VitalStatis Medical Solutions
School of Engineering Science
Simon Fraser University
Burnaby, BC, V5A 1S6
vitalstatis-med@sfu.ca

October 31, 2002

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, BC, V5A 1S6

Re: ENSC 340 Project Palm™ Bio-Reader Design Specification

Dear Dr. Rawicz,

Attached you will find VitalStatis' *Palm™ Bio-Reader Design Specification* which illustrates the design methods for the required functions as stated in the *Palm™ Bio-Reader Functional Specification*. The methods of design for a prototype system that will display EKG readouts as well as breathing rate for the monitoring of hospitalized patients is presented. This document will focus on the design of the prototype only and the issues concerning a production model will be discussed at a later time.

The attached document closely follows the sections defined in the functional specifications for easy referral. However, the requirements for the production model will not be discussed.

Please feel free to contact us should you have any questions, comments or concerns. We can be reached by email through vitalstatis-med@sfu.ca or by phoning our CEO at 604-274-1888 (home) or 604-818-7899 (cell). Thank you for your time.

Sincerely,

See-Ho Tsang

See-Ho Tsang
CEO
VitalStatis Medical Solutions

Enclosure: Palm ™ Bio-Reader Design Specification

Copyright © 2002, VitalStatis Medical Solutions

VitalStatis Medical Solutions

Palm™ Bio-Reader Design Specification

Project Team: See-Ho Tsang, Bob Wai, Cory Jung, Jason Yu, James Hu, David Poon

Contact Email: vitalstatis-med@sfu.ca

Submitted to: Dr. Andrew Rawicz — ENSC 340
Steve Whitmore — ENSC 305
School of Engineering Science
Simon Fraser University

Date Issued: October 31, 2002

Revision: 1.0

Abstract

The design specification outlines the implementation method for our project, VitalChart, as specified in the functional specification. For the design specification, only the core functionalities pertaining to the prototype are discussed. The prototype system consists of a hardware module that clips on to the serial port of a Palm™ m500 series personal digital assistant (PDA) as well as a software package containing a graphical user interface (GUI). The hardware module is designed to capture biofeedback readings from a patient via electrocardiogram (EKG) electrodes. This data is collected using a differential amplifier circuit and is sampled using the on-board analog to digital converter on a PIC16F873 microcontroller. In addition to the EKG readings, the hardware module also collects breathing rate data using a ADXL202E dual-axis accelerometer. Once the breathing data is collected, the microcontroller will calculate a relative breathing rate. The biofeedback data is then sent to the Palm™ Operating System (OS) via a MAX232A RS-232 Driver.

On the Palm™ OS side, the software unpacks the serial data sent by the hardware interface. The data is separated into EKG data and breathing rate by the graphics engine which also normalizes and computes the heart rate for display.

Prior to production, a meticulous test plan, including the consideration of international standards, will be implemented to ensure good quality and performance of our product.

Table of Contents

List of Figures

Glossary

A/D	Analog to digital conversion, conversion of continuous analog signals into digital binary format
API	Application Programming Interface, programming interface by which applications can access the operating system and system resources
Baud rate	Measurement of speed in data transmission, equals one bit per second
Biofeedback	Technique of using monitoring devices to furnish information regarding an autonomic bodily function, such as heart rate or blood pressure
bps	Bits per second, unit of measurement of baud rate
Common-mode voltage	The average voltage between two signals
DSP	Digital signal processing, computer manipulation of analog signals that have been converted to digital form
EIA	Electronic Industries Alliance
EKG	Electrocardiogram, a graphical record of the cardiac cycle
ESD	Electrostatic discharge, also called static electricity
FLASH	A kind of non-volatile storage device which can be erased in blocks or entire chip
Graffiti™	Palm™ OS writing recognition software.
GUI	Graphical user interface, the screens of the software that the user interacts with
I/O	Input/Output
Multiplex	Merging of two or more signals for transmission on the same wire
OS	Operating system, software to interface between application and hardware
Packet	A block of data bundled together in a pre-defined sequence
Parity	The even or odd quality of the number of 1's or 0's in a binary code
PCB	Printed circuit board, a thin board to which electronic components are fixed by solder
PDA	Personal digital assistant, also called handheld computers
Pixel	Basic unit of image composition on a display
PRC	Palm™ Resource file format, file format used for Palm™ PDA applications
Protocol	Standard procedure for regulating data transmission
QRS	QRS complex, name of the electrocardiogram waveform. The various peaks and troughs of a waveform are named P, Q, R, S, and T
RAM	Random Access Memory, a kind of storage device that can be stored or accessed in any order
RS-232	A communication interface standard, also named EIA-232
Stylus	Pen input device of the Palm™ handheld computer
TIA	Telecommunications Industry Association
TTL	Transistor/Transistor Logic, a common semiconductor technology for building discrete digital logic integrated circuit
USART	Universal Synchronous/Asynchronous Receiver/Transmitter,
USB	Universal Serial Bus, an external peripheral standard for communication between a computer and peripheral devices

1. Introduction

This document contains the design specification for the VitalChart Bio-Reader as proposed by VitalStatis Medical Solutions. The details for the functional specification can be found in the document *Palm™ Bio-Reader Functional Specification*, issued October 17, 2002.

This document describes our current design of the VitalChart system in detail. The design specification of the overall system and each component — EKG unit, respiratory measuring device, hardware interface, and Palm™ handheld software, are outlined. The system overview contains the design of overall system, including design of system power supply, and the packaging of the product. For each hardware component, design of the circuit, physical and performance parameters, as well as costs are included. The hardware interface section illustrates design of communications procedure, data transmission format, and microcontroller firmware. Lastly, the software section details how the VitalChart user application will be implemented on the Palm™ handheld.

Our team of engineers will carry out the design during the next two months. The design will be carefully tested and modified as necessary. Any modification will be noted in the post-mortem. Any ideal requirements listed in the functional specification we implement will also be described in the post-mortem.

2. System Overview

VitalChart is a Palm™-based bio-reader. It is able to measure a patient's EKG signal and respiratory rate utilizing a hardware attachment to a Palm™ PDA. Electrodes and sensors are attached to the patient's body and the signals are sent through the hardware to the Palm™ PDA, where they are displayed on the screen by the software. The system can be divided into three main components: measurement hardware, interface hardware, and Palm™-side software. The system block diagram in Figure 1 illustrates how the components in the system interact with each other.

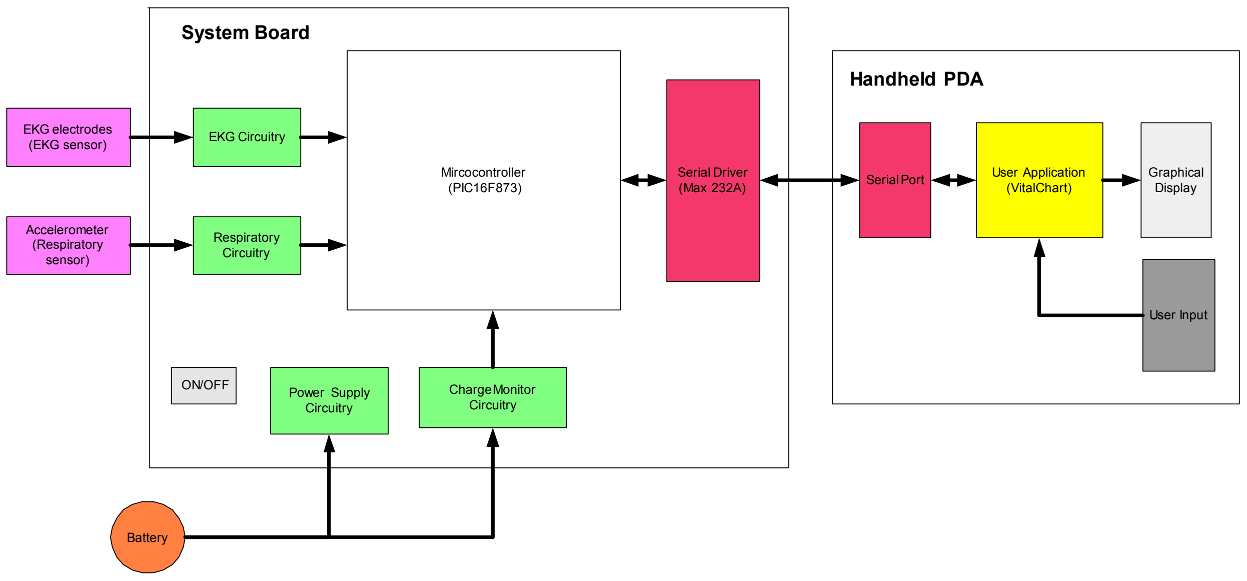


Figure 1: System block diagram

Input to the board is provided by the following components:
- Analog EKG sensor signal
- Analog respiratory sensor signal
- Battery terminals
- On/Off button

Output from the board is:
- USART serial signal

The EKG electrodes and respiratory rate sensor continuously measure heart and respiratory signals. The data is captured by the microcontroller, a PIC16F873A, and sent periodically to the Palm™ PDA via the MAX232A communications driver. Once the data is acquired by the PDA, VitalChart software interprets the data and displays the EKG signal and respiratory rate in a GUI. A display of the patient's EKG signal, pulse rate, and respiratory rate are arrayed in an easily read format for quick diagnosis of the patient's condition. The entire system is powered by a 9V battery.

3. Overall System Design

The specifications stated in this section apply to the overall system. Design specification pertaining to individual components of the system will be described in later sections of the document.

3.1. Handheld PDA

The handheld PDA we will use for VitalChart is a Palm™ m500 PDA available on the market. It is running the Palm™ OS version 4.01, has 8MB of memory, and has a working 16-pin serial connector which can be used to receive data from our hardware module.

