01508 ....................                        if(m_bpm > temp1)
030E 0872          01509 MOVF   72,W
030F 1683          01510 BSF    03,5
0310 1703          01511 BSF    03,6
0311 0255          01512 SUBWF  55,W
0312 1C03          01513 BTFSS  03,0
0313 2B17          01514 GOTO   317
0314 1283          01515 BCF    03,5
0315 1303          01516 BCF    03,6
0316 2B1E          01517 GOTO   31E
0317 1283          01518 BCF    03,5
0318 1303          01519 BCF    03,6
0000               01520 ....................                        {
0000               01521 ....................                                  SEG_state = 3;
0319 3003          01522 MOVLW  03
031A 00F3          01523 MOVWF  73
0000               01524 ....................                                  m_bpm = m_bpm +10;
031B 300A          01525 MOVLW  0A
031C 07F2          01526 ADDWF  72,F
0000               01527 ....................                        }
0000               01528 ....................                        else ten++;
031D 2B1F          01529 GOTO   31F
031E 0AF5          01530 INCF   75,F
0000               01531 ....................
0000               01532 ....................                        break;
031F 2B65          01533 GOTO   365
0000               01534 ....................
0000               01535 ....................              case 3:
0000               01536 ....................                        temp1 = m_bpm;
0320 0872          01537 MOVF   72,W
0321 1683          01538 BSF    03,5
0322 1703          01539 BSF    03,6
0323 00D5          01540 MOVWF  55
0000               01541 ....................                        m_bpm--;
0324 1283          01542 BCF    03,5
0325 1303          01543 BCF    03,6
0326 03F2          01544 DECF   72,F
0000               01545 ....................                        if(m_bpm > temp1)
0327 0872          01546 MOVF   72,W
0328 1683          01547 BSF    03,5
0329 1703          01548 BSF    03,6
032A 0255          01549 SUBWF  55,W
032B 1C03          01550 BTFSS  03,0
032C 2B30          01551 GOTO   330
032D 1283          01552 BCF    03,5
032E 1303          01553 BCF    03,6
032F 2B35          01554 GOTO   335
0330 1283          01555 BCF    03,5
0331 1303          01556 BCF    03,6
0000               01557 ....................                        {
0000               01558 ....................                                  SEG_state = 4;
0332 3004          01559 MOVLW  04
0333 00F3          01560 MOVWF  73
0000               01561 ....................                        }
0000               01562 ....................                        else unit++;
0334 2B36          01563 GOTO   336
0335 0AF4          01564 INCF   74,F
0000               01565 ....................                        break;
0336 2B65          01566 GOTO   365
0000               01567 ....................
0000               01568 ....................              case 4: // Update Display
```

