Fall 12-1-2014

A Highly Adaptive and Cost Effective Second Generation Incubator (SGI) towards Educational, Research and Clinical Processes

Yassir H. Al-Sayagh
University of Nebraska-Lincoln, yassirsheek@hotmail.com

Follow this and additional works at: http://digitalcommons.unl.edu/biosysengdiss

Part of the Bioelectrical and Neuroengineering Commons, Engineering Education Commons, and the Other Biomedical Engineering and Bioengineering Commons


This Article is brought to you for free and open access by the Biological Systems Engineering at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Biological Systems Engineering--Dissertations, Theses, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
A Highly Adaptive and Cost Effective Second Generation Incubator (SGI) towards Educational, Research and Clinical Processes

by

Yassir H. Al-Sayagh

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree Masters of Science

Major: Agriculture and Biological System Engineering

Under the Supervision of Professor Huihui Xu

Lincoln, Nebraska

October, 2014
A Highly Adaptive and Cost Effective Second Generation Incubator (SGI) towards Educational, Research and Clinical Processes

Yassir Al-Sayagh, M.S.

University of Nebraska, 2014

Advisor: Huihui Xu

Today´s rising demand for more reliable and affordable alternatives to organ transplant has led to a growing market for in vitro tissue culture systems. The main objective of tissue engineering as a whole is to employ human tissue equivalents for commercial use. The state-of-the-art processes for producing these so called tissue models are still very expensive, difficult to produce and time-consuming.

To engineer tissues, in vitro, a three-dimensional support structure commonly termed scaffold is needed. Stem cells are then added to the scaffold. In order for tissue to materialize, the scaffold and the cells must be incubated (cultured) at a temperature of 37°C, 5.0% CO2, and 95% air concentration with relative humidity of 95-99%, for a tissue engineered constructs to be viable. This requires a sterile environment, in which the tissue to be assemble. This is commonly accomplished using a petri dish inside an incubator. However, the tissue constructed is not of clinical application quality. Therefore, bioreactors come into play by exhorting physiochemical stimuli to further enhance the tissue engineered, in essence mimicking in vivo conditions and thus procuring quality construct for clinical use.

Therefore, tissue engineering needs a paradigm shift in order to generate clinically useful products. The main objective of this line of research is to design, build and test
Second Generation Incubator (SGI) systems that can simulate in vivo conditions under standard culture frames and sustain cell life to produce a viable tissue engineered construct. The SGI apparatus offers many features such as; ease of use, customizability, compatibility, portability and low cost, which current commercially available incubators and bioreactors lack.

The SGI apparatus is similar to a Matryoshka doll structurally. SGI Housing unit encompasses three air sealed stackable units to maintain the cell environmental conditions. That is, a Cell/Tissue Housing (CTH) unit, Heating unit and Gas unit. In addition, the casing divides the SGI apparatus into three compartment. The top, middle and bottom compartments houses the instruments and devices, the SGI Housing unit and the electronics respectively.

A two week experiment was conducted using adult human mesenchymal stem cells in conjunction with the gelatin scaffolds to form a viable bone tissue engineered construct. The results showed great thermal, gas and humidity regulation and with no adverse signs of contamination. Osteogenetic markers showed lesser osteogenetic levels in that of the SGI apparatus.
Table of Contents

Abstract .............................................................................................................. Error! Bookmark not defined.
List of Tables: ................................................................. vi
List of Figures: ................................................................. vii
Chapter 1: .......................................................................................... 1
  Introduction to Tissue Engineering .............................................................. 1
    Stem Cells .................................................................................. 2
    Scaffolds .................................................................................. 5
    Growth factors: ........................................................................... 7
    Cells, Scaffolds and Signaling Systems to Tissue Formation: ....................... 8
  Tissue Culture .................................................................................. 9
SGIs vs Incubators vs Bioreactors: ................................................................. 10
Thesis Objective: .................................................................................. 13
Design Requirements: ........................................................................... 13
Chapter 2: .......................................................................................... 14
  Second Generation Incubator ................................................................... 14
    Goals of SGIs ............................................................................ 14
      A). SGI Control System: ............................................................. 16
      B). SGI Housing Unit: ................................................................. 28
      C). SGI Casing overall design: ...................................................... 31
    SGI Housing Unit Compartment: ...................................................... 35
      Devices and equipment Compartment: ........................................... 36
  Electronics Compartment: ................................................................. 37
    SGI coming to life: ........................................................................ 39
    Overall materials: .......................................................................... 42
  Expenses Summary: ........................................................................ 42
    Bill of Material: ........................................................................ 43
Chapter 3 Results: Validation of the SGI system ........................................ 48
  Temperature ................................................................................ 48
  CO2 ........................................................................................ 48
  Humidity: ................................................................................ 49
  Application of SGI using bone tissue engineering as a model system: .......... 51
List of Tables:

