ClockAide

Joel Jean-Claude (M, CSE), Sachin Honnudike (W, CSE/EE), Anita Ganesan (EE), Eric Moore (EE)

***Abstract* — The goal of ClockAide is to assist special needs students who are learning how to tell time. The design idea stemmed from a presentation given by a teacher from West Springfield Middle School at the University of Massachusetts Amherst. She introduced several problems she encountered while teaching her students and asked the Senior Design Project teams if there were any innovative ideas to assist her in dealing with these issues. To accomplish this, we have designed a three-mode system in which the student will be able to learn setting and reading time. They will also be able to ask the device itself what the current time is. We will create an electronic device that will be simple and ultimately teach students of all levels how to read and set any analog clock.**

I. INTRODUCTION

ClockAide is an educational device in which we use a keypad and a three-mode user interface to solve a specific problem in a class of students with special needs at West Springfield Middle School. However, it can also be applied to students all around the nation who need help reading and setting time. This presents a significant problem as every child needs to learn how to read and set an analog clock, but many have trouble with this concept. Colleen Steele, a social studies teacher at West Springfield Middle School says that, "If kids want to know the time, they pull out their cell phones." In addition, teachers now see that young students have no idea how to read an analog clock since they are mostly exposed to digital clocks at home.[1] Right now, there are websites and some types of toys dedicated to helping children read or know the general hour it is during the day. Along with this, there are two types of educational toys that have been on the market similar to the one that we are producing. The first is named the Onearoo dual clock and nightlight which only tells time by the hour for children but continues their dependence on adults to tell them the correct time. Another device similar to our design is the Momo Clock which is a sleep trainer for young children. This clock tries to implement a regular sleep cycle for small children by showing an image of a sleeping monkey at night to teach the child that they should be sleeping if the eyes are closed.[2] Created around 2008, these are the most recent devices produced and the most popular so far.[3] Megan Ferrari, the teacher we are working with in West Springfield informed us that she teaches the students with cardboard clocks but are seeing that they have difficulty with telling times due to the angle of the hands. In addition to this, she also told us that the position of the minute and hour hands of the clock confuse the children and continually ask her for assistance when staring at the clock. Our end goal is to allow the students to gain confidence and independence while they use this device and create a sense of accomplishment when they start reading the time off an analog clock. The social impacts of our device could be considerable. While we are building our device specifically to help the students with special needs in West Springfield, this product could help students around the country learn this basic.

II. DESIGN

A. *Overview*

SPECIFICATIONS

Specification	Value
Weight	5 lbs
Height	12 inches
Length	18 inches
Width	12 inches
Power Supply	12V DC
Keypad	Detachable

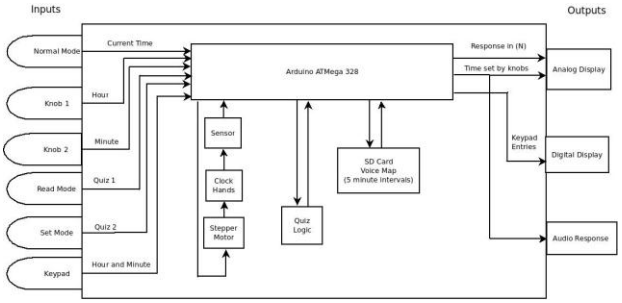


Fig. 1. Current state of the ClockAide. There are knobs that control the hands of the clock, audio feedback through the use of an MP3 shield, and keypad input is functional.

[1] "Massachusetts." *The Republican*. N.p., n.d. Web. 02 Dec. 2012. <http://www.masslive.com/news/index.ssf/2009/12/teachers_say_children_are_beco.html>.

[2] "Sleeptrainer Clock - Momo Monkey." - *UrbanBaby*. N.p., n.d. Web. 02 Dec. 2012. <http://www.urbanbaby.com.au/Sleeptrainer-Clock-Momo-Monkey>.

[3] "American Innovative." - *Company History*. N.p., n.d. Web. 02 Dec. 2012. <http://www.americaninnovative.com/about/>.

To address the challenge of teaching time to children, our group proposes the ClockAide, a device that will interact with a child in a similar way that teachers do. It features three operating modes, Normal, Read, and Set. These modes serve to provide an opportunity for the children to learn how to read time with either a digital or analog display, set time using knobs or a separate keypad, and to check what the current time is. This project was primarily tailored to a class at the West Springfield Middle School, and was expanded to general use for students in the kindergarten through second grade.

We held discussions with Megan Ferrari, the teacher we are working with from West Springfield Middle School. Through these discussions we were able to ascertain what kind of issues she found most challenging when teaching the students how to tell time as well as the difficulties faced by her students when entering their lunch numbers in the cafeteria. By analyzing the two different problems: telling time and practicing with lunch ID Numbers, we came to the conclusion that we could create a system to specifically help the students in her class. We discussed different methods of executing our idea and had to find the proper equipment to begin our design.

To build the ClockAide, our group plans to employ embedded systems technology to implement the features defined. This will provide a platform to combine hardware and software to construct the analog and digital displays, speech prompts, quiz logic, and keypad functionality. To that end, we will use the Arduino microcontroller. It is capable of controlling stepper motors, driving an LCD display, receiving input from a keypad, store voice recordings on an SD card, and contains a real time clock. These are the functions necessary for the ClockAide to address the challenge of teaching time.

Other technologies are available, like the Raspberry Pi. Compared to the Arduino, the Raspberry Pi is more advanced because it features built-in 512 MB RAM, HDMI display, SD card storage, Ethernet and wireless connectivity, and a 700Mhz ARM processor. We chose to start development with the Arduino for this project because the Raspberry Pi was not available for order; the initial supply was low. Once it becomes available, our group plans to use it a new prototype. Currently, the Arduino solution uses an ATMega328 microchip featuring a 16Mhz processor, 2KB SRAM, 1KB EEPROM, 20 digital and analog I/O pins and 32KB flash memory.

These design considerations are accomplished using knowledge from introduction to programming (CMPSCI 121), Digital System Design (ENGIN 112), Computer Systems Lab I & II (ECE 353 & 354), Electronics I (ECE 323), Circuit Analysis I & II (ECE 211&212), and Software Engineering (ECE 373).