3.2. System Power

The hardware module, consisting of the EKG unit, respiratory unit, and hardware interface, will be powered by a 9V battery. An on/off switch will be implemented to control power to the hardware module. In the case where a dual supply must be used to power a circuit, a voltage divider will be used to provide 4.5V and –4.5V.

3.3. Package

The hardware attachment will be designed to be a rigid structure that will "clip" into the Palm™ serial port. Materials under consideration include Plexiglas and plastics. The package will not have sharp edges and should be aesthetically pleasing. Ideally, the dimensions will not exceed 10cm in width, 5cm in length, and 1cm in height.

3.4. Environmental Considerations

Since VitalChart is to be used in a wide variety of environments, component selection is of primary importance. All components used in building the system will be able to operate from 0 to 70°C and humidity range of 0% to 98% non-condensing to ensure that VitalChart is guaranteed to work at least under ideal conditions of room temperature.

4. Electrocardiogram Design

This section will describe the design of the EKG unit.

4.1. General

The EKG unit will detect the voltage potential between two medical pads, then filter and amplify the signal. This output signal is the EKG waveform to be read by the microcontroller. The EKG signal will be sent to an A/D converter on the microcontroller which will use minimum and maximum reference voltages of 0V and 4.5V respectively. Thus, the output signal from the EKG unit must be an analog signal from 0 to 4.5V. This section will describe in greater detail how this will be accomplished.

4.2. Physical

The EKG circuit requires three connections to the patient, a left, right, and ground lead. Furthermore, we will use leads whose shields can be grounded to reduce the amount of noise that will be picked up by the leads. Since the EKG unit should be portable, the size (should be less than 10cm in length, 10cm in width, and 5cm in height) and weight (should be less than 300grams) of the EKG unit will be minimized by implementing the EKG circuit on a PCB.

4.3. Performance

The EKG unit will output an analog signal from 0 to 4.5V by measuring the voltage potential between two medical pads. The location of the medical pads is shown in Figure 2.

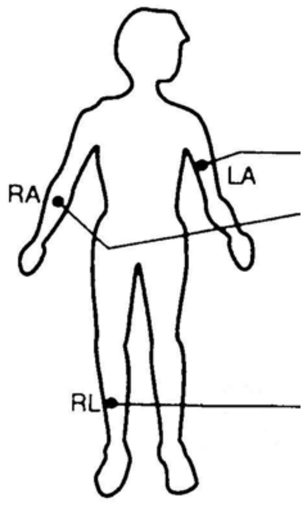


Figure 2: Placement of medical pads

The potential between the right arm and left arm will be amplified and filtered to produce the desired signal from 0 to 4.5V. The circuit shown in Figure 3 will be utilized for amplification and filtering.

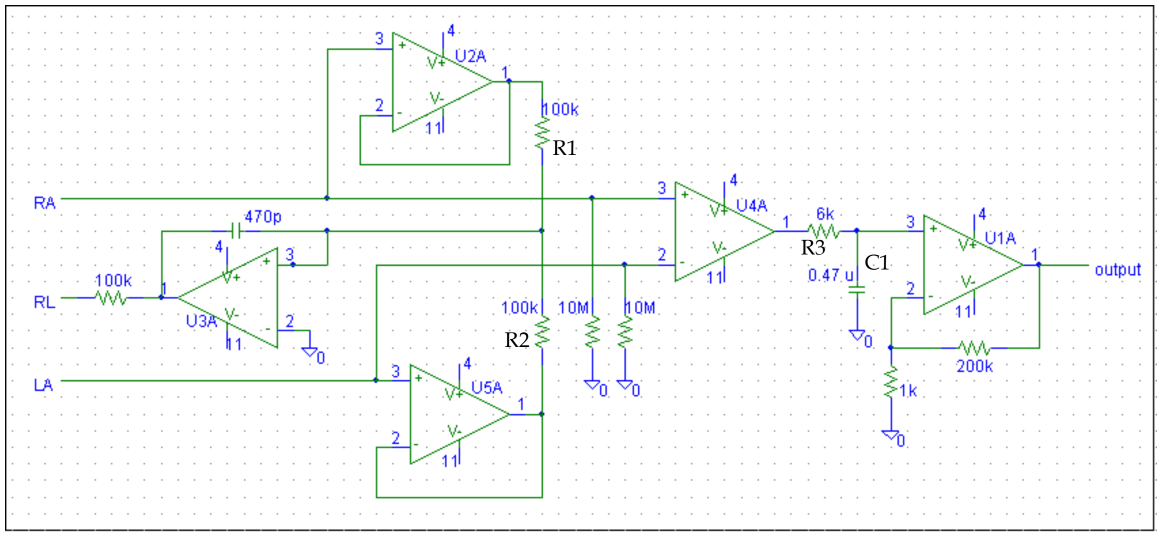


Figure 3: EKG circuit schematic

U4A in Figure 3 is an instrumentation amplifier with a set gain of 10. U2A and U5A are voltage followers which have unity voltage gain and prevent R1 and R2 from loading the electrode connections. The junction of R1 and R2 provides the average of the voltages from the RA and RL electrodes. The average of the voltages represents the common-mode voltage that DC offsets our analog output. The common-mode voltage is fed to U3A and the amplifier generates a current which nearly cancels out the current producing the common-voltage. This means that the active ground circuit will minimize the current through the user's body. The low-pass filter based on R3 and C1 filters out noise and provides adequate low-frequency response. The amplifier U1A has a gain of 201 to scale the output to be 4.5V peak-to-peak. Resisters may be added to offset our signal so the minimum voltage of our analog output is non-negative.

4.4. Cost

The cost of the EKG unit will be kept below $50 by striving to reduce the cost of components without sacrificing the performance of the unit.

5. Respiratory Measuring Device Design

This section will describe the design of the respiratory measuring device.

5.1. General

The respiratory measuring device detects the breathing rate with the use of an accelerometer. The sensor output signal will be fed into the microcontroller through one of the analog channels (AN1) for analog to digital signal conversion. The output signal of the sensor must have minimum and maximum voltages of 0V and 4.5V respectively. A detailed description of the respiratory measuring device design will be provided in this section.

5.2. Physical

The respiratory circuit will be attached to the patient's abdomen. We will use leads with shields that can be grounded to reduce the amount of noise that will be picked up by the them. Since the respiratory unit should be portable, the size (should be less than 10cm in length, 10cm in width, and 5cm in height) and weight (should be less than 150grams) of the respiratory unit will be minimized by implementing the respiratory circuit on a PCB. Ideally, the weight of the respiratory sensor should be less than 10grams.

5.3. Performance

The breathing rate is detected with an accelerometer sensor (ADXL202E). The sensor measures the acceleration created by expansion and contraction of the patient's chest or abdomen in every breath. Since the breathing rate is normally under 40 breaths per minute, the sensor has to be able to detect an input signal with a frequency of less than 1Hz. Filter circuitry will be used to remove white noise and DC offset in the sensor output signal. Amplifying circuitry will also be used to enhance the small output signal from the accelerometer. The voltage range of the respiratory signal must lie between 0V to 4.5V when sent to the microcontroller for analog to digital conversion, and further processing.

Since the accelerometer sensor is very sensitive to vibrations in its surroundings, differential amplifying circuitry may be used to obtain an output signal with external disturbances removed.

Figure 4 shows the general design of the respiratory circuit.

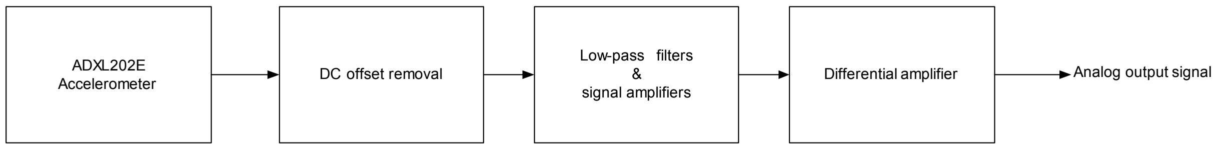


Figure 4: Respiratory circuit block diagram

The accelerometer that we use is Analog Devices ADXL202E. It is a low-cost ±2*g* Dual-Axis accelerometer with duty cycle output. The reason for using it is because ±2*g* accelerometers are close to the most sensitive accelerometers currently available. We originally planned to use the Motorola MMA1260D, an analog output ±1.5*g* accelerometer, but unfortunately Motorola has delayed its issuing date. Figure 5 and Figure 6 show the pinout and functional block diagram of the ADXL202E.

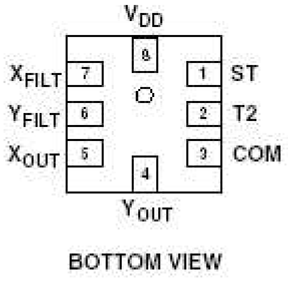


Figure 5: Pinout of the ADXL202E

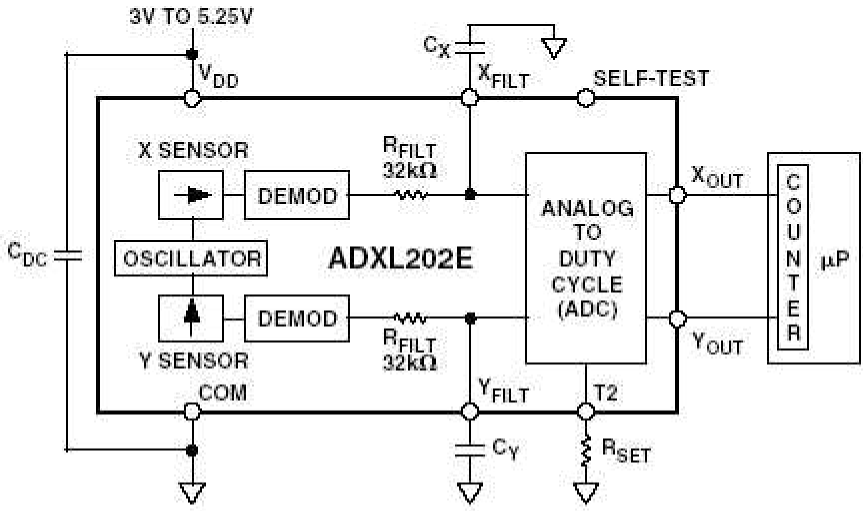


Figure 6: Functional block diagram of the ADXL202E

The Y (vertical) sensor output will be tapped to measure the patient's chest expansion and contraction. Although this accelerometer has a digital output, Y_{OUT}, we decided against using it because it will be easier for us to inspect and amplify an analog signal for analysis. Pin 6 of the ADXL202E will be directly connected to the DC offset removal section of the respiratory circuitry. The output will then be connected to low-pass filters, amplifying circuits and differential amplifying circuit.