Table 1. Material 1 Expense Sheet ........................................................................................................... 43
Table 2. Material 2 Expense Sheet ........................................................................................................... 46
Table 3. Amplified genes, and specific primer ........................................................................................... 54
List of Figures:

Figure 1. Comparison of Modern Bio, Inc. and SGI features .................................................. 12
Figure 2. Sparkfun’s SHT15 humidity sensor. ........................................................................... 17
Figure 3. Heating Unit. .................................................................................................................. 19
Figure 4. CO2 System .................................................................................................................. 21
Figure 5. Media Dispersion Scheme ........................................................................................... 23
Figure 6. Power Supply Schematic ............................................................................................. 23
Figure 7. Schematic of SGI system 1 .......................................................................................... 25
Figure 8. Schematic of SGI system (2) ....................................................................................... 27
Figure 9. Add-on Circuit feature ................................................................................................ 27
Figure 10. Overall SGI Housing Unit Design ............................................................................. 30
Figure 11. 3D Animation of the SGI Housing Unit ................................................................. 31
Figure 12. SGI Casing Front and Side View .............................................................................. 32
Figure 13. SGI Rear View. .......................................................................................................... 32
Figure 14. SGI Top View ............................................................................................................ 33
Figure 15. SGI Isometrical View ............................................................................................... 34
Figure 16. SGI 3D View: Front, isometric and bird eye views .................................................. 35
Figure 17. The Housing Unit Compartment .............................................................................. 35
Figure 18. 3D View of the Device and Equipment compartment .............................................. 36
Figure 19. 3D view of the front panel ......................................................................................... 37
Figure 20. 3D view of the main power unit ............................................................................... 38
Figure 21. SGI front and back view ............................................................................................ 39
Figure 22. SGI Left/Right side views. ......................................................................................... 39
Figure 23. SGI Electronic and Equipment compartments .......................................................... 40
Figure 24. SGI's Housing Unit ................................................................................................... 41
Figure 25. Temperature Fluctuations Graph .............................................................................. 48
Figure 26. CO2 Fluctuations Graph ............................................................................................ 49
Figure 27. Humidity, Temperature vs Time Graphical Analysis ............................................... 50
Figure 28. Human Mesenchymal Stem Cells Morphology ....................................................... 55
Figure 29. Cell Morphology and Attachment .......................................................................... 55
Figure 30. RT-PCR Analysis of ALP and BSP. ......................................................................... 56
Chapter 1: 

Introduction to Tissue Engineering

The field of tissue engineering aims to deliver biological replacements to treat diseased or damaged tissues. A major pursuit of tissue engineering is the in vitro growth of tissue replacements, such as skin, vessels, heart valves, myocardium, ligaments, tendons, cartilage and bone. However, engineered tissues have failed to be implanted on any regular basis. Despite a large and increasing research effort aimed at growing functional tissues in vitro, including experimentation with a vast array of cell sources, tissue scaffolds and applied stimulation (physicochemical, and/or electrical), with few exceptions they lack the required functionality and affordability tissue engineered solutions that compete with the current influx of patients on the transplant waiting list.

Many difficulties arise with direct transplantation due to insufficient donor organs, pathogen transmission and rejection of the donor organ [1 - 3]. As a result, patients can be waiting for a donor organ for years, and when they receive one in time, they need to take immunosuppressive medication for the rest of their lives and risk the need of a replacement organ within days to years after the surgery. This is where tissue engineering has the upper hand; an autogenic tissue engineering transplant (using patient’s own cells) would address most limitations of direct transplantation and avoid difficulties concerning rejection and pathogen transmission. Additionally, there would be no dependency on donors. Therefore, constructing a tissue engineered replacement in vitro is considered an excellent alternative to direct transplantation of donor organs [1-4]. In essences, TE attempts to mimic the function of natural tissue constructs. Therefore, the
development of such a construct requires a careful selection of three key materials: 1) Stem Cells, 2) Scaffolds, and 3) Growth Factors.