B. Normal Mode

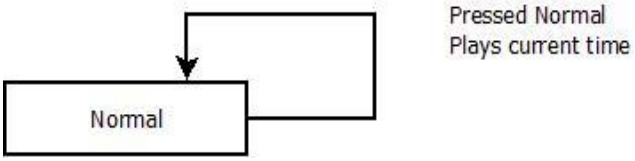
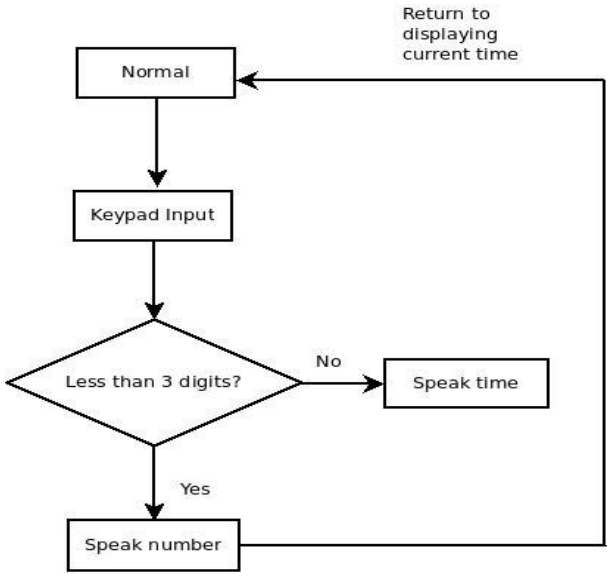


Fig. 2. When the normal button is pressed, the ClockAide speaks out the current time

This mode's primary function is to display the current time. If a child is interested in knowing what the time is, they press the Normal mode button, and the ClockAide will speak it out.

Pressing the Normal mode button will send a signal to the microcontroller to pull up the appropriate voice recording of the time displayed. This recording will be stored on an SD Card. Once the recording is retrieved, the resulting output signal will be decoded using an MP3 Shield, then sent to speakers, transmitting the appropriate sound back to the child.



If a user dials numbers on the keypad, the ClockAide will speak out the number. As shown in figure 3, there will be logic in place to determine whether there are enough digits to speak out a number, or a time.

Fig. 3. Keypad activity in Normal mode

C. Read Mode

This is one of the two modes that are a subset of the Quiz Mode proposed in the Preliminary Design Review. Read mode tests the cognitive abilities of the user. In this mode, the clock chooses an arbitrary time, and prompts the user to submit a response through the keypad. If the answer is correct, ClockAide acknowledges this, and gives the user the choice to try again. If the answer is incorrect, ClockAide would give the

user two additional tries. If both are unsuccessful, ClockAide would speak out the correct time.

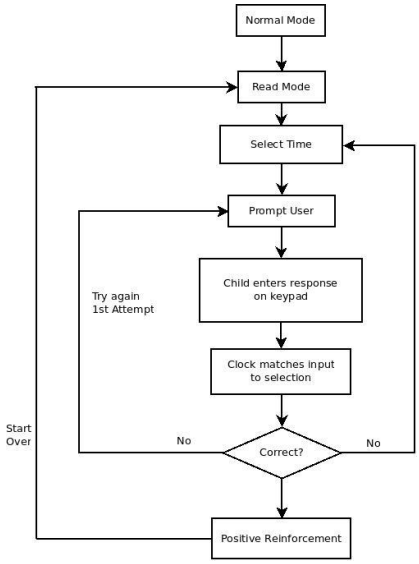


Fig. 4. Read mode logic

D. Set Mode

It is the second of the two quiz modes. In this mode, the clock chooses an arbitrary time, then prompts the child to submit a response through two knobs; the hour hand, then the minute hand. Based on a prior discussion with the teacher, the student is expected to place the hour hand in the correct position relative to the minute hand. For example: to set 2:30, the child places the hour hand between 2 and 3, then the minute hand on the 30 minute mark. Therefore, this will simulate the actual position of a clock at 2:30.

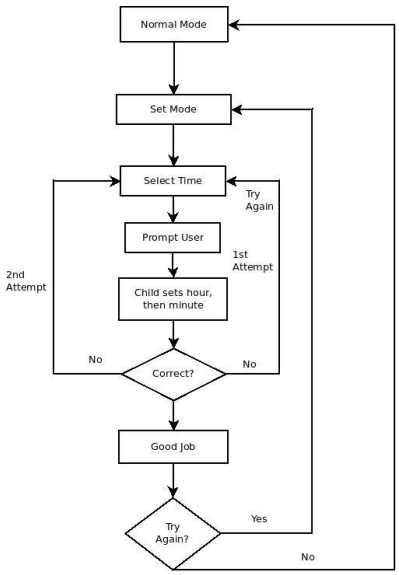


Fig. 5. Logic pattern for set mode

The setting of the hands is done through knobs which transfer the mechanical energy from the hand, to an electrical signal that is passed to a microcontroller, which in turn, controls the stepper motors that move the hands of the clock.

E. Lunch Numbers

This is the mode that is primarily driven by the keypad. The child will enter his or her ID number, and the system would provide a response through the screen. All the student ID numbers would be stored in an SD card as a CSV file. This file can be edited by the teacher through Microsoft Excel, allowing her to update the list every year when she gets new students.

F. Stepper Motors

Stepper motors are brushless DC electric motors that divide a full rotation into a number of equal steps. In our design, we make use of two unipolar stepper motors (one driving the hour hand and the other driving the minute hand) in order to achieve a good representation of time and also to enable us control the arms of the clock using knobs. This design choice was made bearing in mind the relatively simpler complexity of the driver circuit of the unipolar stepper motor versus that of a bipolar stepper motor. Both of these motors have full rotations that are divided into 200 steps.

In order to translate the number of steps moved by the motor to a time, a C++ library (Appendix 1) was written along with its corresponding header file. This library primarily contained logic to drive the motor and to move the motor by a specific number of steps. The number of steps required to set the position of the hour hand is computed using the following formula:

*N = (hour/3)*50 + calculateHourOffset(hour%3) + calculateExtraMinuteOffset(minute)*

The first portion of the formula, *(hour/3)*50*, uses integer division to divide the face of the clock into 4 quarters (each of these quarters contains 3hours) and positions the hour hand in one of those quarters.

The second portion, c*alculateHourOffset*(*hour%3)*, is a function call that uses modular arithmetic to determine which of the hours within the quarters to move the hand to.

The last portion, *calculateExtraMinuteOffset(minute)*, is a function call to adjust the position of the hour hand to compensate for the actual number of minutes that have elapsed since the beginning of the hour.

The number of steps required to set the position of the minute is computed using the following formula:

*N = (minute/3)*10 + calculateMinuteOffset(minute%3)*

The first portion of the formula, *(minute/3)*10*, uses integer division to divide the face of the clock into 20 sectors (each of

these sectors containing 3 minutes) and positions the minute hand in one of these sectors.

The second portion, *calculateMinuteOffset(minute%3)*, is a function call that uses modular arithmetic to determine which of the minutes within the sector to move the hand to.

In Normal mode, both motors move sequentially to provide a correct display of time. In read mode, the position of the motors is set by the microcontroller to enable the student read the time and in set mode the motors are controlled by knobs, used by the student to set the time.

