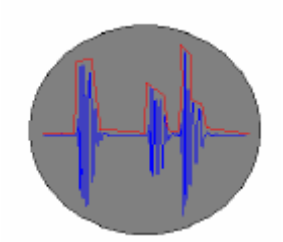
ECE 396

Senior Design II

Academic Advisor: Vladimir Goncharoff

Heart Murmur Detector



Group Members:

Arnold Lee
Jovani Ibarra
Tenille Medley

May 5, 2007

Table of Contents

Abstract:

Heart Failure is the number one cause of death. The primary goal of this project is to develop a product that will make it easier for physicians and medical technicians detect heart murmurs inpatients. Our device will hopefully aid in future diagnosis, and in the long run create a means of early detection of heart murmurs. The main functions of this device include accepting an analog input from a microphone or stethoscope. These audio data are collected, stored and recorded for at least four cycles of real-time heartbeats. We are able to filter out any noise caused by the sensors. Finally, the product will present results in a format that can make it easy for a user to visualize and analyze them. A GUI will also be developed to download signals into a PC and perform further analysis. The product will be tested to detect heart murmurs from a sample database of normal and abnormal heart sounds.

This report describes the steps that we took in developing out senior design project

Step 1. Identification of the Problem.

Every year, death due to heart failure claims the lives of approximately 700 thousands Americans. This research proposes methods that would be useful for an electronic stethoscope that will detect murmurs based on characteristics such as murmur duration, dominant frequency, period (systolic, diastolic), and magnitude. This topic was chosen because there was an aspiration to understand how electrical engineering could tie into other fields, and a desire to investigate ways to improve detection of heart abnormalities while applying theoretical concepts. This investigation will hopefully propose a method that could potentially prolong the life of patients with minor, mild, and major heart murmurs.

The main problem that exists with traditional analog stethoscopes is that it is really hard to hear murmurs, and there is human error involved in the detection of heart murmurs. Many patients have complained time and time again that it has taken two, three, sometimes even four doctors before a murmur is even detected. Another existing problem is that for older doctors hearing ability decreases with age, and younger doctors don't possess the years of experience that older doctors have with detection of heart murmurs.

The expectation of this research is that by extracting appropriate features from the PCG classification of different murmurs will be possible. However, in situations where the timing of the murmur is not distinct from the heart sound, the middle 70% of the murmur will be extracted for further analysis. Ultimately, this research will be used to develop hardware that will identify heart murmurs based on characteristics of the PCG.

Step 2. Research

Mechanics of the Heart

The human heart is the hardest working organ in the body. It is about the size of a fist and can weigh between seven and fifteen ounces. The heart beats approximately 100,000 times and

pumps 2,000 gallons of blood a day. The period of a normal heart beat (lub and dub, or S1 and S2 sounds), takes about a second.

The heart is composed of vena cava, valves, atriums, veins and arteries [figure 3]. The right and left atria and the right and left ventricles are known as the four chambers of the heart. The ventricular septum is a wall that separates the left and right atria, as well as the left and right ventricles. The aorta (an artery) carries blood from the heart to the body.

The pulmonary arteries carry blood from the heart to the lungs to restore oxygen. The pulmonary veins carry blood, which is saturated with oxygen, from the lungs to the heart. Blood, from the lower part of the body, travel through the inferior vena cava to get to the heart. This blood is oxygen deficient. Blood, from the upper part of the body, travels through the superior vena cava to return to the heart. The right atrium accumulates blood from the body and pushes blood through the tricuspid valve into the right ventricle. The right ventricle then forces the blood into the pulmonary arteries and then into the lungs, via the pulmonary valve. The left atrium accumulates blood from the lungs via the pulmonary veins. The blood then travels to the left ventricle, via the mitral valve, then through the aortic valve to the aorta, then finally back to the body.

Depending on how active the body is, the heart can beat from 50 to 100 times in a minute. During the first part of the heartbeat, blood is collected in the left and right atria. The sinoatrial node, which is located in the right atrium, sends a signal, which causes the atria to contract [figure 4]. This contraction causes blood to be pushed through the mitral and tricuspid valves. This part of the heartbeat is called the diastole.

The second part of the heartbeat occurs when the left and right ventricles are filled with blood and the sinoatrial node signals the ventricles to contract. This part of the cycle is known as the systole. During this cycle the mitral and tricuspid valves are shut tight to prevent backflow of blood. The blood is therefore transported through the pulmonary and aortic valves.

When there is some abnormality to the way that the heart operates, the heart will not always work this systematically. Certain heart conditions and abnormalities will cause the valves to not close completely during the systole, or not open fully during the diastole, causing some type of regurgitation or stenosis. In fact, many murmurs are caused by valvular diseases, however, most murmurs are innocent, and therefore do not need medical attention.

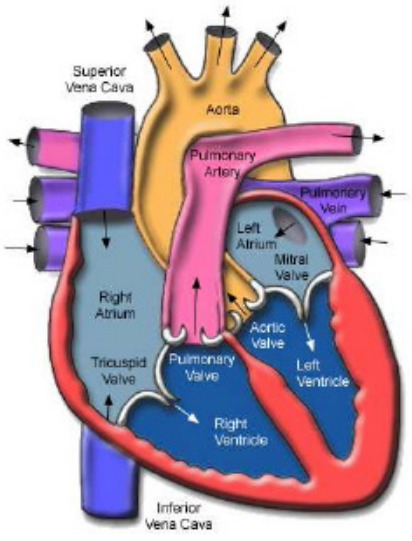
Heart Murmurs

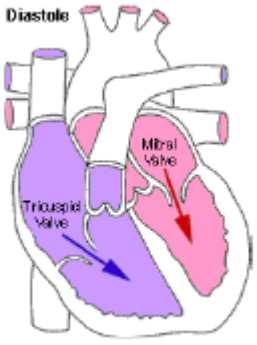
The heart contains four valves: tricuspid, pulmonary, aortic, and mitral. When functioning normally, they all regulate the blood flow, so that it moves in one direction, consistently. The valves have leaflets that resemble a one-way trap door. However, some heart dysfunctions may cause the valves to malfunction. These valvular abnormalities can cause regurgitation (when blood flows the opposite direction of where it is supposed to flow), or it will lead to a type of stenosis (a condition when the valves are so tight that the normal amount of blood cannot flow through).

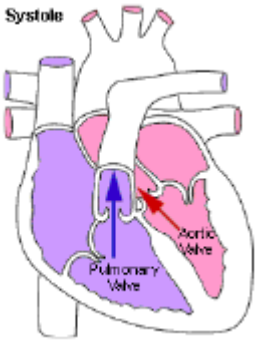
Tricuspid regurgitation is a common disorder that includes the backflow of blood from the right ventricle back into the right atrium. This disorder is caused by damage to the tricuspid heart valve. Complications that would cause tricuspid regurgitation include rheumatic fever and certain diet medications, to name a few. Tricuspid regurgitation is medically treated based on the severity of the symptoms. Mitral valve regurgitation is a disorder in which the mitral valve does not close tightly, therefore allowing blood to flow back into your heart. Similar to tricuspid regurgitation, complications that cause mitral regurgitation include rheumatic fever, diet pills, and deterioration of the valve with age. Pulmonary and aortic regurgitation carry the same characteristics as tricuspid and mitral regurgitation.

A heart murmur is an abnormal sound of the heart that is usually caused by valvular dysfunctions. Systolic murmurs occur between the first and second heart sound (S1 and S2), and diastolic murmurs occur between the second and first heart sounds (S2 and S1). The severity of systolic murmurs is determined by grades, with grade 1 being the lowest in amplitude and 6 being the highest. Murmurs with higher amplitudes can sometimes be heard without a stethoscope. Most murmurs are not serious, and many childhood murmurs disappear with time.