**Stem Cells**

Stem cells over the past few years has received tremendous interest from scientists, because of its potential to provide cures or new treatments for numerous incurable diseases and injuries. The field of stem cell research is still in its early stages. A stem cell, by definition, is an undifferentiated (unspecialized) cell that can produce both new stem cells (self-renewal), and cells that commit to a pathway leading to differentiation. Differentiation is a process where an unspecialized cell changes to acquire features typical of cells of one of the organs in the body. Stem cells remain uncommitted and in a slowly proliferating state in vivo, until they receive a signal to develop into a specialized cell [5].

This differentiation signal triggers altered expression of genes involved in cell division and differentiation. Depending on the signal provided, the cells can undergo divisions to maintain or expand the stem cell pool and to produce differentiated progeny. Some stem cells can produce a variety of differentiated cells [6], while other stem cells only generate a few, or one type differentiated cells.

**Source and type of stem cells:**

Most adult organs in the body contain stem cells, which are referred to as adult stem cells (ASC). These cells have a more restricted differentiation potential compared to the embryonic stem cells (ESC), and lack the capability of the ESC of unlimited proliferation in culture. Stem cells with limited differentiation potential are often called multipotent stem cells. It was thought that ASC could only differentiate into the cell types
of the organ/tissue from which they originated. More recent studies have described differentiation of ASC into mature phenotypes that are different from their organ/tissue of origin [7, 8].

Throughout life, billions of dead cells are replaced by new cells every day in order to maintain tissue homeostasis. The main function of the ASC is to regenerate and repair the tissues in which they reside. Normally, ASCs produce progenitor cells, which proliferate before they differentiate into the mature cells that are being replaced. ASCs have their own specialized environment called niche [9]. For most stem cells the niche is a complex microenvironment composed of extracellular matrix, differentiated cells, stem cells and their progeny, and factors secreted by the cells. The fate of the stem cells is controlled by this niche [10].

There are very few ASCs exist in each tissue, thereby making it a challenge to isolate them in the laboratory. Cells contain proteins called antigens, or surface markers, in their cell membranes. Each cell type has a different set of surface markers, protruding from the cell’s surface. By taking advantage of the uniqueness of these markers, different cells can be isolated and identified by modern techniques of cell biology. [11, 12]

Mesenchymal stem cells (MSCs), for example, are adult stem cells that are usually isolated from bone marrow [13-15] or adipose tissue [16, 17] but can also be isolated from several other tissues such as skeletal muscle [18], trabecular bone [19], dermis [20], periosteum [21], teeth [22], synovial membrane [23], amniotic fluid [24], placenta [25] and umbilical cord blood [26]. In contrast, hematopoietic stem cells give rise to blood cells only, while neural stem cells form only cells in the nervous system. Progenitor cells, on the other hand, are early descendants of stem cells that can
differentiate to form one or more kinds of cells, but cannot divide and reproduce indefinitely. Because of their ease of isolation and extensive differentiation potential, as well as their high in vitro expansion potential, MSCs have been well characterized, and they are believed to have a great potential for cell therapy and tissue engineering.

**Stem Cell Source**

The cell source has an enormous influence on the success of tissue engineering. Cells applicable to tissue engineering may be classified into autologous (patient's own), allogeneic (human other than patient) and xenogeneic (animal origin) [27]. Autologous cells are the most suitable for tissue engineering due to the fact that initiation of an immune response of the implanted tissue is minuscule. Whereas allogeneic and xenogeneic cells are immunogenic and will need an immunosuppressive therapy when a new tissue is engineered from these heterogeneous cells. A problem associated with autologous cells is the difficulty in harvesting a sufficient amount of cells, especially when a patient is aged or has severely been diseased [28]. For instance, it is extremely difficult to harvest cardiac cells from a patient suffering from myocardial infarction (heart attack). If the amount of harvested cells is not sufficient enough for clinical treatment, the cells should be expanded by cell culture.

Due to lack of organ donors and limitations because of immunological rejections and physical size, stem cells may provide ways for obtaining tissues and organs for transplantation. Adult stem cells are the body’s repair mechanism, called into action by normal wear and tear on the body and when serious damage or disease ensues. The patient’s own cells could be seeded on a bio-degradable scaffold that could allow the formation of a particular tissue.
**Scaffolds**

Most mammalian cells are anchorage-dependent and require the presence of a supporting material for growth [29, 30]. Cells are the basic building blocks of tissue, and tissues make up the functioning organs in the body. Groups of cells aggregate and start making their own support structures, called extra-cellular matrix (ECM). The ECM is a mesh-like substance within the extracellular space that supports cell attachment and promotes cell proliferation. [31, 32] This ECM also plays a huge role in coordinating cellular behavior that creates the physiological and function of tissues and organs.