The stepper motors are of the unipolar type, and are used with optical sensors to mark a reference point for rotation. This reference point is set to 12 (noon). This is required for the proper operation of the set mode. These motors can rotate either clockwise, or counterclockwise. It is preferred, however, that the students set the time in the clockwise direction.

G. Keypad

This component is primarily used to allow the children to practice entering their lunch numbers. However, this will also be a second input to the ClockAide for the quiz modes. During Normal mode, if a child dials a time, like 2:30 for example, the ClockAide will speak this time out, and provide a display on an included LCD screen. In addition, they can prompt the ClockAide to speak the current time by pressing the 'Enter' button. During Set mode, this procedure will be used to enter responses. After the child has been prompted to set the clock to a specific time, the child dials their entry on the keypad. The hands of the clock will move accordingly. For Read mode, the child will dial their response on the keypad.

To make the keypad operational, it had to be reverse engineered. This part was purchased from a private vendor, so it did not include proper documentation, or power cables to operate it. Upon detailed inspection, the keypad consisted of three main parts: a button matrix, an LCD screen, and a microcontroller working as an interface for the buttons to the display. For our purposes, the button matrix was repurposed to work with the Arduino microcontroller. After the pins were mapped to the buttons, it was possible to connect it to the Arduino Uno for software development.

This particular keypad was chosen because it was the only one the group found that closely resembled the one at the lunch room at West Springfield Middle School. The buttons are soft to the touch, and are clearly labeled. Our group was not able to obtain an exact model because the extras were discarded, and acquiring a new one would cost over $300.

Fig 6. Left: Keypad currently in use. Right: ClockAide Keypad

H. Time Stamp Playback

Ms. Ferrari recorded all of the hours, minutes (5-minute intervals), and wildcards (ex: o'clock and noon) in a single audio file. Megan's voice was used so that the audio output will be natural and similar to what the students would hear when they practice in class with Ms. Ferrari. The open-source audio editor Audacity was used to process the audio file and remove background noise. The file was amplified prior to being broken into separate MP3 encoded files for each hour and minute. These MP3 files were placed on a micro SD card into separate folders (ex: hours, minutes, wildcard). The Arduino MP3 shield is being used to playback the time. The MP3 shield features a micro SD card slot where MP3 files can be read from an SD card. The MP3 shield also has an audio output where speakers can be connected. The MP3 library was modified so that multiple MP3 files can be played back to back with the correct amount of delay in between. This will ensure that a complete timestamp can be played back (ex: 12:35). The delay between the hour file and minute file was adjusted so that the playback is clear and natural. The modification to the library saves time and space since all of the possible timestamps do not need to be recorded. The individual hour and minute combination can be used to playback any timestamp. An Arduino sketch was written (Appendix 4) to take in a timestamp from serial input and play it back via audio out. This was demonstrated during our PDR presentation.

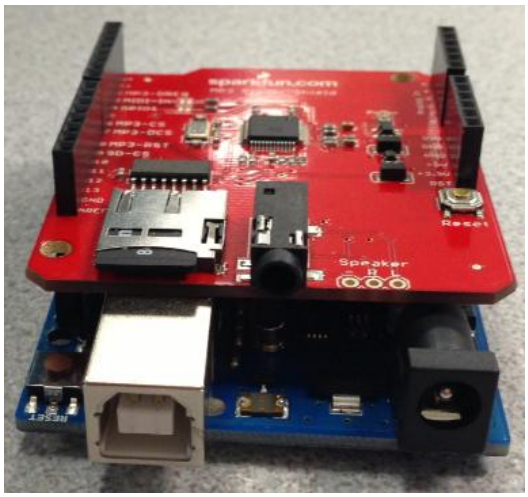


Fig. 7 The SD card shield attached to an Arduino Uno.

III. Class Visit

On 10/31/12 and 11/1/12, our group made two visits to the class at the West Springfield Middle School. The objective of these meetings was to get feedback on how children are taught how to tell time, entering their lunch numbers, and what other features needed to be included in the ClockAide. Eric and Sachin went first, and they gathered data on how children completed worksheets on times, and what would be the best way to interact with the ClockAide. The following were their observations:

- Knobs are better than buttons to set the time
- Disabilities range from Cerebal Palsy, Dislexia, ADHD
- Operational capacity of 2nd grade student
- Students ask the teacher what the time is very frequently

Joel and Anita visited the class for the second day, and gathered the following information:

- Voice recording of the teacher for the analog display of the clock
- Times in the third quadrant(between 9 and 12) of the display tend to be most difficult for the children to read
- Children have a very tough time matching words, colors, and pictures
- Keypad was attached to a POS system that uses picture ID to verify each entry
- Hands of the clock should be clearly identified using different sizes and colors for the hands
- The clock design should be age-appropriate; it should not look like a toy from Toys-R-Us

All of these observations were acknowledged, and tailored the design of the ClockAide to address them. Our project is intended to make a positive impact in the class of students in the West Springfield Middle School. These students range from 6th-8th grade, but are at 2nd grade operating capacity. They have special needs ranging from dislexia, ADHD, motor handicap and Cerebral Palsy. After having gotten feedback from the teacher, our project can accommodate many children in the mainstream school system.

When researching the market, our group found that there are no products that offer the features the ClockAide includes. Most training clocks are designed with the purpose of teaching children when the appropriate time to get up in the morning is. This involves a display that uses colors that represent different actions. In the case of the One-a-roo clock, the colors represent two states: sleeping, and waking up. This is tailored to parents who have kids that tend to wake up in the middle of the night, and want the clock to relieve them of the hassle of getting out of bed to putting the kids back to sleep. The ClockAide is the only device of its kind that aims to teach the child to learn how to read and set time.

IV. Project Management

MDR GOALS

Goal	Status
Stepper Motor Characterization	Complete
Voice Mappings	Complete
Keypad Configuration	Complete

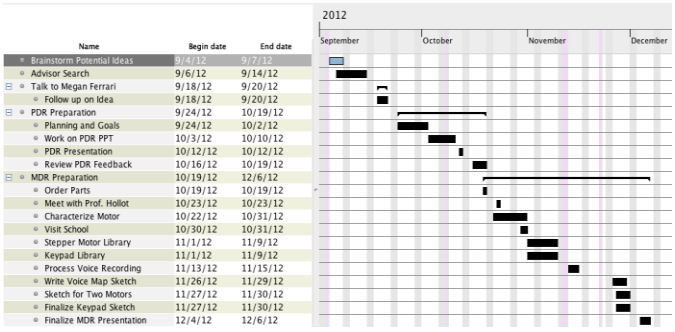


Fig. 8. Gantt Chart for Fall Semester

During PDR our team set the following goals: Characterize the Stepper Motor, Configure the keypad and map voice recording to specific keypad inputs. We have successfully achieved each of these goals and through refinement during lab bench meetings with the course coordinators we were able to achieve more than the stated goals. These achievements include (and are not limited to) complete implementation of Normal mode and Read sub-mode functionality, and partial implementation of Set sub-mode functionality.