```
0000                ..................                        DAC_ENABLE = 1;     // Disable DAC
0337 1505           BSF    05,2
0000                ..................                        DAC_SELECT = 0;
0338 1185           BCF    05,3
0000                ..................                        RAM_OE       = 1;   // Disable RAM
0339 1605           BSF    05,4
0000                ..................                        RAM_WRITE  = 1;
033A 1685           BSF    05,5
0000                ..................
0000                ..................                        // ENABLE UNITS DISPLAY
0000                ..................                        SEG1_EN    = 1;
033B 1487           BSF    07,1
0000                ..................                        SEG2_EN    = 1;
033C 1687           BSF    07,5
0000                ..................                        SEG3_EN    = 0;
033D 1207           BCF    07,4
0000                ..................
0000                ..................                        DATA_BUS   = SevenSeg[Unit];
033E 0874           MOVF   74,W
033F 203D           CALL   03D
0340 00F8           MOVWF  78
0341 0878           MOVF   78,W
0342 0086           MOVWF  06
0000                ..................
0000                ..................                        SEG_CLOCK = 0;
0343 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
0344 1507           BSF    07,2
0000                ..................                        SEG_CLOCK = 0;
0345 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
0346 1507           BSF    07,2
0000                ..................
0000                ..................                        // Enable TENS DISPLAY
0000                ..................                        SEG1_EN    = 1;
0347 1487           BSF    07,1
0000                ..................                        SEG2_EN    = 0;
0348 1287           BCF    07,5
0000                ..................                        SEG3_EN    = 1;
0349 1607           BSF    07,4
0000                ..................
0000                ..................                        DATA_BUS   = SevenSeg[Ten];
034A 0875           MOVF   75,W
034B 203D           CALL   03D
034C 00F8           MOVWF  78
034D 0878           MOVF   78,W
034E 0086           MOVWF  06
0000                ..................
0000                ..................                        SEG_CLOCK = 0;
034F 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
0350 1507           BSF    07,2
0000                ..................                        SEG_CLOCK = 0;
0351 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
0352 1507           BSF    07,2
0000                ..................
0000                ..................                        // ENABLE Hundred DISPLAY
0000                ..................                        SEG1_EN    = 0;
0353 1087           BCF    07,1
0000                ..................                        SEG2_EN    = 1;
0354 1687           BSF    07,5
0000                ..................                        SEG3_EN    = 1;
0355 1607           BSF    07,4
0000                ..................
0000                ..................                        DATA_BUS   = SevenSeg[Hundred];
0356 0876           MOVF   76,W
0357 203D           CALL   03D
0358 00F8           MOVWF  78
0359 0878           MOVF   78,W
035A 0086           MOVWF  06
0000                ..................
0000                ..................                        SEG_CLOCK = 0;
035B 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
035C 1507           BSF    07,2
0000                ..................                        SEG_CLOCK = 0;
035D 1107           BCF    07,2
0000                ..................                        SEG_CLOCK = 1;
035E 1507           BSF    07,2
0000                ..................
0000                ..................                        SEG1_EN = 1;        // Disable display
035F 1487           BSF    07,1
0000                ..................
0000                ..................                        SEG_State = 0;
0360 01F3           CLRF   73
0000                ..................                        UpdateBPM = FALSE;
0361 01FC           CLRF   7C
0000                ..................                        break;
0362 2B65           GOTO   365
0000                ..................
0000                ..................              default:
0000                ..................
0000                ..................                        SEG_State = 0;
0363 01F3           CLRF   73
0000                ..................                        break;
0364 2B65           GOTO   365
0000                ..................    }
0365 118A           BCF    0A,3
0366 120A           BCF    0A,4
0367 2C1E           GOTO   41E
0000                .................. }
```

```
0000                01671 ....................
0000                01672 ....................
0000                01673 ....................

SYMBOL TABLE
  LABEL                             VALUE

PORTA                             00000005
DAC_ENABLE                        00000005
DAC_SELECT                        00000005
RAM_OE                            00000005
RAM_WRITE                         00000005
PUSH_BUTT                         00000005
DATA_BUS                          00000006
PORTC                             00000007
BUZZER                            00000007
SEG1_EN                           00000007
SEG_CLOCK                         00000007
SEG2_EN                           00000007
SEG3_EN                           00000007
PSP_DATA                          00000008
ADDRESS_BUS                       00000008
PORTE                             00000009
TIMER_1_LOW                       0000000E
TIMER_1_HIGH                      0000000F
TIMER_2                           00000011
CCP_1                             00000015
CCP_1_LOW                         00000015
CCP_1_HIGH                        00000016
CCP_2                             0000001B
CCP_2_LOW                         0000001B
CCP_2_HIGH                        0000001C
BLOCK1                            00000028
PSTARTOFARRAY                     00000068
PSCANPOS                          00000069
M_BUZZERONTIME                    0000006A
ECG_MAX                           0000006B
ECG_PEAK_COUNTER                  0000006D
ECG_PEAK_COUNTDELAY               0000006E
INT_COUNT0                        0000006F
SEG_UPDATE                        00000070
SEG_AVERAGE                       00000071
M_BPM                             00000072
SEG_STATE                         00000073
UNIT                              00000074
TEN                               00000075
HUNDRED                           00000076
_RETURN_                          00000078
UPDATEBPM                         0000007C
BEATSPER10SEC                     0000007D
BLOCK2                            000000A0
BPSARRAY                          000000E0
BLOCK3                            00000110
BLOCK4                            00000190
MAIN.I                            000001D0
MAIN.DATA                         000001D1
MAIN.ADDRESS                      000001D2
MAIN.BFORWARD                     000001D3
READARRAY.ADDRESS                 000001D4
SETDAC.X                          000001D4
UPDATE7SEG.I                      000001D4
SETDAC.Y                          000001D5
READARRAY.DATA                    000001D5
UPDATE7SEG.TEMP1                  000001D5
TIMER2_ISR.ADCVALUE16             000001D9
TIMER2_ISR.ADCHI                  000001DB
TIMER2_ISR.ADCLO                  000001DC
TIMER2_ISR.ADCVALUE8              000001DD
WRITEARRAY.ADDRESS                000001DE
WRITEARRAY.DATA                   000001DF
SEVENSEG                          0000003D
TIMER0_ISR                        0000004B
TIMER2_ISR                        0000006A
MAIN                              00000368
SETBAUDRATE                       000001BB
READARRAY                         00000211
SETDAC                            0000028F
UPDATE7SEG                        000002A8