Some of the more common murmurs include the pansystolic murmur, late systolic murmur, and the diastolic quadruple gallop. The pansystolic murmur is present between S1 and S2. This interval is known as the systole. This type of murmur can be caused by tricuspid regurgitation, or mitral regurgitation. The pansystolic murmur occupies the whole interval in the systole. Unlike the pansystolic murmur, the late systolic murmur does not occupy the entire systole. Instead, the pansystolic murmur occurs in the latter part of the systole and extends to S2. The late systolic murmur is usually high in frequency, and it is caused by some regurgitant murmur, usually through the mitral valve. The diastolic quadruple gallop is unique compared to the other murmurs in that it has a fourth heart sound whereas the other murmurs only have a third heart sound S3 (third heart sound). S4 (fourth heart sound) is located in the diastole. This murmur has the sound of a galloping horse. Some of the complications that would cause this type of murmur include tricuspid, mitral, and pulmonary valve regurgitation. Depending on the severity and the amplitude of the murmur, an acoustic stethoscope may not be able to detect some of the murmurs.







Step 3. Finding the Solution and Determining the Project Goals.

Based on the problem description we decided to include the following functions and benefits to our project.

- Accept an analog input from a microphone or stethoscope.
- Collect and actual data heart signals. Typically, we would like the device to be able to store and record at least four cycles of real-time data, and be able to filter out any noise caused by the sensors.
- Detect heart murmurs from a trained database of time series.
- Present the results in a format that can make it easy for the user to visualize and analyze.
- Provide a way to perform further analysis in the signals collected.

After analyzing the problem completely we decided to prioritize the goals of the project as follows:

1. Performance and safety
2. Easy of use
3. Portability
4. Reliability
5. Minimum power consummation
6. Cost and availability of parts
7. Minimum maintenance
8. Aesthetics

Finally we concluded that the device was required to perform the following tasks.

- Inform the user if a person has a heart murmur
- Where the heart murmur is located (diastole or systole)
- Heart rate
- Length / Intensity of murmur
- Severity of murmur due to signal characteristics
- Through research determine what type of heart dysfunction can be associated with the murmurs

Step 4. Determining the Product Specifications.

Environmental Specifications		
Specification ID	**Parameter**	**Requirements**
ENV000	Operating Temperature	0 ^{o}C to 40 ^{o}C
ENV001	Storage Temperature	The product will not be degraded if it is exposed to the fallowing environments: Temperature of -40 ^{o}C Temperature of 75 ^{o}C with 0% humidity Temperature of 75 ^{o}C with 60% humidity
ENV002	Operating Relative Humidity	0% to 60%.
ENV003	Vibrations	up to 2,000 Hz
ENV004	Shock	The product will resist the impact with a concrete floor when falling from a 10m height.
ENV005	Maximum Case temperature	The case temperature should not exceed 50 ^{o} C when the product is operating

SYSTEM SPECIFICATIONS	
Specification ID	**Requirement**
SYS000	Weight from 0.2 kg to 0.3 kg
SYS001	Input voltage of 3.0 V DC
SYS002	Power consumption of less than 3 Watts
SYS003	32 Kbytes of memory
SYS004	Serial Communications Interface
SYS005	8 analog input channels
SYS006	4 digital switch inputs with 2-pin headers (uses one of the analog channels).
SYS007	5 servo controller outputs with 3-pin signal, power & ground headers.
SYS008	3 digital inputs/input capture inputs.
SYS009	8 bits of digital output capable of driving a total of 75ma continuous current.
SYS010	8 by 2 pin header array in series with the 8 digital outputs & 8 pull-up resistors.
SYS011	4 memory-mapped digital input & 3 memory-mapped digital outputs.
SYS012	40 KHz clock
SYS013	An 8 pin header making the microcomputer data bus available for external use.
SYS014	Mode switch header, power switch header, reset switch header, battery power header, power-on Red LED header, battery charge header.
SYS015	5 second recording time (~ 5 heart cycles)

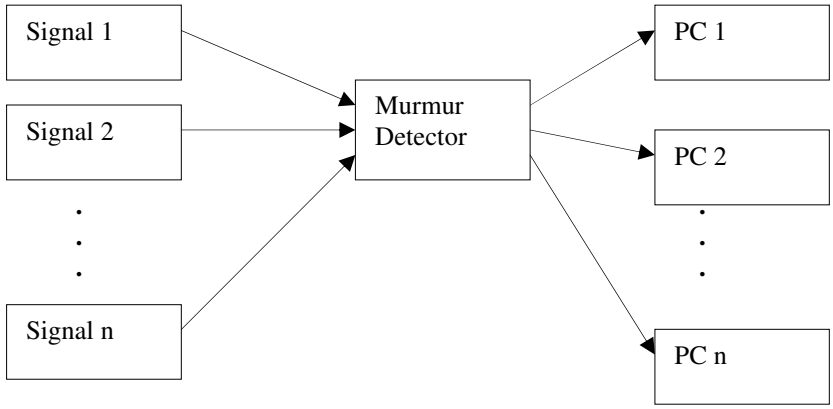
SYS016	A/D conversion
SYS017	1 USB port
SYS018	1 Ethernet port

Step 5. Evaluation of design alternatives

Design A

Our first and most preferred alternative is the murmur detection device that has PC loading, easy use interface, and is relatively inexpensive. This design will compose of both hardware and software design. The hardware will be portable so that data can be transferred to virtually any machine, not just a specific one.

Since this is the most preferred method, the design mechanisms in detail are further explained throughout this report. But in general, the following is a box diagram of how this device is expected to work.



Design B

Our second alternative was to create a software algorithm. This alternative is ideal for someone who already has a PC already has a database of heart signals, and the only thing they would want to do is analyze data efficiently.

Essentially our algorithm would work as follows:

- Data which is loaded onto a sound card, or is saved in an audio file
- User would upload the first audio file
- He would then choose which characteristics of the heart murmur he would like to like to view.
- Characteristics would then upload to a table for easy viewing.

This alternative would best suit someone who doesn't care about waveforms, but instead, they just want the statistical data.

Design C

Finally, our last design alternative is to feed our signals into the box, which will display such features as Heart rate, possible murmur detection, and severity of murmur. This design would not require the purchase of an expensive computer, nor software needed to run such programs on a PC.

Here, our device will have space to store multiple signals and recall features. This approach would be ideal for medical students who would not invest in an expensive device. This, is a very low risk approach to our problem.

Lowest risk approach	Design C
Highest risk approach	Design A
Most difficult approach	Design A
Fall-back design alternative	Design B
Engineering Expo	Design A