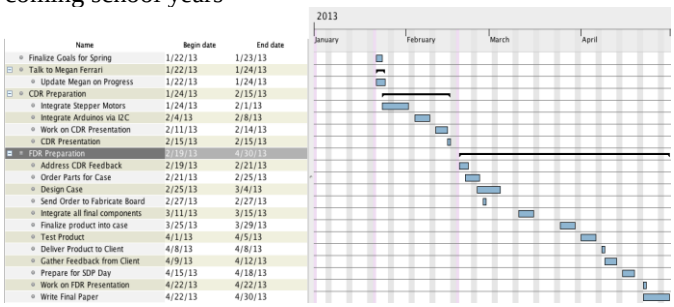
A. Team Member Roles

The roles of the each of our team members combine each of our talents and skills into four independent parts that together with much design and reiteration will create ClockAide. The system has been deconstructed into the following subsystems: the Detachable Keypad with LCD Screen, the Analog Clock with outer casing, Arduino controller for Stepper Motor and Gears. Our visit to West Springfield Middle School revealed that the keypad system used in the lunch room retails for about $500. Due to our budgetary constraints, this proved to be impractical for our design. We decided to buy a keypad which bore significant visual resemblance to the actual keypad used in the school. On its arrival, we discovered it included an invalid power supply unit thus inhibiting our ability to test it in its original configuration. We decided to salvage some parts from the unit and use these parts to create our own keypad with an Arduino Uno and a Serial Enabled LCD screen.

During the initial stages of our project, we divided our efforts between aspects of the clock user interface and keypad usage. Eric took the lead on the stepper motors and developed a scheme to drive them. Sachin worked extensively on the voice recordings; cutting and editing them for eventual use in the

system. Joel and Anita did much research into different types of keypads that were available on the market, as well as the appropriate interfaces to use for connecting it to the main unit of the ClockAide. As the project progressed, Sachin developed libraries for the playback of the voice mappings. Eric developed the libraries for the stepper motor and worked with Sachin in coding these libraries with a view to implement our unique functionalities. Extensive work was done in calculating various parameters relating to how the stepper motors would drive the hands to achieve an accurate representation of time. Joel and Anita reverse engineered the keypad. This was a very involved process as the keypad had no datasheets or documentation. They also had to consider ways to save usage activity to collect data to feedback options for the student while they use the clock in quiz mode and while they practice entering the lunch ID numbers. The LCD screen initially obtained by Joel short circuited a week before the Midyear Design Review presentation. This created a bit of an obstacle for us because we were unable to have another one delivered and configured in time for the demonstration. In its place, Sachin suggested that we connect the keypad in a way so that the display would be on the monitor of a computer. This allowed us to demonstrate the way our keypad audio feedback worked as well as the way the number would appear for the lunch ID numbers.

Each of the team members worked equally on software and hardware sections on each of the components. We debugged each other's code and eventually were able to get the code for the stepper motor, audio feedback and keypad components for each deliverable. From now onwards, we will work on attaching the new LCD screen to the Arduino Uno and focus on the double checking system involved in entering the lunch ID numbers. We must create a way to organize, and report the progress of each student while they use the read and set quiz modes and be able to save such a file for the teacher on the microSD card. Sachin had the idea to use a Comma-Separated Values file (CSV) that stores data (numbers and text) in plain - text format. He believes that this would be the simplest way for the teacher to interpret the data and can be used to transfer data as an excel spreadsheet. One minor problem that we discovered on our visit to the school however, is that each year new students are given new lunch ID numbers so the data must be relatively easy to change by the teacher and thus continue to track each students progress throughout the coming school years



Name	Begin date	End date
Finalize Goals for Spring	1/22/13	1/23/13
Talk to Megan Ferrari	1/22/13	1/24/13
Update Megan on Progress	1/22/13	1/24/13
CDR Preparation	1/24/13	2/15/13
Integrate Stepper Motors	1/24/13	2/1/13
Integrate Arduinos via I2C	2/4/13	2/8/13
Work on CDR Presentation	2/11/13	2/14/13
CDR Presentation	2/15/13	2/15/13
FDR Preparation	2/15/13	4/30/13
Address CDR Feedback	2/19/13	2/21/13
Order Parts for Case	2/21/13	2/25/13
Design Case	2/25/13	3/4/13
Send Order to Fabricate Board	2/27/13	2/27/13
Integrate all final components	3/11/13	3/15/13
Finalize product into case	3/25/13	3/29/13
Test Product	4/1/13	4/5/13
Deliver Product to Client	4/8/13	4/8/13
Gather Feedback from Client	4/9/13	4/12/13
Prepare for SDP Day	4/15/13	4/18/13
Work on FDR Presentation	4/22/13	4/22/13
Write Final Paper	4/22/13	4/30/13

Fig. 9 Gantt Chart for Spring Semester

V. Conclusion

ClockAide will provide a fun educational device for students with special needs in West Springfield Middle School but will also be used to teach people of all ability levels how to read and set an analog clock. We made arrangements to meet with Megan Ferrari during school hours to observe and discuss various design specifications necessary in the successful development of ClockAide.

At this moment, our project is in two independent states. For MDR we will be presenting the operation of Normal, Read and Set modes using the stepper motors. We will also present the audio feedback to display the input from a keypad emitted through a speaker. Lastly, we intend on showing the panel how the stepper motors keep current time in sync with the Arduino Uno.

For the future, we shall start making heavy strides towards casing and building a board that comprehensively encompasses all the independent parts. Integrating the independent parts will be a challenge since none of our team members have experience with this process. Therefore, with the help of our adviser, Professor Soules, and M5 staff we will be able to find the best way to proceed with this part of the project. This will require us to manage our time over the next several weeks, especially over winter break. We shall keep in contact with the staff at West Springfield Middle School in order to get more feedback and more insight so that our device can effectively solve their problems for as long as they choose to use it.