To development an engineered tissue similar to that of the natural tissue to be replaced, an artificial ECM mimic must be designed, such a mimic is called a scaffold. A scaffold is a 3-D construct that serves as temporary support for isolated cells to grow into new tissue. Scaffolds sole purpose is to direct morphogenesis in vitro and to maintain the structure and function of the construct as it is integrated with the host tissues after implantation. The normal function of most stem cells and tissues depends on the interaction of neighboring cells and with the extracellular matrix. Therefore, the design of the scaffold determines the functionality of the construct to a higher extent. Although the final requirements depend on the specific purpose of the scaffold, several general characteristics and requirements need to be considered for all scaffold designs [33 - 36].

The scaffold must:

- be biocompatible, that is, the scaffold should provoke an appropriate biological response in a specific application and prevent any adverse response of the surrounding tissue. [37, 38]
- be biodegradable, should the scaffold material degrade, the scaffold must be non-toxic and without interfering with the functionality of the surrounding tissue. [39]

- promote cell attachment, spreading and proliferation; vital for the regulation of cell growth and differentiation. [40]

- be comparable to that of in vivo tissue, in terms of mechanical strength. A scaffold requires more flexibility or rigidity depending on the application in example cardiovascular versus bone prostheses. [41]

- have good transport properties, ensure sufficient nutrients transport and removal of waste products to and from the cells, respectively. The scaffold should also be highly porous (meanwhile maintaining sufficient mechanical strength). [33, 40 - 42]

- have suitable surface characteristics, for instance surface topography into the scaffold improves tissue organization leading to increased tissue function, supports cell attachment and promotes cell proliferation. [43 - 46]

**Fabrication:**

The scaffolds used in tissue engineering are designed to mimic the structural, chemical and mechanical properties of the body’s natural ECM. Tissue engineering scaffolds are fabricated using methods that attempt to reproduce the structural features of the ECM specifically. Several manufacturing methods exist to produce scaffolds that mimic the ECM including fiber bonding [47 18], phase separation [48,49 18,19], solvent casting [47 ,50], particulate leaching [47,50], extrusion [47,51,52], and freeze drying
Other common techniques include the formation of hydrogels [53] and electrospinning of polymer nanofibers [54].

Tissue engineering scaffolds must be designed with the target organ in mind in order to develop a suitable tissue replacement. Structural features of the scaffold are extremely important for regulating tissue growth and this will change between different tissues. For example, a scaffold that is appropriate for engineered bone may not be suitable for a heart valve replacement.

**Growth factors:**

Biological tissues basically consist of cells, extracellular matrix (ECM) and signaling systems (growth factors) [55]. The cells are the core of the tissue, however, cannot function in the absence of signaling systems and/or of the ECM. The signaling system consists of genes that secrete transcriptional products when differentially activated, and urges cues for tissue formation and differentiation [55].

Cell-cell and cell-ECM interactions are coordinated by several families of membrane-spanning proteins known as adhesion molecules. These are fundamental to cell adhesion, helping to define 3-dimensional cellular organization and also directly participating in cell signaling and controlling cell recruitment, growth, differentiation, immune recognition. Growth factors can be combined within the scaffold - based approach to guide cell behavior by triggering specific reactions through pathway activation [56-58].

Growth factors are proteins involved in the cellular communication system which modulate cell activity in a concentration and time dependent fashion [15, 16] [59-60]. Hundreds of growth factors have been identified that inhibit or stimulate proliferation,
differentiation, migration, or gene expression of various cell types. With regard to cartilage, several growth factors have regulatory effects on cartilage metabolism among which the most investigated are transforming growth factor-β1 (TGF-β1), bone morphogenetic proteins (BMPs) and insulin growth factor-1 (IGF-1) [61-64]. These molecules play a role in the maintenance of the chondrogenic phenotype, the proliferation of chondrocytes. Accordingly, they are promising candidates to be associated with scaffolds to support, induce or enhance the growth and differentiation of different cell types towards the chondrogenic lineage and to orchestrate the cartilage repair.