MEMORY USAGE
```

A2. FINAL YEAR PROJECT SUMMARY

Low cost PC-based digital real-time / storage oscilloscope.

This project achieves the same functionality as a traditional oscilloscope, using a PIC microcontroller for data acquisition (including appropriate analogue circuitry) which transfers the data to the PC (via RS232). A Microsoft Windows based software application will then display the waveform as it would appear on a traditional CRT oscilloscope. This software application has additional features not present on a traditional oscilloscope (e.g. printing / saving waveforms) with greater flexibly as additional features can be added as their developed without the need for new hardware.

A2.1. Frame Structure: Real-Time Mode

Two-bytes are sent per reading.

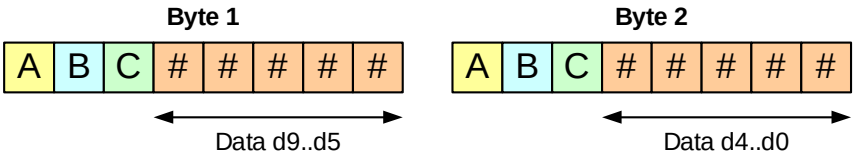


Figure A2.1a. Frame structure: Real Time.

B	C	
0	0	Channel 1
0	1	Channel 2
1	0	Channel 3
1	1	Channel 4

If A = 0, byte 1 which contains data d9...d5 (Upper 5-bits of ADC reading).
If A = 1, byte 2 which contains data d4…d0 (Lower 5-bits of ADC reading).

B & C specify what channel the data is for.

A2.2. The Scope Program

The scope program use in “Real-Time Scroll” mode can be used to view ECG signal in real-time.

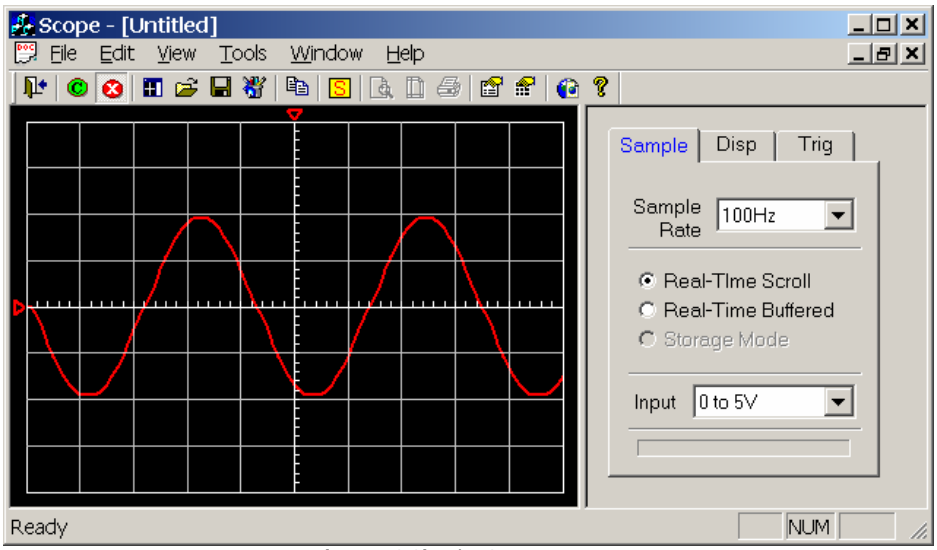


Figure A2.2b. The Scope Program