VI. Appendix

1) *EasyStepper.cpp*

```
/*
  EasyStepper.cpp - - Stepper library for Wiring/Arduino - Version 0.4

  Modification for SDP                    (0.1) by Eric Moore
  Add findZero()                          (0.2) by Eric Moore
  Add moveToMinute(int minute)            (0.3) by Sachin Honnudike and Eric Moore
  Add moveToHour(int hour)                (0.4) by Sachin Honnudike and Eric Moore
  Add normalHour(int hour, int minute);   (0.5) by Eric Moore
  Add normalMinute(int minute);           (0.6) by Eric Moore

  Drives a unipolar or bipolar stepper motor using  2 wires or 4 wires

  When wiring multiple stepper motors to a microcontroller,
  you quickly run out of output pins, with each motor requiring 4 connections.

  By making use of the fact that at any time two of the four motor
  coils are the inverse  of the other two, the number of
  control connections can be reduced from 4 to 2.

  A slightly modified circuit around a Darlington transistor array or an L293 H-bridge
  connects to only 2 microcontroler pins, inverts the signals received,
  and delivers the 4 (2 plus 2 inverted ones) output signals required
  for driving a stepper motor.

  The sequence of control signals for 4 control wires is as follows:

  Step C0 C1 C2 C3
     1  1  0  0  0
     2  0  1  0  0
     3  0  0  1  0
     4  1  0  0  1

  The sequence of controls signals for 2 control wires is as follows
  (columns C1 and C2 from above):

  Step C0 C1
     1  0  1
     2  1  1
     3  1  0
     4  0  0

  The circuits can be found at

http://www.arduino.cc/en/Tutorial/Stepper
 */


#include "Arduino.h"
#include "EasyStepper.h"

/*
 * two-wire constructor.
 * Sets which wires should control the motor.
 */
EasyStepper::EasyStepper(int number_of_steps, int motor_pin_1, int motor_pin_2)
{
  this->step_number = 0;      // which step the motor is on
  this->speed = 0;        // the motor speed, in revolutions per minute
  this->direction = 0;      // motor direction
  this->last_step_time = 0;    // time stamp in ms of the last step taken
  this->number_of_steps = number_of_steps;    // total number of steps for this motor

  // Arduino pins for the motor control connection:
  this->motor_pin_1 = motor_pin_1;
  this->motor_pin_2 = motor_pin_2;

  // setup the pins on the microcontroller:
  pinMode(this->motor_pin_1, OUTPUT);
  pinMode(this->motor_pin_2, OUTPUT);

  // When there are only 2 pins, set the other two to 0:
  this->motor_pin_3 = 0;
  this->motor_pin_4 = 0;

  // pin_count is used by the stepMotor() method:
  this->pin_count = 2;
}


/*
 *   constructor for four-pin version
 *   Sets which wires should control the motor.
 */

EasyStepper::EasyStepper(int number_of_steps, int motor_pin_1, int motor_pin_2, int motor_pin_3, int motor_pin_4)
{
  this->step_number = 0;                  // which step the motor is on
  this->speed = 0;                        // the motor speed, in revolutions per minute
  this->direction = 0;                    // motor direction
  this->last_step_time = 0;                   // time stamp in ms of the last step taken
  this->stepper_position = 0;                 // Number of steps that have gone by/ Keep track of stepper motor position
```

```
  this->number_of_steps = number_of_steps;     // total number
of steps for this motor

  // Arduino pins for the motor control connection:
  this->motor_pin_1 = motor_pin_1;
  this->motor_pin_2 = motor_pin_2;
  this->motor_pin_3 = motor_pin_3;
  this->motor_pin_4 = motor_pin_4;

  // setup the pins on the microcontroller:
  pinMode(this->motor_pin_1, OUTPUT);
  pinMode(this->motor_pin_2, OUTPUT);
  pinMode(this->motor_pin_3, OUTPUT);
  pinMode(this->motor_pin_4, OUTPUT);

  // pin_count is used by the stepMotor() method:
  this->pin_count = 4;
}

/*
  Sets the speed in revs per minute

*/
void EasyStepper::setSpeed(long whatSpeed)
{
  this->step_delay = 60L * 1000L / this->number_of_steps /
whatSpeed;
}

/*
  Moves the motor steps_to_move steps.  If the number is
negative,
   the motor moves in the reverse direction.
 */
void EasyStepper::step(int steps_to_move)
{
  int steps_left = abs(steps_to_move);  // how many steps to
take

  // determine direction based on whether steps_to_mode is +
or -:
  if (steps_to_move > 0) {this->direction = 1;}
  if (steps_to_move < 0) {this->direction = 0;}


  // decrement the number of steps, moving one step each time:
  while(steps_left > 0) {
  // move only if the appropriate delay has passed:
  if (millis() - this->last_step_time >= this->step_delay) {
      // get the timeStamp of when you stepped:
      this->last_step_time = millis();
      // increment or decrement the step number,
      // depending on direction:
      if (this->direction == 1) {
        this->step_number++;
        if (this->step_number == this->number_of_steps) {
          this->step_number = 0;
        }
      }
      else {
        if (this->step_number == 0) {
          this->step_number = this->number_of_steps;
        }
        this->step_number--;
      }
      // decrement the steps left:
      steps_left--;
      // step the motor to step number 0, 1, 2, or 3:
      stepMotor(this->step_number % 4);
      this->stepper_position++;
    }
  }
}

/*
 * Moves the motor forward or backwards.
 */
void EasyStepper::stepMotor(int thisStep)
{
  if (this->pin_count == 2) {
    switch (thisStep) {
      case 0: /* 01 */
      digitalWrite(motor_pin_1, LOW);
      digitalWrite(motor_pin_2, HIGH);
      break;
      case 1: /* 11 */
      digitalWrite(motor_pin_1, HIGH);
      digitalWrite(motor_pin_2, HIGH);
      break;
      case 2: /* 10 */
      digitalWrite(motor_pin_1, HIGH);
      digitalWrite(motor_pin_2, LOW);
      break;
      case 3: /* 00 */
      digitalWrite(motor_pin_1, LOW);
      digitalWrite(motor_pin_2, LOW);
      break;
    }
  }
  if (this->pin_count == 4) {
    switch (thisStep) {
      case 0:    // 1000
      digitalWrite(motor_pin_1, HIGH);
      digitalWrite(motor_pin_2, LOW);
      digitalWrite(motor_pin_3, LOW);
      digitalWrite(motor_pin_4, LOW);
      break;
      case 1:    // 0100
      digitalWrite(motor_pin_1, LOW);
      digitalWrite(motor_pin_2, HIGH);
      digitalWrite(motor_pin_3, LOW);
      digitalWrite(motor_pin_4, LOW);
      break;
      case 2:    //0010
      digitalWrite(motor_pin_1, LOW);
      digitalWrite(motor_pin_2, LOW);
      digitalWrite(motor_pin_3, HIGH);
      digitalWrite(motor_pin_4, LOW);
      break;
      case 3:    //0001
```

```
      digitalWrite(motor_pin_1, LOW);
      digitalWrite(motor_pin_2, LOW);
      digitalWrite(motor_pin_3, LOW);
      digitalWrite(motor_pin_4, HIGH);
      break;
    }
  }
}

/*
   Function to find the Zero Position of the Motor
*/
void EasyStepper::findZero(int opticalSensorPin)
{
   pinMode(opticalSensorPin, INPUT);

   while (digitalRead(opticalSensorPin) != 0)
   {
      step(1);
   }
   step_number = 0;
   this->stepper_position = 0;
}

/*
   Function to move the stepper motor to a particular hour
*/
void EasyStepper::moveToHour(int hour)
{
   //Number of steps required (starting from the zero position)
   int numberOfSteps = -1;

   numberOfSteps = (hour/3)*50 +
calculateHourOffset(hour%3);

   // if(numberOfSteps != this->stepper_position){

      // step(-numberOfSteps);
   // }

   // else step(0);

   step(-numberOfSteps);
}

/*
   Function to move the stepper motor to a particular hour
   Accounts for the offset associated with the minute hand
*/
void EasyStepper::moveToHour(int hour, int minute)
{
   //Number of steps required (starting from the zero position)
   int numberOfSteps = -1;

   numberOfSteps = (hour/3)*50 +
calculateHourOffset(hour%3) +
calculateExtraMinuteOffset(minute);

   // if(numberOfSteps != this->stepper_position){

      // step(-numberOfSteps);
   // }

   // else step(0);
   step(-numberOfSteps);

   this->stepper_position = numberOfSteps;
}


/*
   Function to move the stepper motor to a particular hour
   Accounts for the offset associated with the minute hand
   in normal mode
*/
void EasyStepper::normalHour(int hour, int minute)
{
   //Number of steps required (starting from the zero position)
   int numberOfSteps = -1;

   numberOfSteps = (hour/3)*50 +
calculateHourOffset(hour%3) +
calculateExtraMinuteOffset(minute);

   if(numberOfSteps != this->stepper_position){

      step(-(numberOfSteps-this->stepper_position));
   }

   else step(0);
   // step(-numberOfSteps);

   this->stepper_position = numberOfSteps;
}


/*
   Function to calculate the hour offset
*/
int EasyStepper::calculateHourOffset(int hour)
{
   if(hour == 0)
   {
      return 0;
   }
   if(hour == 1)
   {
      return 17;
   }
   if(hour == 2)
   {
      return 34;
   }
   return -1;
}

/*
   Function to calculate the extra minute offset for the hour
hand
*/
```

```
int EasyStepper::calculateExtraMinuteOffset(int minute)
{
   if(minute/15 == 0)
   {
      return 0;
   }
   if(minute/15 == 1)
   {
      return 4;
   }
   if(minute/15 == 2)
   {
      return 8;
   }
   if(minute/15 == 3)
   {
      return 12;
   }
}

/*
   Function to move the stepper motor to a particular minute
*/
void EasyStepper::moveToMinute(int minute)
{
   //Number of steps required (starting from the zero position)
   int numberOfSteps = -1;

   numberOfSteps = (minute/3)*10 +
calculateMinuteOffset(minute%3);

   step(-numberOfSteps);

}

/*
   Function to move the stepper motor to a particular minute in
normal mode
*/
void EasyStepper::normalMinute(int minute)
{
   //Number of steps required (starting from the zero position)
   int numberOfSteps = -1;

   numberOfSteps = (minute/3)*10 +
calculateMinuteOffset(minute%3);

   if(numberOfSteps != this->stepper_position){

      step(-(numberOfSteps-this->stepper_position));
   }

   else step(0);
   //step(-numberOfSteps);

   this->stepper_position = numberOfSteps;
}


/*
   Function to calculate the minute offset
*/
int EasyStepper::calculateMinuteOffset(int minute)
{
   if(minute == 0)
   {
      return 0;
   }
   if(minute == 1)
   {
      return 3;
   }
   if(minute == 2)
   {
      return 6;
   }
   return -1;
}

/*
  version() returns the version of the library:
*/
int EasyStepper::version(void)
{
  return 1;
}
```