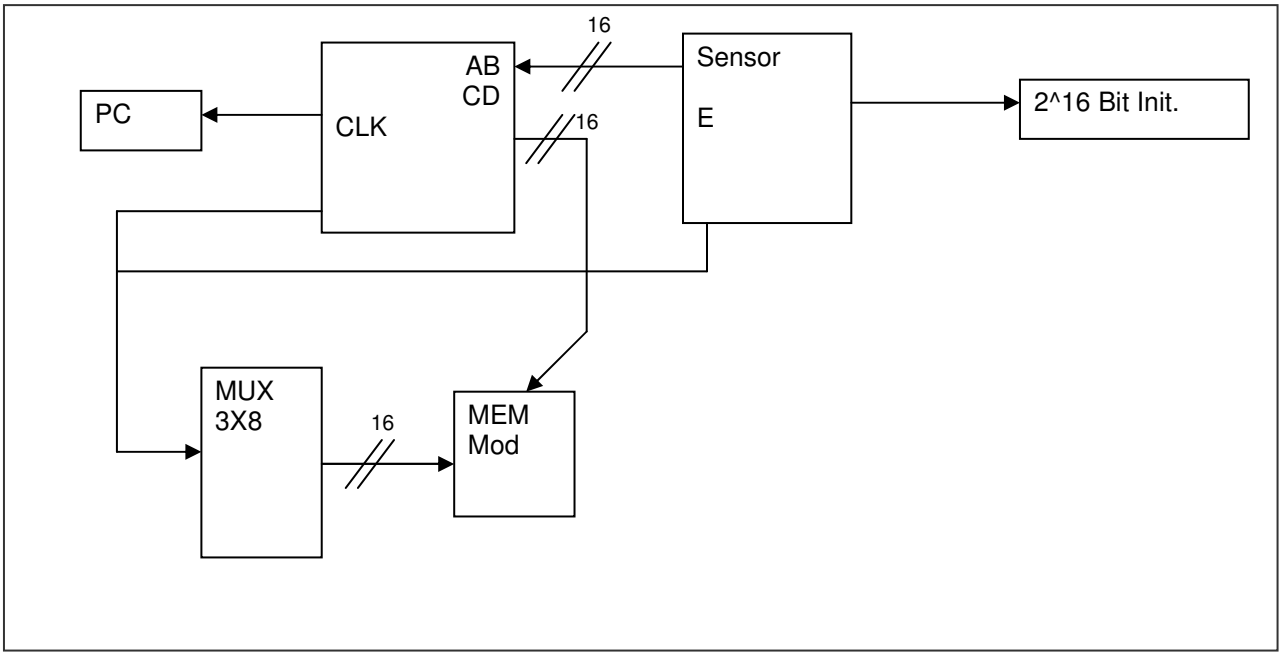
Decision Matrix

		Design Alternatives		
Design Goals	**Weighting factors**	**Design A**	**Design B**	**Design C**
Performance	100	9 / 900	9/900	6/600
Safety	100	7 / 700	9/900	7/700
Easy of use	90	5/450	8/720	8/720
Portability	80	9/720	4/320	9/720
Reliability	70	8/560	5/350	7/490
Minimum Power Consummation	65	5/325	9/585	4/260
Cost	50	7/350	4/200	7/350
Availability of parts	50	6/300	6/300	6/300

Minimum maintenance	40	5/200	4/160	5/200
Aesthetics	30	7/210	5/150	7/210
	Total	**4715**	**4585**	**4550**