To exert their action, each growth factor requires different dosages and length of exposure to the cells. Consequently, they can potentially induce undesired side effects when presented in wrong fashion and if present at systemic levels [65, 66]. Therefore, scaffolds associated with growth factors should provide the means to precisely control their doses and supplementation rate at a local level. In addition, growth factors are labile and have a short half-life in the body. These different characteristics and requirements logically lead to the development of controlled release approaches for the delivery of growth factors from scaffolds. By offering a sustained release of the growth factor to the site of implantation, one can expect to induce a longer and more stable tissue response. Multiple growth factors should be released independently from the scaffold to orchestrate the repair.

**Cells, Scaffolds and Signaling Systems to Tissue Formation:**
To conceptualize, tissue formation begins with the process of building a scaffold from sources ranging from proteins to synthetic polymers, each with unique mechanical properties. Once scaffolds are created, cells with or without growth factors can be
introduced. In some cases, the cells, scaffolds, and growth factors are all mixed together at once and in other cases are not always simultaneously used, but often in combination to regenerate or replace tissue [67-69]. Each type of tissue requires distinct conditions and therefore, demands the understanding of the specific natural biological environment in vivo to allow optimization of culturing in vitro. If the culturing environment is similar to that observed within the in vivo state, a tissue develops.

**Tissue Culture**

To maintain cell viability in vitro, more than sterility is necessary. The proper environment, mimicking that of the body, must be simulated. Therefore, temperature must be kept at 37°C, humidity at approximately 95-99% and a CO2 level of 5% for mammalian cells [70]. In a laboratory setting, this is simulated using a few different devices. The most common device used is an incubator. An incubator is a smaller cabinet that provides the proper temperature and CO2 level for culturing tissues/cells in media. Humidity isn’t often controlled explicitly since cells are typically in a flask suspended in culturing media, thus creating its own humid environment. Sterility in these incubators can be compromised as there are no active measures for maintaining sterility, therefore anything that is placed in an incubator needs to be handled under sterile conditions. Bioreactors are also often needed to provide any further stimuli to properly cultivate cells and tissues. Bioreactors are often custom devices that provide the necessary stimuli for their associated tissue. The stimuli varies based on the type of tissue, however typically compression, shear stress, repetitive elongation or electrical stimuli [71-80] are often applied. They are often used in combination with an incubator as the bioreactors don’t often control temperature, CO2 or humidity, and focus primarily on providing mechanical stimulation.
**Bioreactors for tissue engineering:**

Bioreactors should be designed and fabricated following specifications that differ from tissue to tissue. Tissue culture parameters such as temperature, pH, biochemical gradients, and mechanical stimulation must be continuously controlled during the maturation period because bioreactors should provide an in vitro environment mimicking the in vivo conditions [81, 82]. Thereby, the design of an appropriate bioreactor for a specific tissue is important, but also difficult in that bioreactors provide all the environment control and regulatory factors necessary for cell/tissue culture. Some research groups consider integrating multi stimuli in the culture process. The most common method combines the perfusion and mechanical stimuli together in one bioreactor. For example, a continuous pulsatile perfusion system is integrated with mechanical stimuli for vascular tissue [83, 84]; in other cases the perfusion system and static/dynamic compression loading are applied to cartilage tissue [85], and in another study, the perfusion of fluids to induced shear stress have been applied in bone tissue engineering as well [86]. Other groups considered integrating more than one external stimulus into the system. For instance, Feng et al. [87] developed a device to provide both electrical stimulation and dynamic tensile force during culture process.

The above techniques have a number of shortcomings and limit their applicability for in vitro applications. First of all, the relationships between external stimuli and cell reactions are still unclear. Secondly, using most of the above bioreactors to make tissue constructs requires a complex process beyond the scope of this thesis and the commercially available bioreactors and CO2 incubators. These specifically designed bioreactor systems make the size and quality of tissue patch variable and the culture
process difficult to operate. Industrial development has been hindered by difficulties in devising cost-efficient processes and guaranteeing product viability. In addition, applying external stimulation during the culture process increases the risk of contamination, especially for long term in vitro culture, due to the physical connection and the complicated assembly procedure. Therefore, this thesis will focus primarily on developing a Second Generation Incubator (SGI) apparatus for educational, research and clinical applications with enhanced features illustrated in Figure 1.
<table>
<thead>
<tr>
<th>Tissue Culturing Systems</th>
<th>Bioreactor</th>
<th>Incubator</th>
<th>SGI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physiochemical Stimulus</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Maintain adequate conditions for cell life</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Preserve aseptic conditions.</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Sterilizable.</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Highly customizable</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Compatible</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Portable/Mobile</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Low Cost</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>All in One Equipment</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Figure 1. Comparison of Modern Bioreactors, Incubators, and Second Generation Incubator characteristics.
**Thesis Objective:**

The main objective of the thesis is to design, build and test a tissue culture apparatus that can simulate in vivo conditions under standard culture frames and sustain cell life to produce a viable tissue engineered constructs. Although the SGI apparatus is initially designed for educational purposes, the apparatus has the potential to be applied within research and the clinic.