2) EasyStepper.h

```
/*
  EasyStepper.h - - Stepper library for Wiring/Arduino -
Version 0.1

  Modification for SDP (0.1) by Eric Moore

  Drives a unipolar or bipolar stepper motor using  2 wires or 4
wires

  When wiring multiple stepper motors to a microcontroller,
  you quickly run out of output pins, with each motor requiring
4 connections.

  By making use of the fact that at any time two of the four
motor
  coils are the inverse  of the other two, the number of
  control connections can be reduced from 4 to 2.

  A slightly modified circuit around a Darlington transistor
array or an L293 H-bridge
  connects to only 2 microcontroler pins, inverts the signals
received,
  and delivers the 4 (2 plus 2 inverted ones) output signals
required
  for driving a stepper motor.

  The sequence of control signals for 4 control wires is as
follows:

  Step C0 C1 C2 C3
     1  1  0  1  0
```

```
    2  0  1  1  0
    3  0  1  0  1
    4  1  0  0  1

  The sequence of controls signals for 2 control wires is as
follows
  (columns C1 and C2 from above):

  Step C0 C1
    1  0  1
    2  1  1
    3  1  0
    4  0  0

  The circuits can be found at
  http://www.arduino.cc/en/Tutorial/Stepper
*/

// ensure this library description is only included once
#ifndef EasyStepper_h
#define EasyStepper_h

// library interface description
class EasyStepper {
  public:
    // constructors:
    EasyStepper(int number_of_steps, int motor_pin_1, int
motor_pin_2);
    EasyStepper(int number_of_steps, int motor_pin_1, int
motor_pin_2, int motor_pin_3, int motor_pin_4);

    // speed setter method:
    void setSpeed(long whatSpeed);

    // mover method:
    void step(int number_of_steps);

    void findZero(int opticalSensorPin);
    void moveToHour(int hour);
    void moveToHour(int hour, int minute);
    void moveToMinute(int minute);
    void normalHour(int hour, int minute);
    void normalMinute(int minute);

    int version(void);

  private:
    void stepMotor(int this_step);

    int direction;      // Direction of rotation
    int speed;          // Speed in RPMs
    unsigned long step_delay;    // delay between steps, in ms,
based on speed
    int number_of_steps;      // total number of steps this motor
can take
    int pin_count;        // whether you're driving the motor with
2 or 4 pins
    int step_number;        // which step the motor is on
    int stepper_position;    // Number of steps that have gone by

    // motor pin numbers:
    int motor_pin_1;
    int motor_pin_2;
    int motor_pin_3;
    int motor_pin_4;

    long last_step_time;      // time stamp in ms of when the last
step was taken

    int calculateExtraMinuteOffset(int minute);
    int calculateMinuteOffset(int minute);
    int calculateHourOffset(int hour);

};

#endif
```