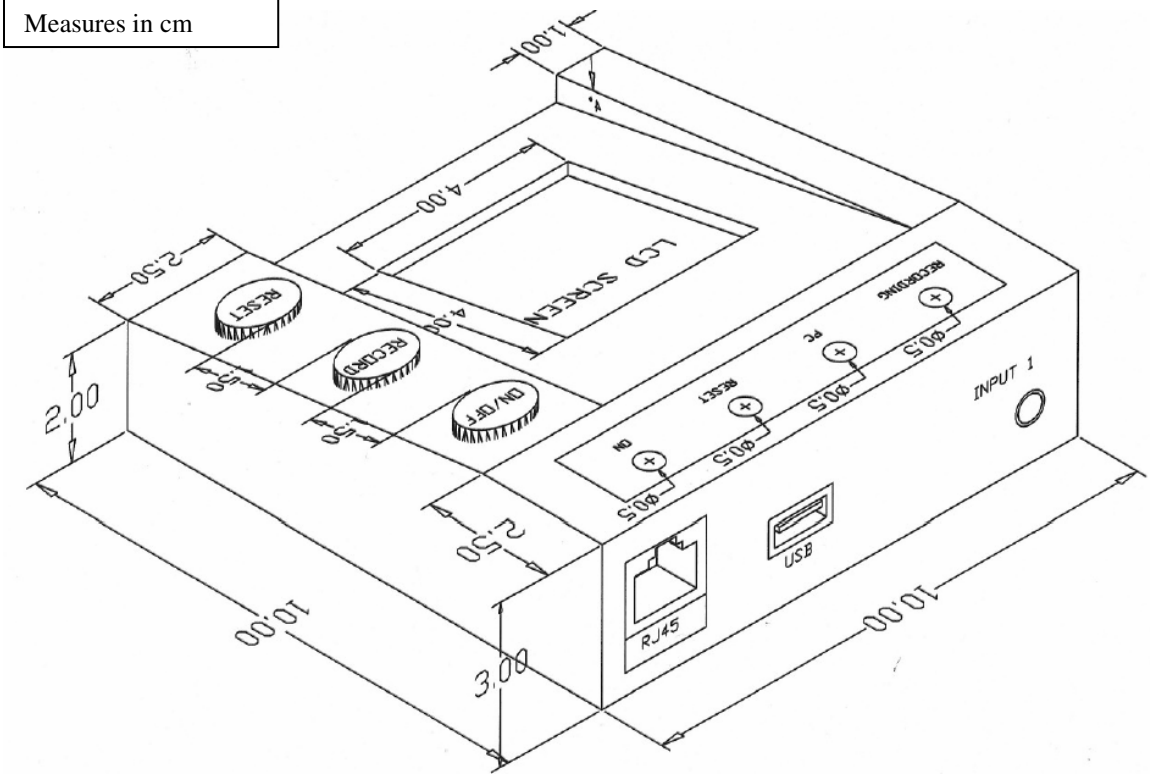
Step 6. Product Design

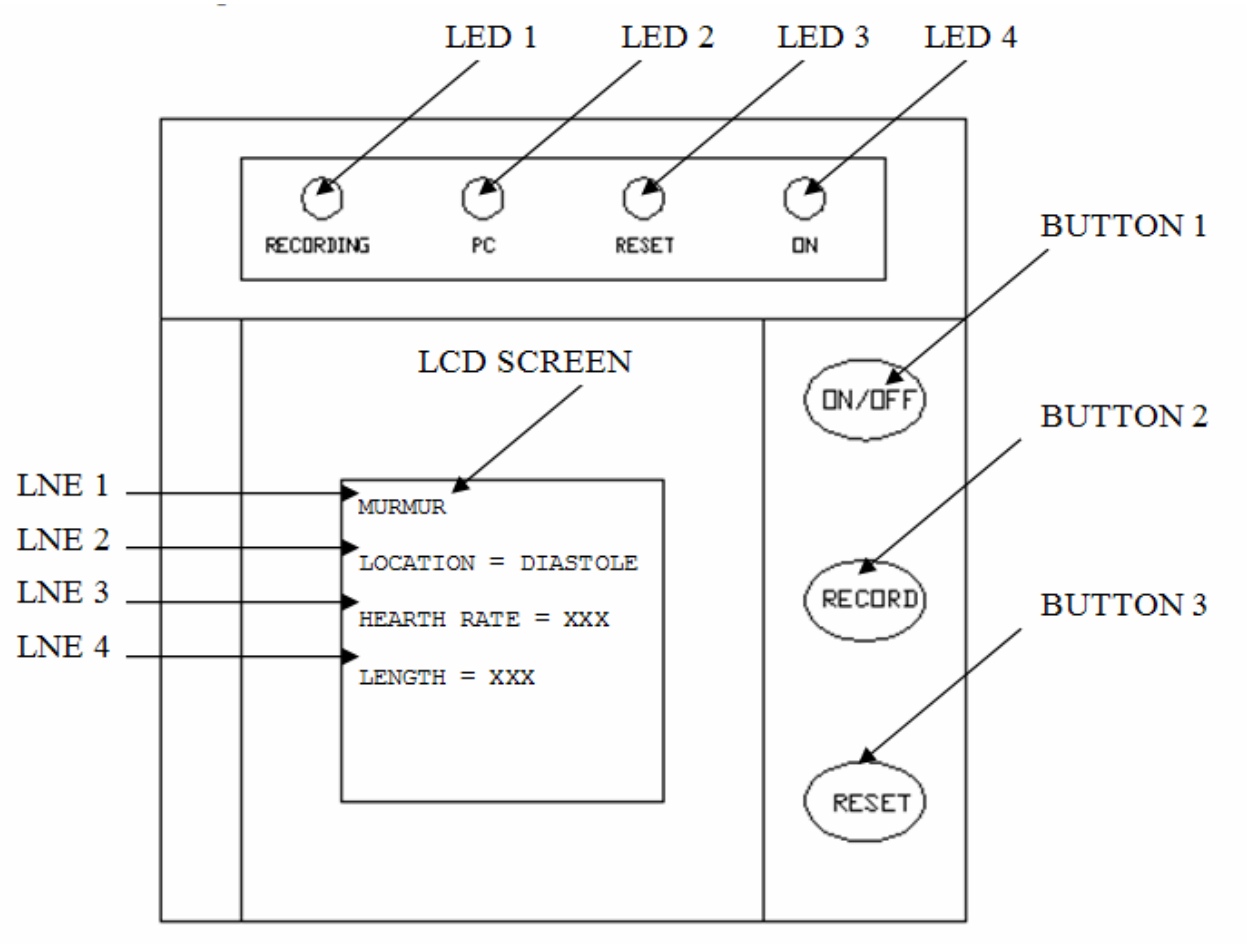
Microcontroller Hardware

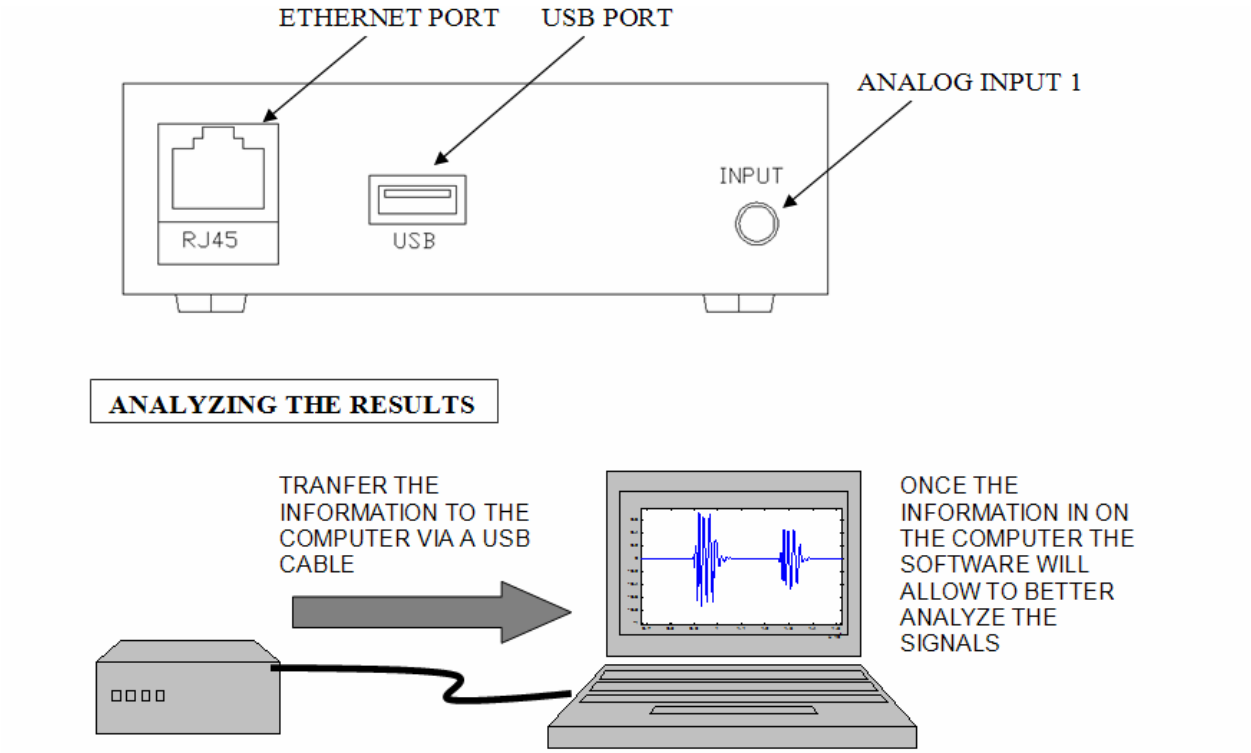


Prototype

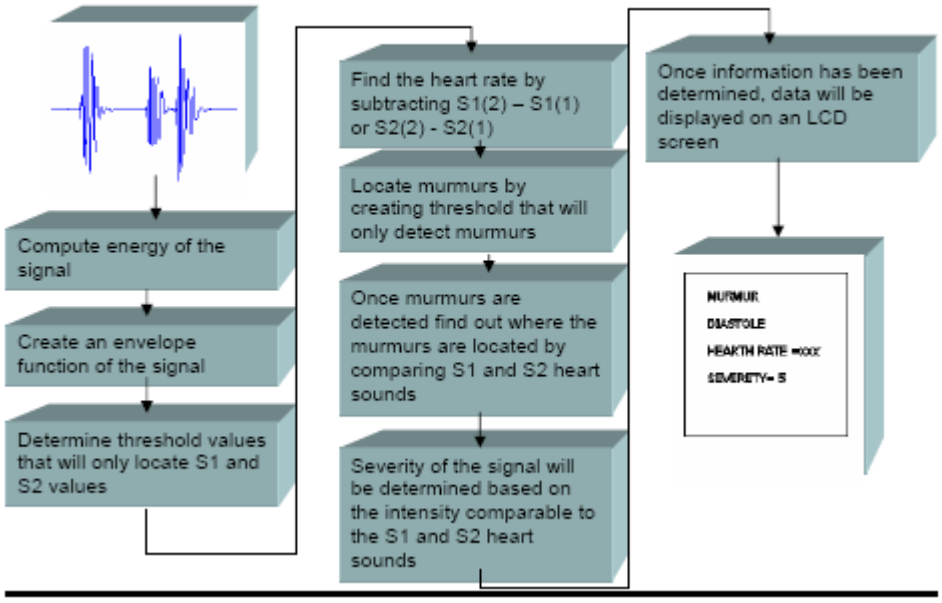
Measures in cm

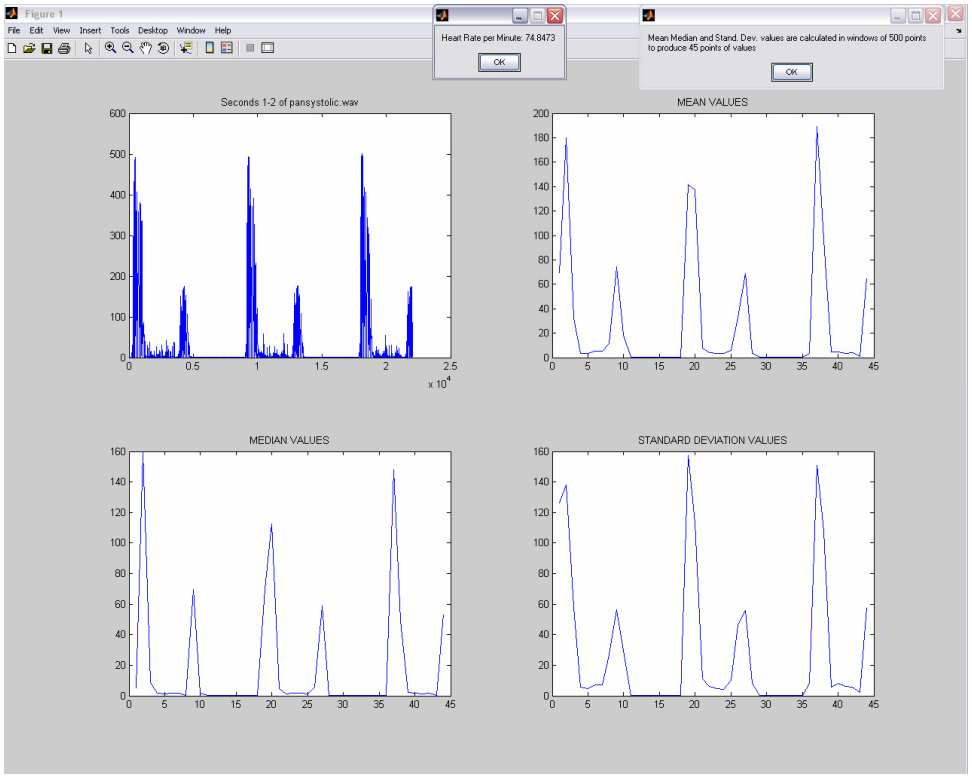



LED 1
LED 2
LED 3
LED 4
RECORDING
PC
RESET
ON
BUTTON 1
LCD SCREEN
ON/OFF
BUTTON 2
LNE 1
LNE 2
LNE 3
LNE 4
MURMUR
LOCATION = DIASTOLE
HEARTH RATE = XXX
LENGTH = XXX
RECORD
BUTTON 3
RESET