**Design Requirements:**

The SGI shall adhere to conditions required for cell life in terms of environment, sterility, ease of use and at a low cost. These conditions are pursued through the following sub-conditions, thus SGIs must:

- **✓** Maintain adequate conditions for cell life.
  - o Temperature (37° C (+/-0.5° C))
  - o Gas atmosphere (5% CO₂, 95% air)
  - o Relative humidity (95-99%).
- **✓** Preserve aseptic conditions.
- **✓** Sterilizable.
- **✓** Ease of use.
- **✓** Highly customizable.
- **✓** Compatible.
- **✓** Portable.
- **✓** Low Cost.
Chapter 2:

Second Generation Incubator

Goals of SGIs

In order to develop a device that will aid in the generation of tissue engineering constructs, the “Second Generation Incubator” (SGI) will have the following requirements. It will have to provide the proper environment to maintain cell viability and promote proper cellular activity. This requires SGI workspace to actively maintain sterility of the environment. Bacterial and fungal contamination can kill off all the cells used in any sort of tissue engineering, therefore providing an aseptic environment at minimum is necessary for creating any tissue construct during cultivation. In addition to maintaining sterility, the proper temperature, humidity, and CO2 concentration must be able to be controlled. As described previously, a temperature of 37°C, humidity of 95-99%, and a CO2 concentration of 5% are necessary for mammalian cells. Beyond providing the proper environment for cells, the work space must also provide the proper hardware for supporting such requirements. The design of SGI apparatus will incorporate the support hardware necessary for maintaining tissue/cell hemostasis. By achieving these goals, the SGI apparatus should provide an optimized platform for cultivating tissue engineered constructs.

The process begins with the establishment design requirements previously mentioned. Sterility aspect were considered at each phase. To recap, the main objective of the thesis is to design, build and test SGI culturing apparatus that can simulate in vivo conditions under standard culture frames and sustain cell life to produce a viable tissue engineered construct. Additionally, the volume must be easy to use, resterilizable, and
expenses shall be minimized. To accomplish these requirements, the SGI apparatus will be broken down into three domains:

A. SGI Control System:
   - Humidity Control
   - Thermal regulation and Control
   - CO2 Control
   - Aseptic Environment
   - Media Dispenser
   - Electronics
   - Circuit Customizability

B. SGI Housing Unit:
   - Sterility Derived Design
   - Cell/Tissue Housing Unit
   - Heating Unit
   - Gas Unit

C. SGI Casing:
   - SGI Housing Unit Compartment
   - Devices and equipment Compartment
   - Electronics Compartment
A). SGI Control System:  
**Humidity Control:**
Maintaining appropriate humidity conditions minimizes media evaporation. Evaporation may result in changes in the media's pH; pH changes may be detrimental to cell viability [88]. Since the volume is heated, some evaporation is anticipated; therefore, adequate humidity maintenance aims to prevent excess evaporation. To ensure that appropriate humidity is maintained, several options were considered including a humidifying water bath, a steam or ultrasonic humidifier, humidity packets, and saturated sponges. Preliminary tests helped to determine that, if anything, only a minimal water supply was needed to maintain adequate humidity to prevent adverse changes in pH; therefore, steam and ultrasonic humidifiers were eliminated. Both of these would lead to an increasingly complex design and increased contamination risks. Sponges were found to assist in pH regulation in a similar way to a water bath; however, evaporation with no humidification source was found to be low. Hence, neither of these options was further developed, and humidification was not considered to be necessary for this iteration of the sterile SGI design.

For certain projects that may be of importance to control humidity, SGI will incorporate a humidity sensor. A humidity sensor needs to be able to communicate with the microcontroller in a compatible manner (Serial or I2C), be able to read 0-100% humidity, be relatively accurate (+/- 1%), be diffusive read type, and not need a reference to sense the humidity. Low cost, low voltage and a large amount of reference materials for interfacing would be highly preferred. From these requirements, Sparkfun’s SHT15 humidity sensor (figure 2) was used. It is inexpensive, runs off 3.3V, making it low voltage, communicates two-wire digital interface, operates between 0 and 60 °C, and has
RH of ±2% accuracy. The board it’s mounted on and has the dimensions of 18 mm wide and 18 mm long.