5.4. Cost

The cost of the respiratory unit will be kept below $50 by striving to reduce cost of components without sacrificing the performance of the unit.

6. Hardware Interface Design

This section will describe the design of the hardware interface.

6.1. General

The interface will utilize the PIC16F873 microcontroller as the main unit for the hardware interface. The PIC16F873 is a 20MHz microcontroller with 28 pins, five 10-bit A/D converters, 4K FLASH memory, 192 bytes of data memory, three 8-bit I/O ports, three timers, and supports universal synchronous asynchronous receiver transmitter (USART) serial communications. The pinout for the PIC16F873 is shown in Figure 7.

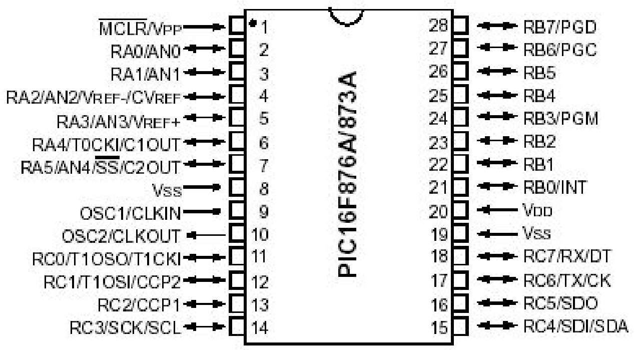


Figure 7: PIC16F873 pinout

The microcontroller will convert analog signals from the EKG unit and respiratory unit to digital using one of the five on-chip A/D converters. These signals must be a minimum of 0V and maximum of 4.5V for the A/D converters to correctly read the signals.

The output of the A/D converters will be buffered. For the respiratory data, the microcontroller will perform a simple digital signal processing algorithm to calculate the respiratory rate. The respiratory rate will be saved as an 8-bit value in an allocated location and will later be sent to the Palm™ PDA for display.

The converted EKG or respiratory data will be sent to the Palm™ PDA using the USART serial port. The Palm™ PDA requires an input signal that follows the RS-232 communications interface standard, so a MAX232A RS-232 Driver/Receiver is used to convert the output signal from the microcontroller to the RS-232 standard. The pinout of the MAX232A RS-232 Driver/Receiver are shown in Figure 8.

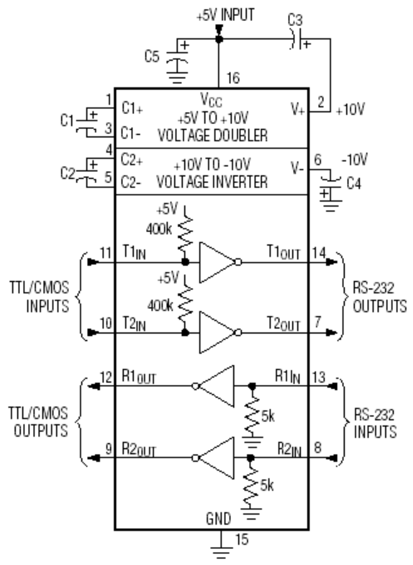


Figure 8: MAX232 driver pinout

6.1.1. Electrical Interface Connection

The Palm™ handheld uses a 16-pin serial/USB connector to communicate with the cradle and the desktop computer. We will be using the same connector to communicate with our hardware attachment. When viewing the handheld front, the pins are defined 1 to 16 from left to right. The Palm™'s 16-pin connector has USB slave, serial (TIA/EIA-232) and some additional signals to support attachment/detachment of a peripheral with Peripheral ID. Table 1 shows the signals associated with each pin of the connector.

Table 1 : Electrical interface signals

Pin #	Signal Name	Function
1	GND	Shield Ground, Charging Ground (Internally connected to Pin-7)
2	USSB_D+	USB D+ signal. Slave for synchronization to PC. Not a master for peripheral attachments.
3	USB_D-	USB D- signal. Slave for synchronization to PC. Not a Master for peripheral attachments.
4	VBUS	USB VBUS. Slave for synchronization to PC. Not a Master for peripheral attachments.
5	HS_IRQ	Wakes handheld. The HotSync® button momentarily connects this pin to Pin-9. Produces a HotSync Interrupt Notification in the Palm OS 4.0 for applications to use to detect the presence of their peripheral. Load resistance range 104,500 – 115,500 ohms
6	unused	Not Connected. Palm reserves for future use
7	SG	Signal Ground (Internally connected to Pin-1)
8	ID	Peripheral ID: a peripheral must tie this to the appropriate 1% value resistor connected to ground. List of Supported Categories of Peripherals: USB Cradle: short

		RS-232 Cradle: 7.5K ohm Mfg. Test Cradle: 20K ohm RS-232 Peripheral: 100K ohm Modem: 220K ohm Undocked: open (>10,000K ohm)
9	VOUT	Regulated Voltage out: 3.3V +- 0.2V at 100mA. From VCC (U12)
10	RXD (in)	Receiving Data: from peripheral to handheld. Connects to U11 Pin-13. Load resistance range 3K – 7K ohms (when not active the transceiver pin is in tristate)
11	TXD (out)	Transmit Data: from handheld to peripheral. Connects to U11 Pin-15. Drives 3K ohms to +-% Volts (when not active the transceiver pin is in tristate)
12	DETECT	Peripheral Attach/Detach detection: a peripheral must tie this pin to Pin-7 (SG)
13	CTS (in)	Clear To Send: hardware flow control handshake signal. Connects to U11 Pin-14. Load resistance range 3K – 7K ohms (when not active the transceiver pin is in tristate)
14	RTS (out)	Request To Send: hardware flow control handshake signal. Connects to U11 Pin-16. Drives 3K ohms to +-5 Volts (when not active the transceiver pin is in tristate)
15	DTR (out)	Transmit Data: from handheld to peripheral. Conects to U11 Pin-17. Drives 3K ohms to +-5 Volts (when not active the transceiver pin is in tristate)
16	VCHRG (in)	Positive terminal for the external DC supply that powers the internal charging circuit that charges the Li-Ion Polymer cell (which comes with a protection circuit). Input: 5.0 VDC +- 5% @ 1.0A. Receives the output of the approved Palm charger (Motorola model R410510 power supply)

Transceiver: U11 Maxim MAX232A

The signals that we will be utilizing are the two RS-232 communication signals (RXD, TXD), the peripheral identification ID, signal ground SG, and the peripheral attach/detach DETECT signal.

The DETECT signal will be connected to SG to indicate a device is connected. The peripheral ID will be connected through a 100K ohm resistor to the signal ground SG to indicate the hardware attachment is a RS-232 peripheral.

6.1.2. System Connection

Figure 9 illustrates conceptually how the hardware attachment will be interfaced to the Palm™ handheld. The RS-232 output signal from the handheld will be connected to the USART input and to the PIC microcontroller through the MAX232A RS-232 Driver/Receiver. The input signal to the PDA will also be connected from the USART output through the RS-232 driver. The RTS/CTS flow control signals will be connected but not implemented unless needed in the future. Components may be added on the signal lines to ensure proper communication between the devices and to meet signal voltage level and current specifications.

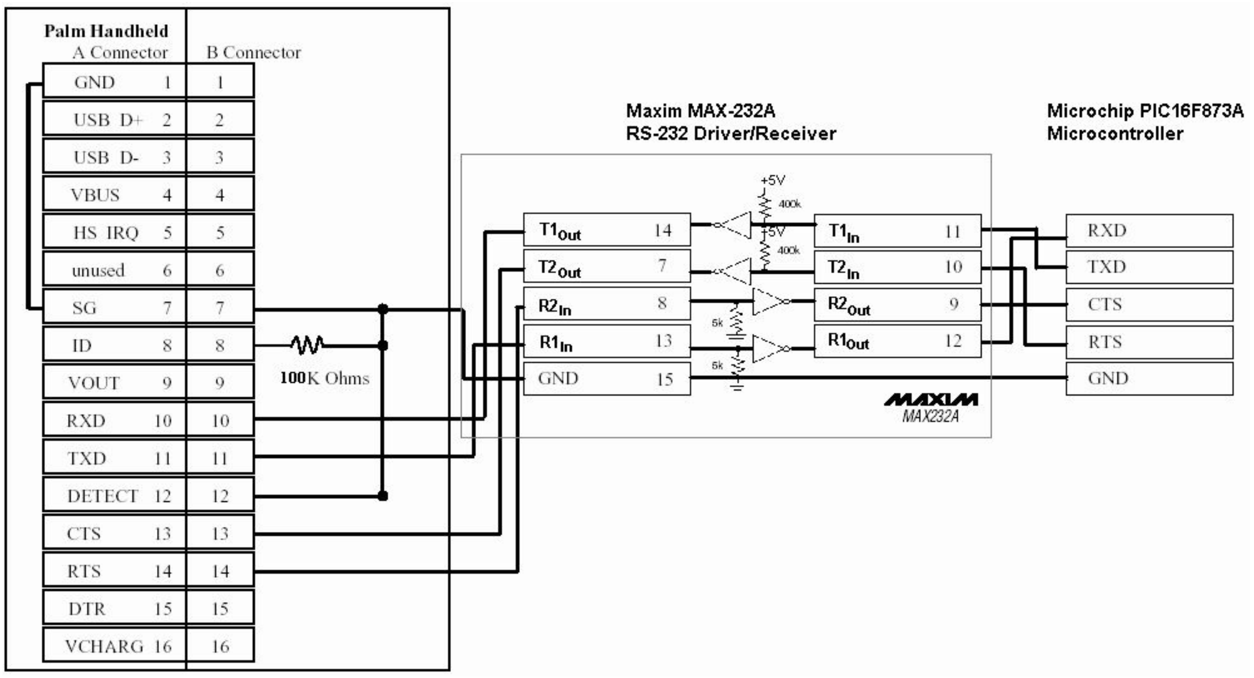


Figure 9: System connection diagram

6.2. Physical Requirements

The microcontroller we will use, the PIC16F873, has more than enough pins for input and output, two of which are A/D converter inputs in Port A which will be used to receive data from the EKG unit and respiratory unit. To send data to the Palm™, the microcontroller will utilize USART serial communications pins in Port C as an output. The interface will be a maximum of 10cm in width, 10cm in length, and 4cm in height. To minimize the size and weight of the interface we will implement it on a PCB.