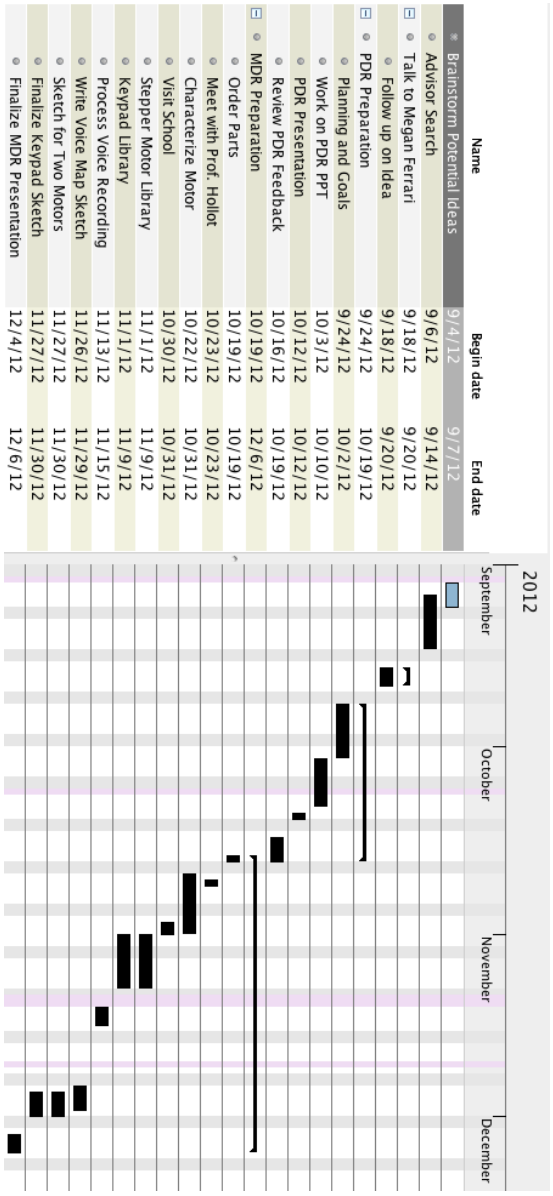
3) NormalMode.ino (Arduino Sketch)

```
#include <EasyStepper.h>
#include <Time.h>
#include <Wire.h>


const int stepsPerRevolution = 200;
int opticalSensorPinHour = 2;
int opticalSensorPinMinute = 3;

EasyStepper hours(stepsPerRevolution, 4,5,6,7);
EasyStepper minutes(stepsPerRevolution, 8,9,10,11);

void setup() {


  // set time
  setTime(20, 35, 0, 6, 12, 2012);

  // set the speed at 10 rpm:
  hours.setSpeed(10);
  minutes.setSpeed(10);

  //intialize pins as input
  pinMode(opticalSensorPinHour, INPUT);
  pinMode(opticalSensorPinMinute, INPUT);

  // initialize the serial port:
  minutes.findZero(opticalSensorPinMinute);
  hours.findZero(opticalSensorPinHour);
}

void loop() {
   minutes.normalMinute(minute());
   hours.normalHour(hourFormat12(),minute());

}
```

4) Sketch to play back a time stamp typed in via serial input

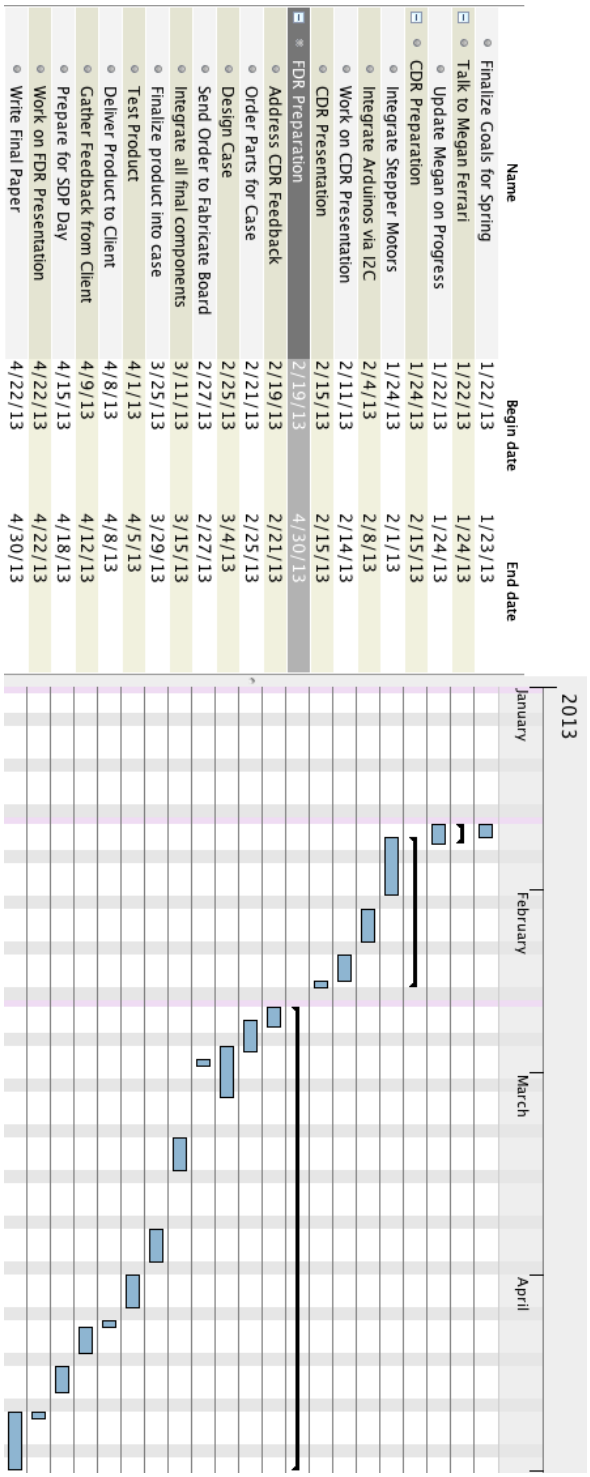
```
/*
 ClockAide - VoiceMap Demo
*/

#include <SPI.h>
#include <SdFat.h>
#include <SdFatUtil.h>
#include <SFEMP3Shield.h>
#include <Wire.h>

SFEMP3Shield MP3player;

byte result;

void setup()
{
  Serial.begin(9600);
  MP3player.begin();
}

void loop()
{
  if (Serial.available()>0)
  {
    int time = Serial.parseInt();
    int hour = time/100;
    int minute = time%100;
    if(isTimeValid(hour,minute))
    {
      Serial.print("Hour: ");
      Serial.println(hour);
      Serial.print("Minute: ");
      Serial.println(minute);
      speakTime(hour, minute);
    }
    else
    {
      Serial.println("Time is invalid! Try again :-)");
    }
  }
}

boolean isTimeValid(int hour, int minute)
{
  if(hour >= 1 && hour <= 12 && minute >= 0 && minute <
60)
    return true;
  return false;
}

void speakTime(int hour, int minute)
{
  if(minute != 0)
  {
    String hourFilename = "hrs" + String(hour) + ".mp3";
    String minuteFilename = "min" + String(minute) + ".mp3";
    char charHourFilename[30];
    char charMinuteFilename[30];

    hourFilename.toCharArray(charHourFilename, 20);
    minuteFilename.toCharArray(charMinuteFilename, 20);

    //MP3player.SetVolume(100,100);
    MP3player.playMP3(charHourFilename);
    delay(700);
    MP3player.stopTrack();
    //MP3player.SetVolume(100,100);
    MP3player.playMP3(charMinuteFilename);
  }
  else
  {
    String hourFilename = "hrs" + String(hour) + ".mp3";

    char charHourFilename[30];

    hourFilename.toCharArray(charHourFilename, 15);

    //MP3player.SetVolume(100,100);
    MP3player.playMP3(charHourFilename);
    delay(700);
    MP3player.stopTrack();
    //MP3player.SetVolume(100,100);
    MP3player.playMP3("oclock.mp3");
  }
}
```