![Sparkfun’s SHT15 humidity sensor.](image)

**Thermal regulation and control:**
Thermal regulation is important, because even slight temperature deviations can have severe consequences on the cell. Temperatures significantly above 37° C may denature proteins required for cell function, and compromise the integrity of the thermo-sensitive microtubules, which are crucial for cell division [88, 89]. This damage delays cell division, and can lead to an increase in apoptosis (cell death), which would be detrimental to the engineered tissue construct [89]. Lower temperatures may deter cell growth and division, potentially causing the cells to ball up and die.

To develop a temperature control environment, a feedback system capable of heating as well as a method of sensing the temperature thus providing proper control. An electronic method of heating will be necessary for it to be controlled via a microcontroller. The temperature must also be electronically sensed in order to interface with the control scheme.
Silicon rubber heater was used as a heating element. This heating element should be able to be easily incorporated into the design due to their availability in terms of variable size, voltage requirements and wattage output. The heater runs off at 12 V, and operate over the range of 0°C – 80°C, with 1 W/cm². For this project, a four 4”x4”, 12 V, 16 Watt heaters were selected. The temperature sensors should run off 5V, be accurate to at most ± 0.5 °C, read temperatures ranging from -55°C – 125°C, and be easily interfaced with the microcontroller. To accommodate these requirements, Sparkfun’s DS18B20 temperature sensors were selected. They run off 5V and provide a digital signal calibrated by the manufacturer. They do not require any amplification or calibration making them easily interfaced within the SGI apparatus and do not require any further equipment, in addition are waterproof. The sensors have an accuracy and resolution of ±0.5°C and operate between -55-125°C.

For SGI to be viable for tissue culture applications, a non-intrusive protocol must be employed. There for, the heating chamber unit must be capable of transferring heat efficiently and maintain thermal regulation. Different materials have been investigated to accomplish the role of low cost, non-toxic, reusable heating material that has great heat transfer characteristics. Aqua Pearls hot/cool packs were optimal for SGI applications as observed in Figure 3.
Figure 3. Heating Unit.
The Heating Unit contains three aspects. (a) Temperature Sensor, (b) Heating aqua-gel medium. (c) Heating Source.

**CO2 Control**
To provide control of the concentration of CO2, a method of infusing the tissue engineered construct with CO2 evenly and monitoring the concentration is necessary. The air flow from the surrounding environment can be taken advantage of by providing 95% air to the SGI Gas Unit. Since there is only about 0.039% CO2 in the air, in order to maintain 7.2-7.4 pH requirements for mammalian cells to survive, a 5% CO2 must be present. Using a mini Fluval 20g-CO2 Disposable Cartridges tank and an electronic solenoid to control the incremental injections of gas will provide the necessary infusion of CO2 required. The solenoid will have to be able to be controlled electronically, preferably from a 24VDC source, be of a normally closed design, and be able to accommodate at least 0 to 50 psi of CO2. A regulator to control the pressure released from CO2 tank will also be required. It must be calibrated for CO2, and be able to
regulate the pressure from 0-50 psi. The solenoid chosen was a Redhat one-way valve, of normally closed design, with an electronic solenoid that operates using ±12VDC. The inlet and outlet have 1/8” NPT female fittings and 1/32” orifice (depends on the size of the SGI tissue chamber). The Ingersoll Rang regulator chosen is capable of regulating up to 0-250 psi, which greatly exceeds its intended use. A silicon air tubes compatible with both the regulator outlet fitting and inlet of the solenoid valve was purchased locally from Thermal Scientific. Brass fittings rated to withstand up to 100 psi of pressure were used to mount and connect the solenoid to the SGI gas chamber and channel the air flow to the appropriate location.