6.3. Performance

This section will outline the design considerations for the performance of the interface hardware.

6.3.1. Microcontroller Firmware

The microcontroller is the most important component of the hardware interface. The microcontroller firmware will follow the flowchart illustrated in Figure 10. Individual parts of the flowchart will be described in more detail in the following sections.

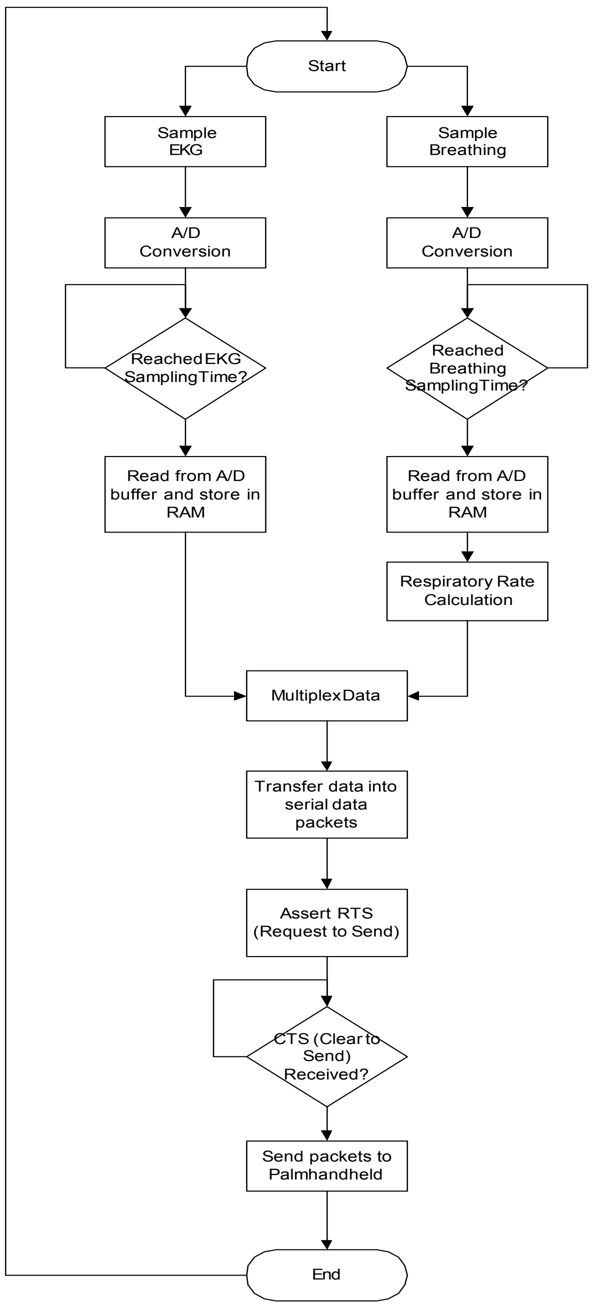
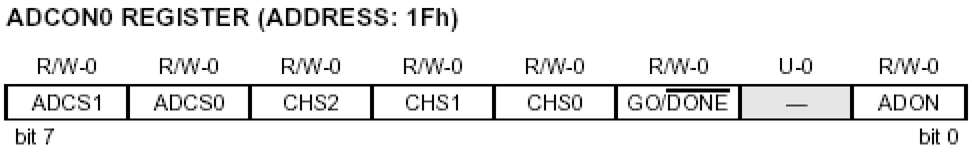

Start
Sample EKG
Sample Breathing
A/D Conversion
A/D Conversion
Reached EKG SamplingTime?
Reached Breathing SamplingTime?
Read from A/D buffer and store in RAM
Read from A/D buffer and store in RAM
Respiratory Rate Calculation
Multiplex Data
Transfer data into serial data packets
Assert RTS (Request to Send)
CTS (Clear to Send) Received?
Send packets to Palmhandheld
End

Figure 10: Microcontroller firmware flowchart

6.3.2. A/D Conversion

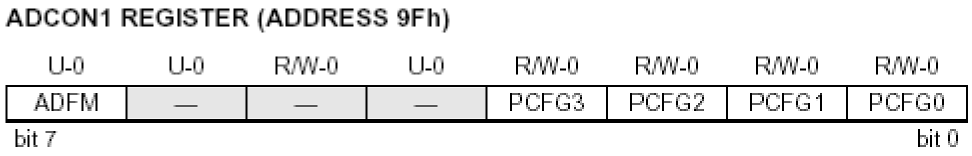
To perform 10-bit A/D conversion the A/D module on the microcontroller must be initialized. The ADCON0 register controls the operation of the A/D converter and the ADCON1 register configures the functions of the port pins. Figure 11 shows the contents of the ADCON0 register and Figure 12 shows the contents of the ADCON1 register.



ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

Figure 11: ADCON0 register



ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Figure 12: ADCON1 register

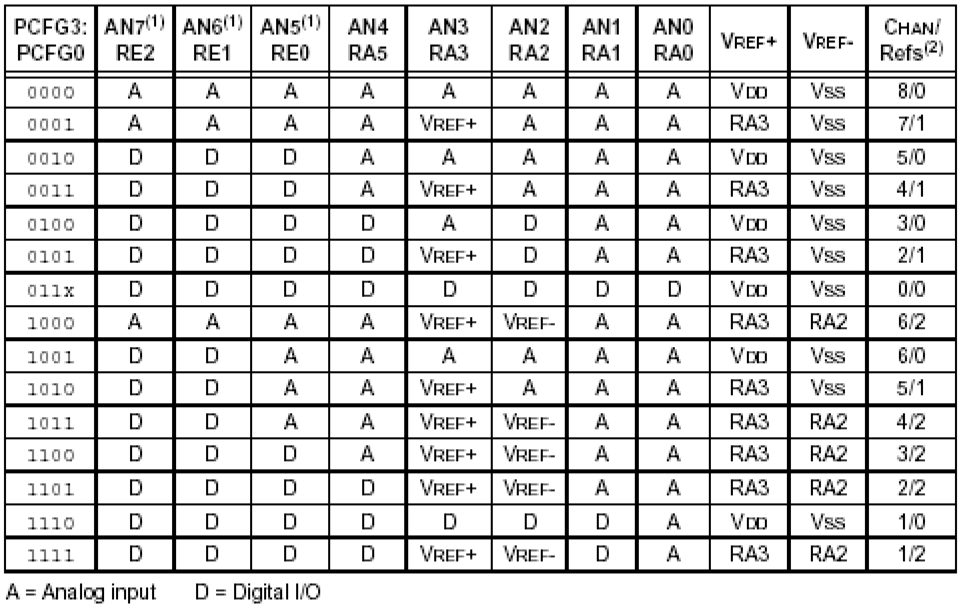
The bits of the ADCON0 register are defined as follows:

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)
bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
bit 2 **GO/DONE:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)
bit 1 **Unimplemented**
bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Since we need to perform A/D conversion only every 0.05s, we can use the slowest A/D conversion clock of FOSC/32. The EKG unit will connected to channel 0 (RA0/AN0) and the respiratory unit will be connected to channel 1 (RA1/AN1). Therefore, the ADCON0 register will be initialized to 10000XX1 when reading from the EKG and 10001XX1 when reading from the respiratory unit.