5) Fall Gantt Chart



Name	Begin date	End date
Brainstorm Potential Ideas	9/4/12	9/7/12
Advisor Search	9/6/12	9/14/12
Talk to Megan Ferrari	9/18/12	9/20/12
Follow up on Idea	9/18/12	9/20/12
PDR Preparation	9/24/12	10/19/12
Planning and Goals	9/24/12	10/2/12
Work on PDR PPT	10/3/12	10/10/12
PDR Presentation	10/12/12	10/12/12
Review PDR Feedback	10/16/12	10/19/12
MDR Preparation	10/19/12	12/6/12
Order Parts	10/19/12	10/19/12
Meet with Prof. Hollot	10/23/12	10/23/12
Characterize Motor	10/22/12	10/31/12
Visit School	10/30/12	10/31/12
Stepper Motor Library	11/1/12	11/9/12
Keypad Library	11/1/12	11/9/12
Process Voice Recording	11/13/12	11/15/12
Write Voice Map Sketch	11/26/12	11/29/12
Sketch for Two Motors	11/27/12	11/30/12
Finalize Keypad Sketch	11/27/12	11/30/12
Finalize MDR Presentation	12/4/12	12/6/12

6) Spring Gantt Chart



Name	Begin date	End date
Finalize Goals for Spring	1/22/13	1/23/13
Talk to Megan Ferrari	1/22/13	1/24/13
Update Megan on Progress	1/22/13	1/24/13
CDR Preparation	1/24/13	2/15/13
Integrate Stepper Motors	1/24/13	2/1/13
Integrate Arduinos via I2C	2/4/13	2/8/13
Work on CDR Presentation	2/11/13	2/14/13
CDR Presentation	2/15/13	2/15/13
FDR Preparation	2/19/13	4/30/13
Address CDR Feedback	2/19/13	2/21/13
Order Parts for Case	2/21/13	2/25/13
Design Case	2/25/13	3/4/13
Send Order to Fabricate Board	2/27/13	2/27/13
Integrate all final components	3/11/13	3/15/13
Finalize product into case	3/25/13	3/29/13
Test Product	4/1/13	4/5/13
Deliver Product to Client	4/8/13	4/8/13
Gather Feedback from Client	4/9/13	4/12/13
Prepare for SDP Day	4/15/13	4/18/13
Work on FDR Presentation	4/22/13	4/22/13
Write Final Paper	4/22/13	4/30/13

VII. ACKNOWLEDGEMENTS

Megan Ferrari – Teacher at West Springfield Middle School

She is the direct beneficiary of this project. She reached out to the group at the beginning of the semester and explained what her needs were. Our group chose to work with her. She also invited us to see the children and get feedback on the device.

Team ClockAide faculty advisor – Professor T.B. Soules

Professor Soules met with our group weekly to discuss ideas, future plans and continuously helped us overcome hindrances in our design. We bounced different concepts with him and took his advice very seriously due to his extensive background and knowledge in Electrical and Computer Engineering.

Instructors / Course Coordinators – Professor Christopher Hollot and Professor Christopher Salthouse

These two professors gave us great insight and advice during one-on-one meetings with our group in lab, email correspondence and classroom lectures about different design principles, working productively in a group and prototyping to meet specifications.

Senior Design Project Evaluators - Professor Leonard and Professor Ciesielski

These two professors took time out of their busy schedules to watch and evaluate our progress during each stage of our design. Thus far, they gave fantastic and important information during our Preliminary Design Review (PDR) presentation that helped us reshape and re-evaluate different components of our project. Specifically, they informed us on what they expect for the Midyear Design Review (MDR) presentation.

SDP Technician - Francis Caron

Mr. Caron has been an invaluable resource to all the teams in enrolled in the Senior Design Project course. He has helped us order all the equipment for our project and set up our work station computers. He helped us to keep organized during this year long capstone from the initial idea phase to the final completed design.

REFERENCES

[1] G. Boulton-Lewis, L. Wiss, S. Mutch, "Analysis of Primary School Children's Abilities and Strategies for Reading and Recording Time from Analogue and Digital Clocks," Queensland University of Technology

[2] Microchip PIC16C5X Datasheet. EEPROM/ROM-Based 8-bit CMOS, pp. 4, 13, 37 Microchip Corporation. Copyright 2002.

[3] Sipex SP233ACP Datasheet. pp. 3. Copyright 2000.

[4] Atmel AVR ATMega328 Datasheet. 8-bit Microcontroller with 32Kbytes In-System Programmable Flash. Copyright 2005.

[5] Xiamen Ocular GDM1602K Datasheet. No date available.

[6] Roberto Guido, 2012-11-23. Arduino – ArduinoBoardUno. Arduino wiki. Available: http://www.arduino.cc/en/Main/arduinoBoardUno

[7] Scott Fitzgerald, 2012-10-20. Arduino – Reference | History. Arduino wiki. Available: http://arduino.cc/en/Reference/HomePage?action=diff

[8] Paul-Arthur Asselin, 2011-5-5. What's the password? Arduino + keypad. Arduino wiki. Available: http://bildr.org/?s=what%27s+the+password

[9] crocboy, 2010-3-8. Connecting an LCD to an Arduino. Instructables.com. Available: http://www.instructables.com/id/Connecting-an-LCD-to-the-Arduino/#step1

[10] Hacktronics, 2012-11-23. Arduino Character LCD Tutorial. Arduino wiki. Available: http://www.arduino.cc/en/Main/arduinoBoardUno

[11] Roberto Guido, 2012-11-23. Arduino – ArduinoBoardUno. Arduino wiki. Available: http://www.arduino.cc/en/Main/arduinoBoardUno

[12] Roberto Guido, 2012-11-23. Arduino – ArduinoBoardUno. Arduino wiki. Available: http://www.arduino.cc/en/Main/arduinoBoardUno

[13] Stepper Motor, 2012-11-01. Stepper motor - Wikipedia, the free encyclopedia. Available: http://en.wikipedia.org/wiki/Stepper_motor