Step 7. Development of the Signal Processing Software





Step 8. Purchasing the components:

DEVELOPMENT AND IMPLEMENTATION

Start interacting with the controller.
We started to get familiar with the build in library functions and basic commands

Step 9. Firmware Development

Developing a Test Platform

We build this platform for the purpose of testing and simulation. The platform consisted of a solderless board in which we connected different purpose ICs (logic gates), LEDS and other circuit elements that were needed throughout the development of the project, capacitors resistors. We made connections from the each of the IN's and OUT's of the controller to the board (which remained unchanged for a long period of time). We also attached meters and LED (test proves) in different points of the circuit in order to make it easy to troubleshoot and the platform was an essential part of the infrastructure of our project.

There are several things that are needed to get started. First we had to get a GNUARM tool chain. Also, since we are using Windows rather then Linux, we are required to download CYGWIN, a UNIX emulator.

The other tool that we used is a means to get the programs down to the board. This is done with MC Helper, a program the manufacturer provided to make this step (and others) easy.

This can also be done with sam7util, an open source project.

MC Helper connects to the USB port on the Controller when the controller is in SAM-BA mode. The Make Controller Kit can be put into this mode by shorting out the Erase pins on the Application Board while power is applied. Subsequently, after disconnecting it and reconnecting it from the power it will be running in SAM-BA mode. When in SAM-BA mode, new programs can be downloaded to the SAM7x. MC Helper can also de-activate SAM-BA mode again once the code is downloaded. When the board is again powered down and back up again it will be running the new program.

MC Helper has another very useful function - it allows commands and debugs information to be passed to and from the controller via Ethernet or USB using a protocol called OSC.

We begin the development by starting a new task called "heart_detect", which is part of the main() function of the "Make.c" file. This file is the central of user application level of the controller's operating system. To receive the signal from the condenser microphone, we connect the microphone output to one of the controller input, and put in the following function to the code:

int AnalogIn_GetValue (int *index*)

Parameters:

index An integer specifying which input (0-7).

Returns:

The value as an integer is (0 - 1023).

In our case the input is from port 7. This function will use the built-in A/D converter to convert a voltage level between 0 (common ground) to 3.3 Volts into an integer from 0 to 1023. The stronger the sound from the microphone, the higher the input voltage.

We sample the value from the A/D input at a rate of 200 Hz. We stored these integers into a 1000-slot array, making it capable to store signals of the most recent 5-seconds input. Now the program will attempt to calculate the heart rate, which is the first item to be displayed to the LCD screen. Here are the code sections corresponding to the recording data:

```
  for (i=0;i<MH;i++)
        {
           temp_input = AnalogIn_GetValue(0);

             if (temp_input > 90) // adjustable factor for heart
strength
             {
               rate_store[i] = 'A';
             }
             else if (temp_input > 65)
             {
               rate_store[i] = 'B';
             }
             else
             {
                  rate_store[i] = '0';
             }
```

```
            Sleep (5000/MH);

           if (DigitalIn_GetValue(5) == 0) break;

         }


         Sleep (10);
         Serial_SetChar (12);
         Sleep(10);

         }
```

In this case MH is the number of slots, in this case 1000. If the A/D converter shows an input of higher than 90, it will be considered a "heart beat" and marked that slot in the array 'A'. If it is between 65 to 90, it will be marked as "B" for murmur and noises. Below 65 it is considered no signal and marked as O. Later on, in the program,

```
for (i=0;i<MH;i++)
         {
           if (rate_store[i] == 'A')
           heart_raw_rate = heart_raw_rate + 1;
            else if (rate_store[i] == 'B')
              murmur_qualifier = murmur_qualifier + 1;

         }

           heart_rate = heart_raw_rate * 12;

       // set murmur flag

       if (murmur_qualifier > (heart_rate / 2))
        {
              murmur_flag = 1;
        }
        else
        {
              murmur_flag = 0;
        }
        murmur_length = 5000 * murmur_qualifier / MH;
```

This module will process the array one by one. The heart rate will be calculated by multiply the # of A's * 12, since heart rate is in beats per minute and the device is recording in 5 seconds interval. Also if there are murmur noises which get as strong as the noise of the heart beat, a murmur flag will be set and it will affect the display of the data later.

The LCD will show something like this:

<u>Sample 1:</u>

Heart rate: 77.65
No Murmur
Murmur length: No Murmur

Murmur period: No Murmur

Sample 2:

Heart rate: 84.39
Murmur detected
Murmur length: Systolic
Murmur period: Late

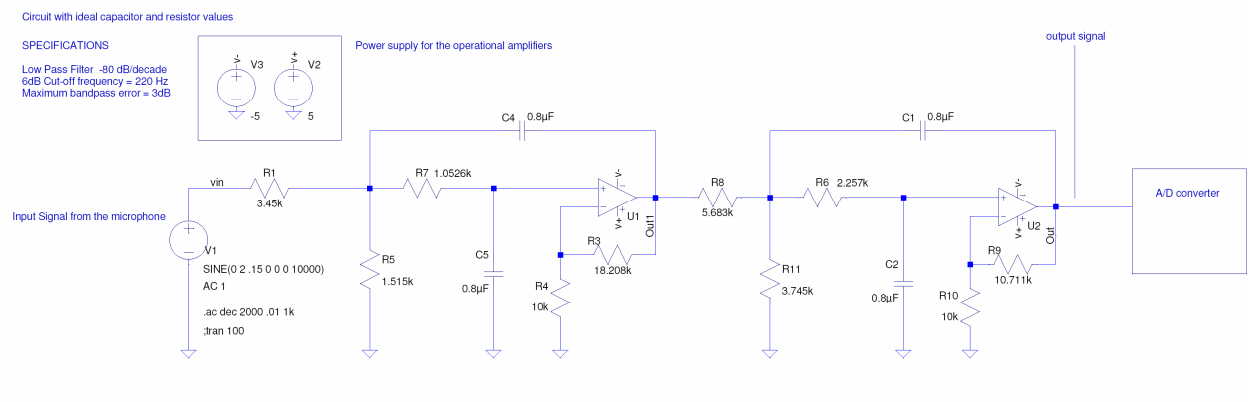
Sample 3:

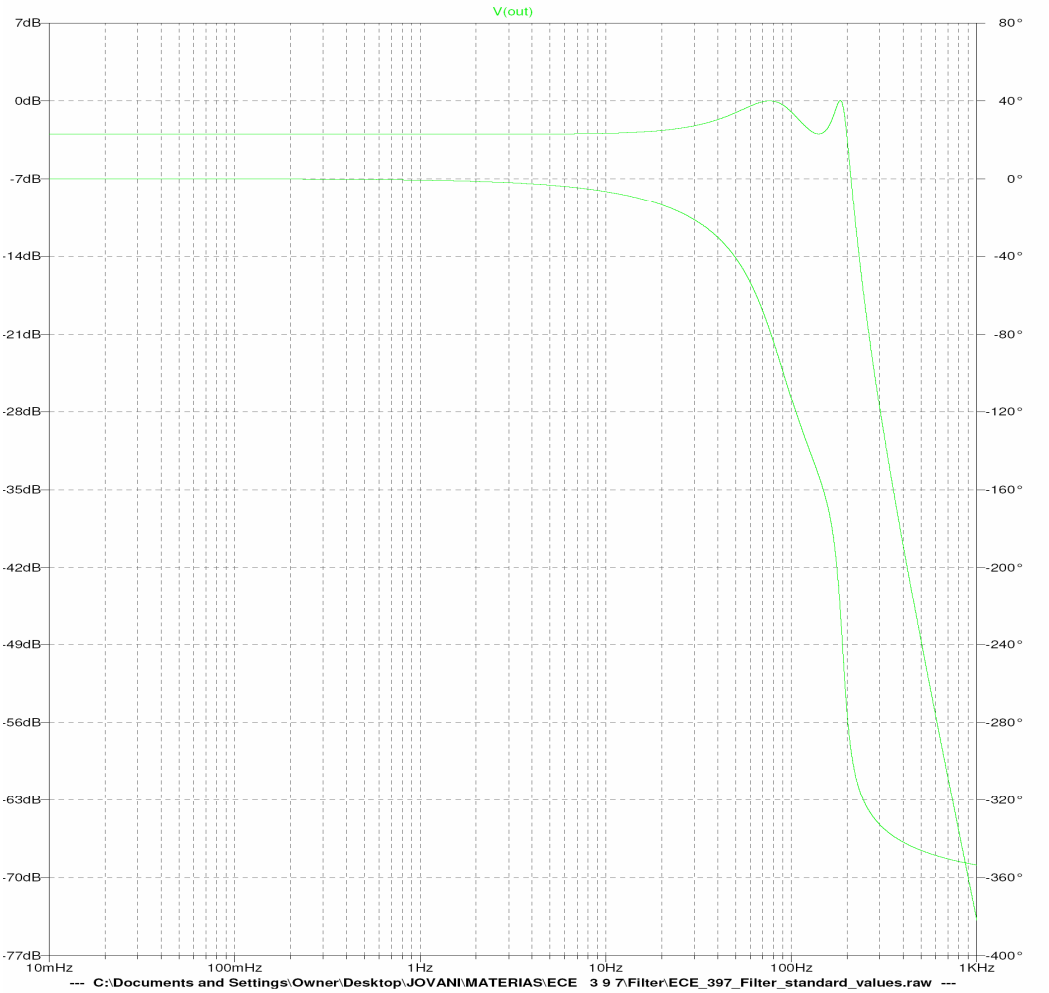
Heart Rate: 68.45
Murmur detected
Murmur length: Diastolic
Murmur period: Early

Construction, Testing and Implementation of the Analog Filter.

We designed a low pass filter to eliminate the noise coming from the microphone and obtain a clean input signal to the A/D converter. With this filter we were able to obtain better signal readings and further optimize the signal processing part of the firmware.

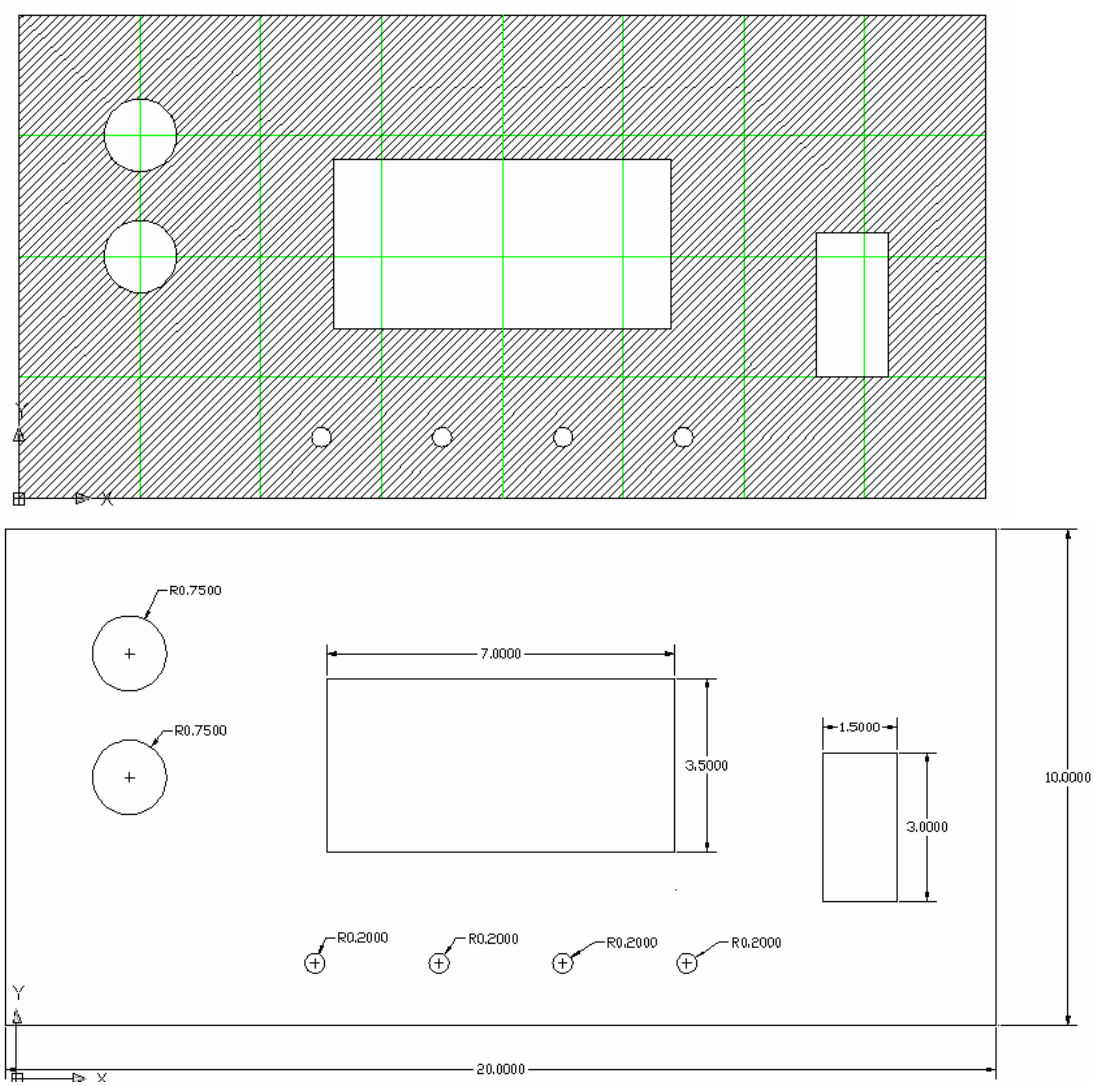
1. Heart signals are usually lower than 150 Hz. Therefore, we decided to have a cut off frequency of 220 Hz leaving some margin.
2. We needed the response to be down by at least 40dB at 300Hz because there was a high energy noise in the 3000-320Hz band. With a 40 dB roll off we were able to attenuate the noise to a level that can't affect the A/D converter output.
3. We decided to make the pass band error of less than 5 dB in order not to distort the signal too much.
4. Based on the specifications I designed a low pass filter made up of 2 separate second order low pass sections.
5. Initially the filter was designed with exact part values, then I changed the part values o standard values. I simulated the filter in LTSpice and obtained the expected response.
6. Next, we obtained the parts and built the circuit in the bread board to test it in the lab using the oscilloscope and the function generator. The results were as predicted.
7. After that we constructed the circuit by soldering all the circuit elements in the circuit board
8. Finally integrated it into the product and run a test to see if it could filter the noise as we expected.
9. We build the filter in a circuit board with standard part values





Construction of the Enclosure.

The following drawings describe the way in which we made the holes to integrate the LCD, buttons and switches for the device.


R0.7500
R0.7500
7.0000
3.5000
1.5000
3.0000
10.0000
R0.2000
R0.2000
R0.2000
R0.2000
20.0000

ECE 397 Project Management.

The following table outlines the timeline of the activities that took place during the second part of out senior design project (ECE 397). The first column contains the initials of the group members that were responsible for each task. The second column contains a description of the task. The blue shaded area represents the time consumed in each task. The black shaded area indicates the spring break.