The bits of the ADCON1 register are defined as follows:

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'
bit 6-4 **Unimplemented**
bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:



PCFG3: PCFG0	AN7[1] RE2	AN6[1] RE1	AN5[1] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs[2]
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss	2/1
011x	D	D	D	D	D	D	D	D	VDD	Vss	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

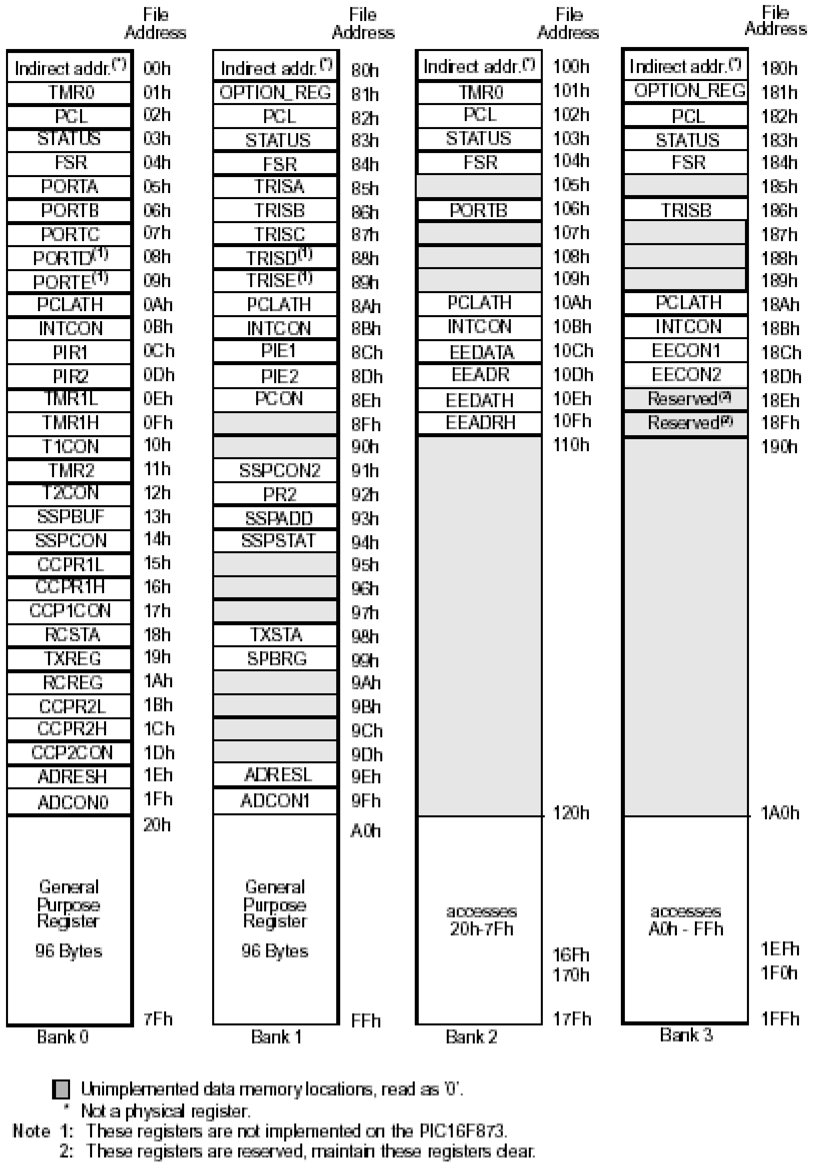
A = Analog input D = Digital I/O

Since we do not have any constraints on the A/D result format, we will arbitrarily choose the left justified format. For the A/D port configuration control bits, we need channels AN1/RA1 and AN0/RA0 for analog inputs. V_{ref+} and V_{ref-} must be tied to V_{DD} and V_{SS} respectively. This corresponds to a PCFG3:PCFG0 of 1001. Therefore, ADCON1 will be initialized with 0XXX1001.

The following steps will be followed to perform an A/D conversion:

1. Initialize the ADCON1 register to 0XXX1001
2. Initialize the ADCON0 register to 100000X1 for reading from the EKG or 100010X1 for reading from the respiratory unit
3. Configure the A/D interrupt
4. Start conversion by setting the GO/DONE bit in ADCON0
5. Read converted data from the A/D result register

The data from the A/D result register will be saved in the microcontroller's RAM for further processing. The register file map for the PIC16F873 is shown in Figure 13.



File Address	Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3
00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h	Indirect addr.(*)
01h	TMR0	81h	OPTION_REG	101h	TMR0	181h	OPTION_REG
02h	PCL	82h	PCL	102h	PCL	182h	PCL
03h	STATUS	83h	STATUS	103h	STATUS	183h	STATUS
04h	FSR	84h	FSR	104h	FSR	184h	FSR
05h	PORTA	85h	TRISA	105h	▨	185h	▨
06h	PORTB	86h	TRISB	106h	PORTB	186h	TRISB
07h	PORTC	87h	TRISC	107h	▨	187h	▨
08h	PORTD(1)	88h	TRISD(1)	108h	▨	188h	▨
09h	PORTE(1)	89h	TRISE(1)	109h	▨	189h	▨
0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	PCLATH
0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	INTCON
0Ch	PIR1	8Ch	PIE1	10Ch	EEDATA	18Ch	EECON1
0Dh	PIR2	8Dh	PIE2	10Dh	EEADR	18Dh	EECON2
0Eh	TMR1L	8Eh	PCON	10Eh	EEDATH	18Eh	Reserved(2)
0Fh	TMR1H	8Fh	▨	10Fh	EEADRH	18Fh	Reserved(2)
10h	T1CON	90h	▨	110h	▨	190h	▨
11h	TMR2	91h	SSPCON2		▨		▨
12h	T2CON	92h	PR2		▨		▨
13h	SSPBUF	93h	SSPADD		▨		▨
14h	SSPCON	94h	SSPSTAT		▨		▨
15h	CCPR1L	95h	▨		▨		▨
16h	CCPR1H	96h	▨		▨		▨
17h	CCP1CON	97h	▨		▨		▨
18h	RCSTA	98h	TXSTA		▨		▨
19h	TXREG	99h	SPBRG		▨		▨
1Ah	RCREG	9Ah	▨		▨		▨
1Bh	CCPR2L	9Bh	▨		▨		▨
1Ch	CCPR2H	9Ch	▨		▨		▨
1Dh	CCP2CON	9Dh	▨		▨		▨
1Eh	ADRESH	9Eh	ADRESL		▨		▨
1Fh	ADCON0	9Fh	ADCON1		▨		▨
20h	General Purpose Register 96 Bytes	A0h	General Purpose Register 96 Bytes	120h	accesses 20h-7Fh	1A0h	accesses A0h - FFh
				16Fh		1EFh	
				170h		1F0h	
7Fh		FFh		17Fh		1FFh	
	Bank 0		Bank 1		Bank 2		Bank 3

▨ Unimplemented data memory locations, read as '0'.
* Not a physical register.
Note 1: These registers are not implemented on the PIC16F873.
2: These registers are reserved, maintain these registers clear.

Figure 13: PIC16F873 register file map

We will reserve bytes 20h to 40h (32 bytes = 16 samples) in RAM for this purpose.

6.3.3. Respiratory Rate Digital Signal Processing

We will utilize DSP to calculate the respiratory rate to be sent to the Palm™ PDA. Once the microcontroller has saved five breathing samples in RAM, we will perform a simple DSP algorithm and update the respiratory rate. As a result, the interface will perform the algorithm every 0.25 seconds. The resulting value will be saved at address DDh where it will eventually be sent to the Palm™.

For the simple DSP algorithm, we will allocate memory locations A0h to DCh (60 samples) for storage. Data in these memory locations will contain the time average of the respiratory data and represent 15 seconds of respiratory data. From the time average of the signal, we will then perform comparisons on the time-average data and to calculate the respiratory rate. The following steps outline the DSP algorithm that will be used.

1. Perform a time average of five respiratory samples
2. Store the result in one of the 60 memory address allocated for storage of time average data
3. Update the designated location for the next iteration of the DSP algorithm
4. Count number of maximums after a minimum is found (number of counts = respiratory rate)
5. Store the respiratory rate into address DDh for transfer to the Palm™ PDA

The stored respiratory rate should be 8-bits, resulting in a maximum breathing rate of 256 breaths per minute, which is more than we need.

6.3.4. RS-232 Communication

The RS-232 standard is an asynchronous serial communication method. The transmitter and the receiver are not synchronized, and data transactions can begin at any time. It is up to the receiver to detect and receive transmissions. The standard defines a format for the data transmitted between devices. The data format is illustrated in Figure 14.

Start Bit (1-bit)	Raw Data (8-bit)	Parity Check Bit (1-bit)	Stop Bit (1-bit)

Figure 14: RS-232 serial data structure

An on state is termed marking, while an off state is termed spacing. The line between the devices is held in the marking state when it is idle. At the beginning of each transfer, a spacing start bit is sent. The start bit causes a mark to space transition on the line, which can be easily detected by the receiver. Eight or nine bits of data are then transmitted across the connection. Depending on the number of data bits used, the ninth bit can be an optional parity bit. Lastly one or more marking stop bits are sent to complete the transfer.

The configuration we will be using for serial communication is as follows. We will use eight data bits to transmit information. A detailed description of how information will be contained in the data structure can be found in the Data Transmission Format section. The parity bit will be enabled, and odd parity is arbitrarily chosen for simple error checking. The data packet will be discarded if an error is detected on inspection of the parity bit. One stop bit will be used for back-to-back data transfers. Neither hardware nor software flow control will be implemented as the PDA is fast enough that it can process all the data sent from the microcontroller as it arrives.

6.3.5. *Data Transmission Format*

Serial data bytes are ordered using Motorola-based packaging scheme where the most significant byte of the data is located at the lowest memory address. The data structure format we will use for transmission to the handheld is illustrated in Figure 15.

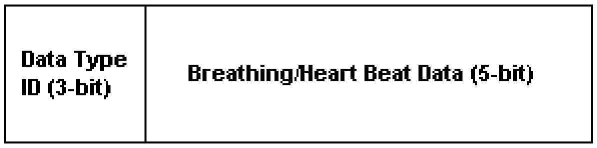


Figure 15: Data byte structure

The serial data byte structure uses the most significant three bits of the eight bit serial packet for a "Data Type ID", and the last five bits for respiratory rate/EKG raw data. Since the A/D conversion has a 10-bit output, a single piece of EKG data will be split into two serial data packets and sent one after the other. To distinguish between different bytes of the same piece of data, the "Data Type ID" indicates which device (breathing/EKG) the data is coming from, as well as whether the data bits represent the most significant or the least significant 5-bits.

Currently, only four data type IDs are used. They identify the lower and upper 5-bits of EKG data, and the lower and upper 5-bits of respiratory rate data. The remaining reserved values can be used for hardware expansion in the future. For example, if we need to send commands, a data type ID can be assigned to system commands, and the 5-bits of the data can contain the actual command.

Definition of the data structure and the data type IDs are show in Table 2.

Table 2: Serial byte data structure

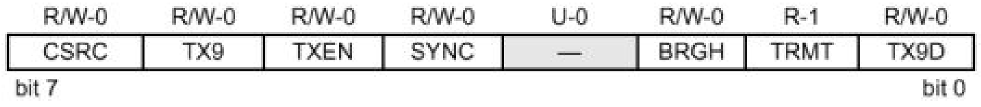
Byte Ordering	Description	Value
S_DATA [7:5]	Data Type ID	000 – LSB of Heart Beat Data 001 – MSB of Heart Beat Data 010 – LSB of Breathing Data 011 – MSB of Breathing Data 100 to 111 – Reserved
S_DATA [4:0]	Breathing/Heart Beat Data	5-bit Unsigned Integer

6.3.6. *Serial Port Configuration*

The USART serial I/O module on the PIC16F873 microcontroller will be used to implement the serial communication between the Palm™ handheld and the hardware attachment. The USART is capable of transmitting and receiving synchronous and asynchronous serial data. For our project, we will use its asynchronous communication functionality.

The USART can be easily configured to work with RS-232 serial data. On the transmission side, it can package the data into an RS-232 compatible format, and send the data out through a port on the microprocessor. The receiver on the USART will not be enabled because we do not foresee communication from the Palm™ handheld to the hardware attachment. However it will be connected in hardware should we need to implement receiving capability in the future.

The USART transmit and receive functions are configured through the transmit status and control register (TXSTA) and receive status and control register (RCSTA) registers. Figure 16 shows the contents of the TXSTA register.



R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Figure 16: TXSTA: Transmit status and control register (98h)

The bits of TXSTA register are as follows:

bit 7 **CSRC**: Clock Source Select bit
<u>Asynchronous mode:</u>
Don't care
bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission
bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0= Transmit disabled
bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0= Asynchronous mode
bit 3 **Unimplemented**: Read as '0'
bit 2 **BRGH**: High Baud Rate Select bit
<u>Asynchronous mode:</u>
1 = High speed
0 = Low speed
bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full
bit 0 **TX9D**: 9th bit of Transmit Data, can be Parity bit

To enable the transmission and set the correct configuration, we should write the value 0110 X0XX to the TXSTA register. The ninth bit of transmit data will be the parity bit for odd parity checking, and will be computed for each piece of data transferred. The high baud rate select bit is used to set the baud rate divider, and will be set to '0' for our use.

Data to be transmitted is written to the TXREG transmit register for temporary storage. When the microcontroller detects the data, it will transfer it to the USART transmit shift register, which then packages and outputs the data to the line serially. When the transfer is complete, the microcontroller clears the TXREG register and raises the TXIF flag to indicate the transfer is complete. The next piece of data can then be written to the transmit register.

The baud rate of the communication is set through the Baud Rate Generator Register (SPBRG). The baud rates possible from the USART are listed in Table 3.

Table 3 : Baud rates for asynchronous mode (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

Our system clock will operate between 8MHz and 20MHz. The Palm™ handheld and the hardware attachment will communicate at baud rate of between 2400 to 19200. Depending on the baud rate chosen, the appropriate value will be written to the SPBRG register.

The serial data reception functionality is controlled through the Receive status and control register (RCSTA). Figure 17 shows the contents of the RCSTA register.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Figure 17: RCSTA: Receive status and control register (18h)

The bits of RCSTA register are as follows:

bit 7	**SPEN**: Serial Port Enable bit
	1 = Serial port enabled
	0 = Serial port disabled
bit 6	**RX9**: 9-bit Receive Enable bit
	1= Selects 9-bit reception
	0 = Selects 8-bit reception
bit 5	**SREN**: Single Receive Enable bit
	Asynchronous mode:
	Don't care
bit 4	**CREN**: Continuous Receive Enable bit
	Asynchronous mode:

1 = Enables continuous receive
0 = Disables continuous receive

bit 3 **ADDEN**: Address Detect Enable bit
<u>Asynchronous mode 9-bit (RX9 = 1)</u>
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte
0 = No framing error

bit 1 **OERR**: overrun Error bit
1 = overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D**: 9th bit of Received Data (can be parity bit, but must be calculated by user firmware)

Since the receive side of the USART is not used, the value of '0' is written to bit 7 of the register to disable the receiver.

The steps to perform asynchronous transmission is as follows:

1. Initialize SPBRG register for appropriate baud rate
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN
3. Set transmit bit TX9 to enable 9-bit transmission
4. Enable transmission by setting bit TXEN
5. Load the 9th bit into TX9D
6. Load data to the transfer register

6.3.7. Serial Signal Driver

The RS-232 standard defines voltage levels and drive strengths of the signals used. The Maxim MAX232A RS-232 Driver/Receiver is used to convert signals between TTL which is the level used by the microprocessor I/O, and the signal levels used for RS-232 communication.

The four signals on the Palm™ handheld used for serial communication are TX, RX, RTS, and CTS. RTS and CTS are not used as they are signals for hardware flow control. The input signals going into the handheld (RX) is connected to the R1in pin of the MAX232 driver. The output signal coming from the handheld (TX) is connected to the T1out pin of the driver. These two signals are connected to the appropriate pins on the microcontroller, as illustrated earlier in Figure 9.

6.4. Cost Requirements

The cost of the hardware interface will be kept below $225 by striving to reduce the cost of components.

7. Software Design Specification

This section describes the high-level design of the Palm™ handheld software for the VitalChart software.

In order to complete the requirements as described in the functional specification, the following section describes the general design requirements for the operation of the Palm™ OS software. All software designed for the Palm™ OS platform is based on event-driven architecture. When the software starts, it constantly checks the event queue for a new event. When a new event has entered the queue, the Palm™ OS will schedule and allocate resources for the new event. The event handler of the software will then identify the event object type, retrieve the event pointer, and execute it.

7.1. Software System Overview

In our VitalChart software, we have the following event handlers: System, Application, Hardware Communication, Forms, and Menu. Figure 18 is a high-level block diagram for typical program control flow.

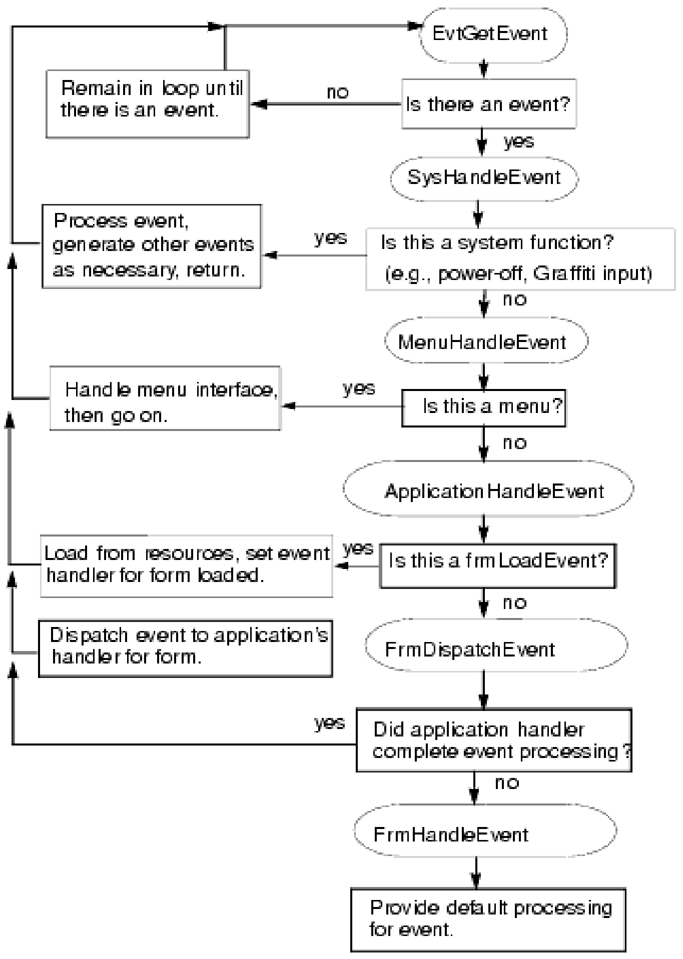

EvtGetEvent
Is there an event?
no
Remain in loop until there is an event.
yes
SysHandleEvent
Is this a system function? (e.g., power-off, Graffiti input)
yes
Process event, generate other events as necessary, return.
no
MenuHandleEvent
Is this a menu?
yes
Handle menu interface, then go on.
no
ApplicationHandleEvent
Is this a frmLoadEvent?
yes
Load from resources, set event handler for form loaded.
no
FrmDispatchEvent
Dispatch event to application's handler for form.
Did application handler complete event processing?
yes
no
FrmHandleEvent
Provide default processing for event.

Figure 18: Program control flow

7.1.1. Event Loop

After the application performs the start-up routine, the application will automatically enter an event loop. When a new event is received, the event loop will process the event object and pass it to the appropriate event handler. The event loop will run indefinitely unless either the application has terminated or a new event is received.

7.1.2. Application Event Handler

The application event handler is mainly used as an intermediate event processor which directs the received event to the appropriate form handler or user-defined handler.

7.1.3. System Event Handler

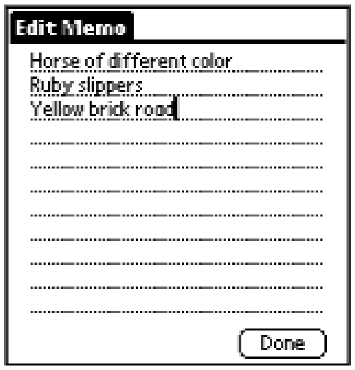
The system event handler is responsible for processing application launch, termination, and system functions. When our application starts, the Palm™ OS sends a launch flag which tells the application what mode it should be running in. VitalChart software has a normal launch and termination mode. When the application terminates, a termination event is generated and all resources allocated to the application will be released back to the OS. System events such as power-off, button clicks, text-input using Graffiti, and alarm generation are also managed by the system event handler.