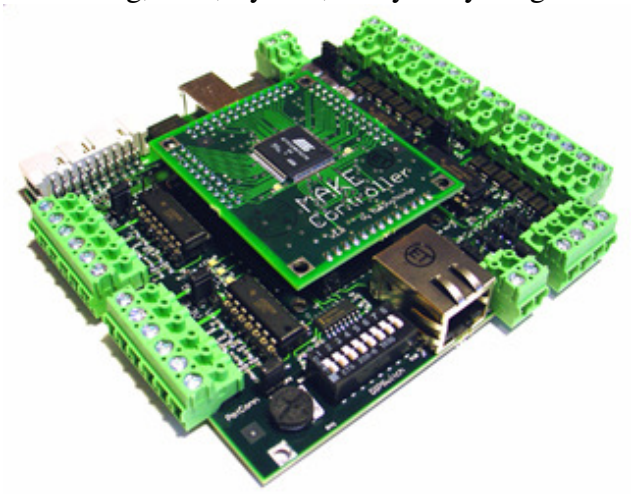
		19-Jan	26-Jan	2-Feb	9-Feb	16-Feb	23-Feb	2-Mar	9-Mar	16-Mar	23-Mar	30-Mar	6-Apr	13-Apr	20-Apr
Members	**TASKS**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**		**11**	**12**	**13**
AL	Buy LCD														
TM	Buy Potentiometer														
JI	Buy 4 status LEDS														
TM	Signal Processing Software (MATLAB)														
TM	Translation to C														
JI, AL	Test Platform with LCD														
AL	Code to store to store and save signals														
AL	Code to download signals														
JI, AL	FIRMWARE to display results in LCD														
JI, AL	FIRMWARE to control status LEDS														
AL	FIRMWARE to interface with PC														
JI, AL	FIRMWARE to control buttons														
JI, AL	**Firmware Test 1**														
JI	Install Potentiometer														
AL	Buy Battery														
AL	Installation of Battery														
JI	Design of Filter														
JI	Buy Enclosure														
TM JI AL	Construct enclosure														
TM JI AL	**Put all component together**														
TM JI AL	Final Modifications														
TM JI AL	**Final Test**														

AL Arnold Lee, JI Jovani Ibarra, TM Tenille Medley

Appendix

About the Hardware Controller

The **Make Controller Kit** is a fully programmable, open source hardware platform for projects requiring high performance control/feedback, connectivity, and ease of use. It can also be used as an interface to a variety of desktop environments like Max/MSP, Flash, and Processing, Java, Python, Ruby - anything that supports open sound control (OSC).



It features:

- **8 analog inputs** - 10-bit inputs read voltages from 0-3.3V while protecting the controller from higher voltages.
- **8 high current digital outputs** - can be configured to drive 8 individual outputs, 4 DC motors, 2 stepper motors, or any combination.
- **4 standard servo controllers** - easily provide external power for driving large loads.
- **8 position DIP switch** - for manual configuration.
- **USB and Ethernet interfaces** - can be used simultaneously.
- **CAN interface** - for networking several boards together.
- **JTAG port** - for on-chip debugging.

. The Make Controller Kit consists of two parts, the **Controller Board**, and the **Application Board**. The Controller Board includes the microcontroller itself, along with its essential components. It physically plugs into the Application Board, which provides a robust interface between the delicate Controller and the devices that connect to it.

About controller firmware programming and controller software stack

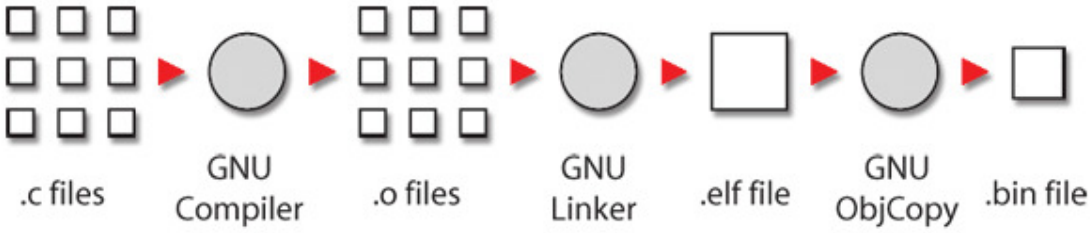
The Make Controller Kit is based on the Atmel AT91SAM7X256 Microcontroller which is an

amazing piece of hardware. Aside from having much more memory than is typical of a microcontroller and being very fast, it has an incredible array of built-in hardware. An Ethernet Mac for networking, a USB port for directly connecting to a local computer, an 8 channel A/D converter and numerous other features. All these features need software around them to activate them and make them useful. When programming a device like this, it really pays to have as much software library help around as possible. We have identified the features most likely to be used by hobbyists, artists and creative professionals, and implemented software routines to help out with these tasks. In addition we have included a RTOS - a software mechanism to assist with the management of critical resources and to make it very easy for a microcontroller to do many things at once and very quickly.

Programming the Make Controller Kit is done in the C programming language. C is a low level language that many, if not most embedded systems are written in. Converting the human readable files into a file that can be uploaded onto the board is a multi-step process that is done by a tool-chain. The Make Controller Firmware has many C files associated with it that perform different functions. Many files are associated with the network functionality, many help provide the basic operating system functions, several provide OSC and USB functions and many provide the simple wrappers that permit complex devices like servos to be controlled with a single line of code.

The firmware projects are designed so that programmers can start by writing their programs in a single file without needing to refer to all these other files, yet if necessary, they are all available in source code. The files in C (usually with a file suffix of .c) are translated into object files (usually with a file suffix of .o) a form the microprocessor can understand, by a compiler. These files are then combined together by a linker into a file in executable format called an ELF file. In the final step, this ELF file is turned into a bin file which can then be uploaded straight to the Controller.

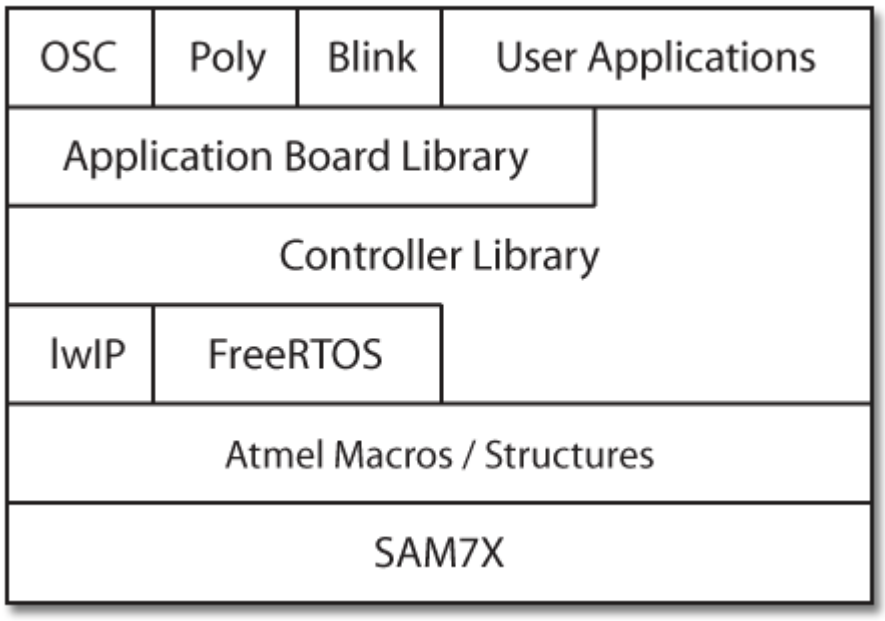
Tool chain



This whole process consists of one invocation of the compiler for each file and one additional invocation each of the linker and ObjCopy utility. This would be very tedious to do manually so there is a utility that is commonly used to automate this process called Make. Because Make is given complete knowledge of all the files involved in the build, it can also be used to considerably shorten build times by only rebuilding the files that need to be rebuilt. The final step is to send the bin file up to the board. This can be done with our utility program MC Helper via USB. This process invokes a special facility on the SAM7X chip called SAM-BA - short for SAM-Boot Assistant. Like most Microcontroller systems, and unlike most desktop computers,

all the code - operating system, network stack, etc. are all compiled and downloaded at once. There is no concept of an OS that is permanently resident in the controller and separate applications that are loaded into it.