7.1.4. Menu Event Handler

A menu event handler is responsible for processing menu interaction. Whenever the user activates the menu, a menu event is generated. A menu event object contains the application identification flag, originator code, and the name of the menu function which the user is accessing. Whenever a menu event is received, a corresponding form event is also generated.

7.1.5. Form Event Handler

A form is the graphical user interface area of the application such as the buttons, plots, table, and menu bar. For example, when user clicks on the "About" menu, a form is generated to show the expanded menu bar and the "About" screen. An example form is shown in Figure 19.



Horse of different color
Ruby slippers
Yellow brick road

Figure 19: A Palm™ form

The form event handler is responsible for managing and processing form displays. Every element in the GUI area of the Palm™ is a form object. Whenever the user opens a menu or click on a button, the form handler will update the new form to the screen.

7.2. Graphical Display Design

The graphical display facilitates the display of the EKG and respiratory data on the GUI. The data samples displayed are sent to this block through the Software to Hardware Interface as described in Software to Hardware Interface Design. This section outlines the functions and high-level blocks for the graphical display.

7.2.1. General

The graphical display engine handles the mathematical manipulation of each data point received from the Software to Hardware Interface. This block calculates the heart rate from the EKG data as well as controls the functional aspects of the display interface. In addition, the respiratory rate is displayed as transmitted by the Software to Hardware Interface. Figure 20 shows a mock-up of the graphical display.

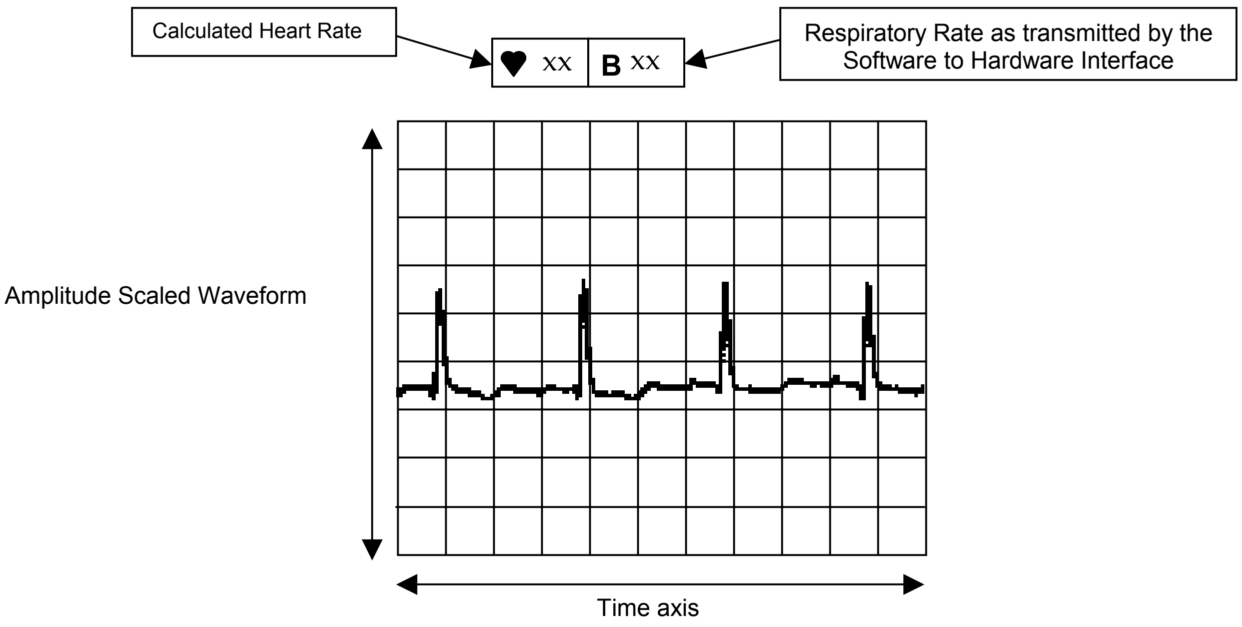

Calculated Heart Rate
♥ xx
B xx
Respiratory Rate as transmitted by the Software to Hardware Interface
Amplitude Scaled Waveform
Time axis

Figure 20: Graphical display mock-up

The background grid can be removed via the Forms Event Handler at the control of the user as described in Form Event Handler. The user input directly controls the existence of this grid. The amplitude of the waveform is normalized such that the display will show the maximum swing. The time axis will be calculated by the sampling rate of the signal and will be injected into the system. In addition, this block will have a graphics buffer that can be used to enter a pause state to retain the EKG image on the display as described in the performance section below. From this buffer, the display can also be restarted after it exits the pause mode. This graphics buffer is also utilized to provide a sweeping effect for the incoming data. Figure 21 shows the flow of the incoming data.

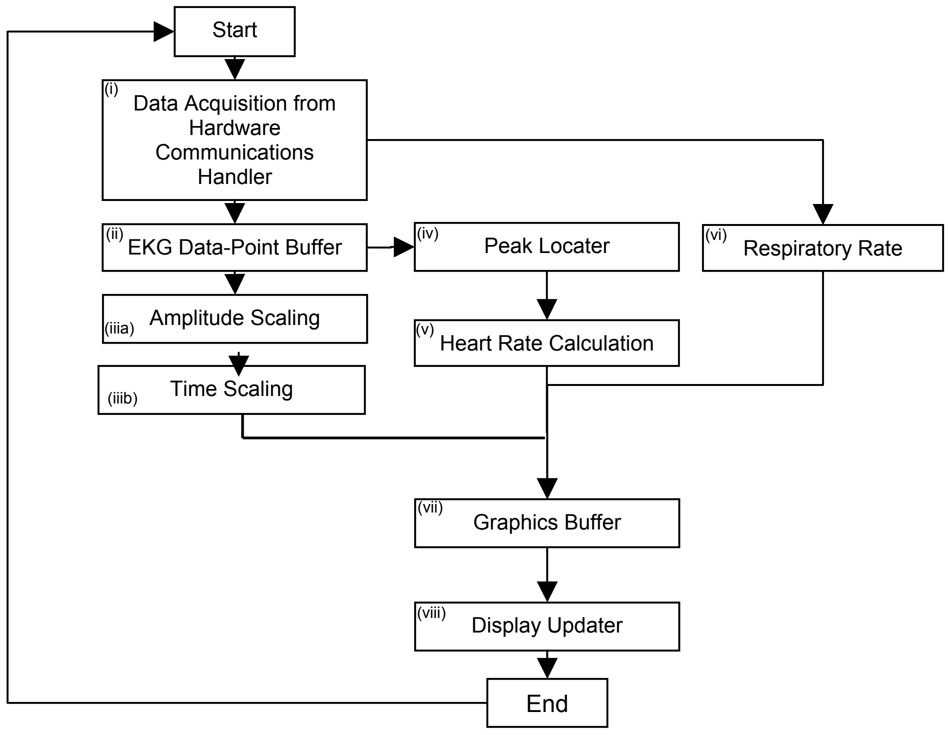


Figure 21: Graphics engine overview

7.2.2. *Performance*

This section describes the performance of the blocks as illustrated in the previous section. The Roman numeral corresponds to the equivalent block.

7.2.2.1. *Data Acquisition Block*

i. The data acquisition block is described in the Software to Hardware Interface Design section and provides the Graphics Engine with 16-bit unsigned integer representation for the EKG as well as the integer value of the respiratory rate.

7.2.2.2. *EKG Data-Buffer*

ii. Each EKG data point is assigned with the following structure as it is passed from the Data Acquisition Block:

```
Sample.value = data value (unsigned 16-bit)
Sample.time = un-initialized (float)
```

These data points will then be stored in a 2-second sample size buffer for further processing. When the application is invoked, this buffer will fill with EKG data to begin a pipeline for the later processing blocks.

7.2.2.3. *Amplitude Scaling Block*

iiia. The EKG data points are amplitude scaled to display in the Main Window Plot Area. The amplitude normalizing factor is calculated from the data values stored in the EKG Data Point Buffer. Normalizing is required due to the limited display space of the Main Window Plot Area. If the input signal is weak, then the data must be re-scaled to allow for higher resolution display.

7.2.2.4. *Timing Scaling Block*

iiib. The timing scaling block uses a time constant generated by the Palm™ OS by converting the sampling frequency of the hardware into a time spacing. Each time space is separated by the inversion of the sampling frequency times 2π.

7.2.2.5. *Peak Locator*

iv. The EKG data buffer is analyzed to calculate a threshold value. This threshold value will be used to locate the first peak of the QPRS waveform.

7.2.2.6. *Heart Rate Calculator*

v. The timing between each peak is averaged using a moving average filter. The time between peaks is then used to calculate the relative beats per minute.

7.2.2.7. *Respiratory Rate Buffer*

vi. The respiratory rate is read directly from the Hardware Communications Handler and is stored into a variable for display. This buffer is only updated once every 15 seconds and will continue to hold the current number until updated.

7.2.2.8. *Graphics Buffer*

vii. The EKG data is converted into a corresponding x-axis and y-axis location from the sample.value and sample.time values. This point is plotted into a graphics buffer that holds the updated image. This allows the plotting of new data to occur while the Main Window is displaying the previous values. In addition, the heart rate and respiratory rate numbers are also plotted into the graphics buffer. Furthermore, the buffer can be paused by entering a suspend mode. This mode allows for the retention of the current screen for the user and this function is controlled directly by the Play/Pause – Button described in Play/Pause – Button.

7.2.2.9. *Display Updater*

viii. The graphics buffer data overwrites the current Main Window. The data stored in the buffer becomes visible to the user. The waveform displayed at the time of the previous sample is kept until the new plot-point is computed. Each pixel of the waveform is plotted in a 160x160 resolution screen. The display updater will update the screen for the respiratory rate every 15 seconds using the Palm™ OS timing functions.

7.3. Graphical User Interface Design

The GUI for the Palm OS Platform is designed using the Codewarrior Constructor for Palm™ OS. The GUI handles the event triggering and data display for the VitalChart application. The following sections describe each component separately. Using the Palm™ m500 series, the color spectrum will automatically be configured to 256-bit grayscale.

7.3.1. General

The built in taps and presses of the Palm™ OS will be utilized to create events for the GUI. The software links all the process functions to the user such that each active area corresponds to an event and will be handled by the Application Event Handler.

7.3.1.1. Main Window

After invoking the VitalChart application, the Main Window Form is displayed. This form allows the interaction between events and the system. While in operation, the tool bar will be hidden from view by the Forms Event Handler. Figure 22 shows the mock-up of the Main Window.

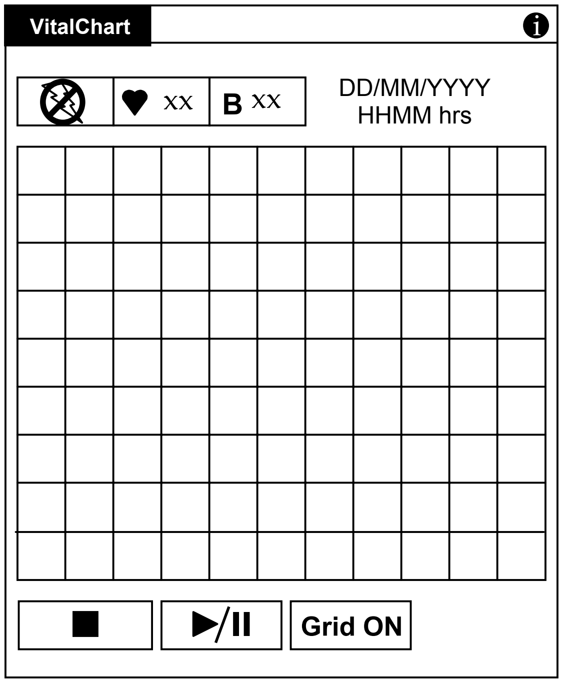


Figure 22: Main window mock-up

7.3.2. Performance

The buttons and menus are discussed in the following sections.

7.3.2.1. About (i) – Button

The about button brings forth the about form as described in Form Event Handler. This button is located at the top right corner of the main window and is visible at all times. When tapped, this object will invoke the display of the About Form. The About menu item will be located in the top left hand corner of the Main Window at all times. When tapped, this object will invoke the display of the About Form via the Form Handler.

7.3.2.2. Stop – Button

The stop button halts the data acquisition. This button is located at the bottom of the Main Window, to the left of the play/pause button. This button will be mapped to a stop event within the Event Handler in the Codewarrior Constructor. When this button is tapped, the application will cease to display data until the play/pause button is pressed.

7.3.2.3. Play/Pause – Button

The play/pause button allows the current screen to be held constant for closer examination. This button is located at the bottom of the Main Window, to the right of the stop button. The button will be mapped to the play/pause event within the Codewarrior Constructor. When the application is in stop mode, tapping this button will cause the application to begin data acquisition. Tapping this button when the application is running will cause the application to hold its current screen. Tapping this button during pause mode will resume data acquisition.

7.3.2.4. Grid On/Off Button

The grid on/off button allows the grid of the plot area to be turned on or off. This button is mapped to the grid-on/off event within the Codewarrior Constructor. When the grid is on, the button will display “Grid ON”. When the grid is off, the button will display “Grid OFF”. These functions will be performed using the Event Handler. This function directly influences the grid functionality of the Graphics Display as described in Graphical Display Design.

7.3.2.5. Date/Time Display

The date and time for the start time of the current data acquisition is displayed on the top left hand corner of the Main Window. This time stamp will be taken from the Palm™ OS internal clock.

7.3.2.6. Plot Area

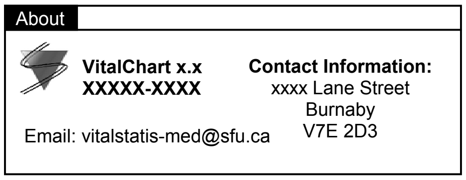
The plot area contains the region for the display of the EKG waveform this data is scaled and normalized for display by the Graphics Display as described in Graphical Display Design.

7.4. Application Menu Design

The application menus are forms that display extra information to the user. The form handling is accomplished by the Forms Event Handler.

7.4.1. About Page

The about page is an information form created using Codewarrior Constructor for Palm™ OS. This form is opened using the Forms Event Handler when the About(i) button is pressed on the GUI as described in About (i) – Button. Figure 23 depicts a mock-up of the About form.



About

VitalChart x.x
XXXXX-XXXX

Contact Information:
xxxx Lane Street
Burnaby
V7E 2D3

Email: vitalstatis-med@sfu.ca

Figure 23: About form

7.5. Software to Hardware Interface Design

Our VitalChart software uses the built-in serial communication port on the Palm™ handheld for communication with the external respiratory circuit and EKG circuit. The communication between the external hardware circuit and the Palm™ handheld is based on the RS-232 serial communication protocol.

7.5.1. General

The serial communication on Palm™ PDA is completely interrupt-driven for receiving data. The Palm™ PDA utilizes a 5-pin (SG, TxD, RxD, CTS, RTS) serial port for communication. All data going into the PDA uses the Motorola-based byte ordering scheme by which the data is packaged with the most significant byte at the lowest address. The data packet can be either 2-byte (16-bit unsigned integer) or 4-byte (32-bit unsigned integer) long.

The Palm™ OS provides an API called Serial Manager which performs high-level serial I/O and flow control for RS-232 communication. A user-defined interrupt handler will regularly receive interrupt signals from the VitalChart software to acquire new data from the serial I/O port. Figure 24 is a high-level block diagram illustrating the serial communication interface.

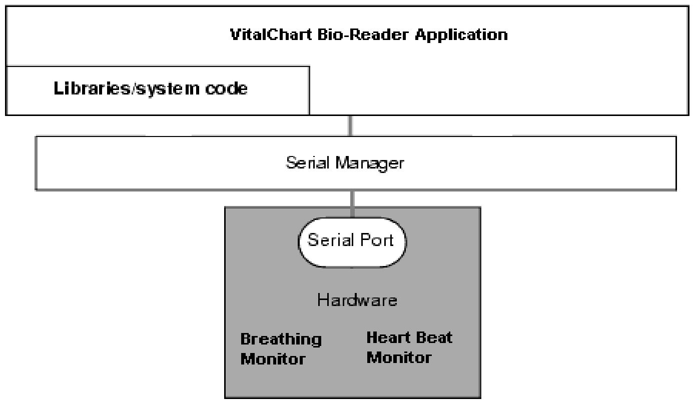


Figure 24: Palm™ OS serial communication architecture using SerialLink Manager

7.5.2. Performance

The following sections describe the performance requirements for the hardware communication interface for the Palm™ OS handheld.

7.5.2.1. Interrupt Event Handler

The interrupt event handler processes interrupt-driven event such as data acquisition and serial communication. The VitalChart software regularly generates an interrupt event for respiratory rate and EKG data acquisition. When an interrupt is generated, all other system functions are disabled until the interrupt has been serviced.

7.5.2.2. Serial Link Manager

The Palm™ OS built-in Serial Link Manager provides a high-level programming interface for communication between the VitalChart software and external hardware circuitry for respiratory rate and EKG monitoring. To start receiving data, the Serial Link Manager first configures the serial port with the correct baud rate and flow control flag. Once configuration is completed, the serial port will listen to any incoming data and receive it asynchronously. The received data will be stored in the memory buffer and can be retrieved anytime it is needed.

7.5.2.3. File Format Design

Our application uses the Palm™ Resource (PRC) file format for storing GUI and code resources. A PRC (or an application) file can be installed onto the Palm™ PDA from a desktop computer via a HotSync operation.

8. Conclusion

This document outlines the detailed design of both hardware and software components of the VitalChart. In the hardware section of the design specification, we identified the components we will use, and attached the schematics and connection diagrams of how the hardware components will interact to accomplish the functions listed in the functional specification. Communication between the hardware attachment and the PDA is also described in detail. The software section of the document includes flowcharts of the microcontroller firmware and the PDA software application, and describes how the various sections of the user application will be implemented on the Palm™ m500 PDA.

With the completion of the design specification, the VitalStatis development team is ready to proceed to prototyping the product and implementing the features listed in the functional specifications. We are confident that we will be able to produce a fully functional prototype that meets the system requirements by the projected completion date of December 13, 2002.

9. References

Analog Devices Inc, (2002) “ADXL202E - Low-Cost ± 2*g* Dual-Axis Accelerometer With Duty Cycle Output Data Sheet (Rev. A, 10/00).” http://www.analog.com/productSelection/pdf/ADXL202E_a.pdf (Accessed October 29, 2002).

Lammert Bies, (2002) “Lammert Bies: RS232 Specifications.” http://www.lammertbies.nl/comm/info/RS-232_specs.html (Accessed October 26, 2002).

Maxim Integrated Products, (2002) “MAXIM +5V-powered, Multichannel RS-232 Drivers/Receivers.” http://pdfserv.maxim-ic.com/arpdf/MAX220-MAX249.pdf (Accessed October 24, 2002).

Maxim Integrated Products, (2002) “Selecting and Using RS-232, RS-422, and RS-485 Serial Data Standards.” http://www.maxim-ic.com/appnotes.cfm/appnote_number/723 (Accessed October 26, 2002).

Microchip, (2002) “PIC16F87X Data Sheet.” http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf (Accessed October 24, 2002).

Palm, Inc., (2002) “Event Loop.” http://www.palmos.com/dev/support/docs/palmos/EventLoop.html#977418 (Accessed October 26, 2002).

Palm, Inc., (2002) “Serial Communication.” http://www.palmos.com/dev/support/docs/palmos/SerialCommunication.html#989132 (Accessed October 26, 2002).