Firmware Stack



At the lowest level is the SAM7X microcontroller itself. It is a 32bit ARM7 Thumb Core surrounded by a large number of Atmel peripherals. The programmer's model of the SAM7X includes 37 core registers, Flash Memory, RAM, and hundreds of memory-mapped special registers for controlling the various peripherals.

Reading and writing 100's of registers by address is something that no-one should ever have to do. For example, 0xFFFD8020 is the memory location of the register that contains the last converted value from the Analog to Digital converter. Using this number directly would be very hard to read and maintain. To assist programmers to write more readable code, the Atmel header file (AT91SAM7X256.h) contains the names of all the peripheral registers and all the bits in those registers. It is, as a result, roughly 2700 lines long and quite hard to navigate! Atmel has also provided several macros for doing common tasks to those registers. Most programmers of the Make Controller Kit will never have to deal with the SAM7X at this low level. But people wanting to access peripherals not covered by other layers of the software or who want to use covered peripherals in different ways will need to work at this level.

Free RTOS is an open source operating system that works nicely with the SAM7X to provide memory management, task switching, task synchronization, task communication and some code to talk to the more complex peripherals like USB. Of all the services Free RTOS provides, the perhaps the most valuable and certainly the most complex are the multitasking services.

lwIP is an open source TCP/IP stack. This provides all the software routines needed to create a full featured Internet interface. With this code, the Make Controller Kit can send and receive UPD packets and can support reliable TCP connections. These are the basis for the OSC functions and Web Server code.

The Controller Library creates a higher level of abstraction still from the Atmel structures, lwIP and FreeRTOS. The purpose of this wrapping is to provide access to all these functions using simple consistent function names and in some cases to hide some of the more complex functionality from the user. In addition to the SAM7X's built in peripherals, the Controller board has a few extra pieces of hardware that the Controller Library also provides wrappers for; the EEPROM and the status LED.

The Application Board Library builds on the Controller Library to provide functions that the Application Board provides. The Application library therefore has functions for manipulating the board's inputs and outputs in a variety of ways, controlling Servo motors, etc.

The Blink task is a simple task that uses the routines from the lower layers to Blink the status LED on the controller approximately once a second. The OSC routines also use features provided by the lower layers (like the Network code, USB code, etc.) to provide OSC services. Poly does the same, except it provides small pre-built behaviors.

Finally, User Applications run at this top level, having access to everything else in the system.

GUI Code

```
clear all;
close all;
[FileName,PathName]=uigetfile('*.wav','Please select WAV - File');
A=cat(2,PathName,FileName);
[signal,fs]=wavread(A);
for i=1;
dt1(1,i)=1/fs(1,i); %% each column represents a different signal
t1(1:2*fs(1,i))=(1:2*fs(1,i)).*dt1(1,i); %%time vector
figure(i)
AmpSig(1:2*fs(1,i),i)=1000*abs(signal(1:2*fs(1,i),i).^2);
G='Seconds 1-2 of ';H=FileName;
B=cat(2,G,H)
subplot(2,2,1),plot(AmpSig(1:2*fs(1,i),i)),title(B)
[SigMax(i,1),SigInd(i,1)]=max(AmpSig(1:2*fs(1,i),i));
thresh_1(i,1)=round(0.85*SigMax(i,1));
q=0;
while (AmpSig(q+1,i)<=thresh_1(i,1))& (q < 2*fs(1,i))%% Search for first peak
   q=q+1;
end
Peak1Loc(i,1)=q; %% Store peak location
PeakMask(i,1)=Peak1Loc(i,1)+ 4000;%% create a mask of 4000 pts away from first peak
r=PeakMask(i,1);
while (AmpSig(r,i)<=thresh_1(i,1))&(q < 2*fs(1,i)) %% Search for second peak
   r=r+1;
end
Peak2Loc(i,1)=r; %% Store second peak location
p2p(i,1)=Peak2Loc(i,1)-Peak1Loc(i,1);%% number of samples between 1 beat
%% Heart rate Calculations
samp_sec(i,1)=fs(1,i); %% calculate samples per second
beat_samp(i,1)=1/p2p(i,1); %% calculate beats per sample
RateSec(i,1)=samp_sec(i,1).*beat_samp(i,1); %% calculate heart rate per sec
RateMin(i,1)=RateSec(i,1)*60; %% beats per min
%% Heart rate segment done
HBeat=num2str(RateMin(i,1));J='Heart Rate per Minute: ';
K=cat(2,J,HBeat);
%% Heart rate segment done
%%--------------------------------

%%--------------------------------
%% 2. Murmur Type (Systolic or Diastolic)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate mean, median, standard deviation for each signal
%% For the three dimensional array, n represents number of values for
%% each stat, i represents each signal (1-9). I will use these values
%% to decide location of murmur
n=1;
for m=1:500:(2*fs(1,i)-500)
   %% window size = 500. calulates the mean med and std for all nine
   %% signals over each window size and stores value in columns 1-3 of
   %% stats array
   stats(n,1,i)=max(mean(AmpSig(m:m+500,i)));
   stats(n,2,i)=max(median(AmpSig(m:m+500,i)));
   stats(n,3,i)=max(std(AmpSig(m:m+500,i)));
```

```
    %% StorStat(1-3) array stores mean, med, std, respectively for all nine signals
    %% i.e. StorStat1 is 44x9 array of all the mean values from
    %% stats(n,1,i)
    StorStat1(n,i)=stats(n,1,i);
    StorStat2(n,i)=stats(n,2,i);
    StorStat3(n,i)=stats(n,3,i);
    %figure(i)
    subplot(2,2,2),plot(StorStat1(:,i)),title('MEAN VALUES')
    subplot(2,2,3),plot(StorStat2(:,i)),title('MEDIAN VALUES')
    subplot(2,2,4),plot(StorStat3(:,i)),title('STANDARD DEVIATION VALUES')
    n=n+1;
    msgbox('Mean Median and Stand. Dev. values are calculated in windows of 500 points to produce 45
points of values','replace')

end
msgbox(K)
end
```

Work Cited

1. Vivek Nigam and Roland Priemer, “Simplicity Based Gating of Heart Sounds”.
2.Vivek Nigam and Roland Priemer, “Online Non-Invasive Fetal Sound Analysis”.
3.H. Liang, S. Lukkarinen and I. Hartimo, “Heart Sound Segmentation Algorithm Based on Heart Sound Envelogram”, Computers in Cardiology, pp.105-108,7-10, Sept. 1997
4.Daniel Graupe. Time Series Analysis, Identification and Adaptive Filtering. Krieger. 1989
5.http://cnyack.homestead.com/files/artran/stft2t1.htm
6.http://www.mayoclinic.com/invoke.cfm?objectid=8DD7377A-974F-4C1F-96CC6645AE772151&dsection=2
7.http://www.egeneralmedical.com/listohearmur.htm
8.http://www.3m.com/us/healthcare/professionals/littmann/jhtml/sounds/fourth_heart_sound.jhtml
9.http://www.universityhealth.org/documents
10.http://www.rossonhousemuseum.org/stethoscope_history.html
11.http://www.sciencedirect.com/science12.http://fitca.com/Heart_Murmur/heart_murmur.html
13.http://sln.fi.edu/healthy/stats.html
14.http://www.dpsinfo.com
15.http://www.tmc.edu/thi/anatomy2.html
16.http://www.tmc.edu/thi/anatomy.html
17.http://www.blaufuss.org/ 18.http://www.wikipedia.com/