An electronic CO2 sensor is required to provide proper control over the concentration of CO2 in the tissue chamber of the SGI as exemplified in figure 4. It must be a diffusive sensor type, not require any reference gas, be able to read a range of concentrations from 0-10% at minimum, easily interface with a microcontroller, and be affordable. After a cursory web search, there was only one affordable option that met all the design requirements. The SenseAir (K-33) CO2 sensor reads a range of 0-30% CO2 through a diffusive sensor, has an accuracy of ±100 ppm, runs off of 5-14 VDC, maximum of 250mA, and communicates via a serial port, I2C, digital and analog. It also has a warm up time of less than 1 minute and can take measurements every one second. All other comparable options were far outside the acceptable price range, exceeding $75.
Aseptic Environment

SGI is designed to prevent contamination through the use of air filtration, using the proper HEPA filters to evenly distribute the air flow and filter any particulates. The HEPA filters must filter out anything larger than 0.2 microns to provide the aseptic environment. Because submicron particle contamination can significantly jeopardize the tissue engineered construct. Thus the use of high efficiency submicron air filters have become a standard "current good manufacturing practice" allowing for more productive and economical processing. To accommodate GMP of air/CO2 filtration, two one inch inline radial 0.2 micron HEPA filter was chosen. In addition, the filter can accommodate humidity up to 99.99%.

Figure 4. CO2 System

The entire CO2/air control system. (a) CO2 tank (b) Mini cartilage piercer with on/off tuning. (c) Air pressure regulator. (d) Solenoid Valve (e) Y connector that combines the CO2 with air. (f) One-way air flow (g). 0.2 Micron HEPA filter. (h) CO2 sensor (i) Controlled regulation of 5% CO2 & 95% air.
**Media Dispenser:**

A culture media dispenser utilizes peristaltic pump to delivering small fixed volumes of growth medium liquid to and from tissue constructs and serves as a mass transport unit. What makes the peristaltic pump so unique is that when operated, no biological or chemical contamination can be introduced. It is internally sealed from the environment and designed for easy of cleaning and sterilization before use. Two peristaltic pumps are utilized in SGI system and serves as media exchange scheme. They are connected to the microcontroller and controllable via two switches.

Peristaltic pumps have so far been the industry standard for dispensing volumes of laboratory media. They are capable of dispensing fluid continuously from a reservoir into any number of receptacles, can be used for coarse, thick, or gritty fluids. Unlike many of the other dispenser types, the only contact the pumped fluid has during delivery is with the tubing in which it is being transported. There is never any contact with an internal piston, plunger, or diaphragm, which ensures the highest degrees of sterility and prevents almost any chemical cross contamination from the outside environment.

This peristaltic pump is purchased from adafruit industries, runs on 12 VDC, with 300 mA maximum current requirement. Peristaltic pump is composed of a DC motor and a pump head with three rollers pattern. As the motor turns, the roller presses on the tube pushes the fluid though. The flow rate adjusted between 10-100 ml and controlled via PWM signals form the microcontroller.
Figure 5. Media Dispersion Scheme

(a) Peristaltic pump made up of a motor, roller head and comes with silicon tube (b). Motor (c) Three roller head, pushes fluid through. (d) The two peristaltic pumps are controlled via two switches to allow media exchange.

**Electronics**

To provide power for the SGI control system, a Logisys ATX power supply unit was selected. The ATX power supply provides all the voltage and current requirements such as: 3, ± 5, ±12 voltage with rated amperage of 10 (Figure 6).

Figure 6. Power Supply Schematic.

SGI apparatus Main power supply electric circuit diagram.
The SGI requires an onboard microcontroller in order to control all the devices and sensors in order to maintain the proper environment to maintain cell viability. Based on the previously selected sensors, the microcontroller must have multiple analog to digital converter pins, serial ports, an I2C communication bus, a large number of digital input and output pins, as well as a decent amount of memory. It also must be powered using 12 VDC and have onboard regulation for 5 V output. In retrospect, be easily programmable, user friendly and at low cost. Based off these requirements, an Atmega644p AVR was selected. This microcontroller is part of the Arduino family. They are famous for ease of installation, programmability, and compatible with most sensors. Atmega644p runs off 7-12 VDC, has internal regulators to output 3.3 and 5 VDC, has 32 programmable I/O lines, 24 digital pins, 8 analog pins, 2 serial ports, one I2C port, 64KB of flash memory and a clock speed of 16MHz. These capabilities will provide the necessary control over all the necessary electronics for the device.

The sensor input as well as necessary user input must be able to be displayed electronically. Therefore, an LCD display is necessary. The LCD display must, at minimum, be able to display four rows of sixteen characters and have a built in LED for illumination. A monochrome sixteen by four character LCD display was chosen to match these requirements as well as the desired aesthetic. The display requires a 5 V source, the use of one serial port, and six other digital I/O pins.
The user must be able to control when each system is implemented. Therefore, a system of user input is required. Toggle switches were chosen to provide on/off control for each system. This requires six total toggle switches, three of which are used for system reset, flow of media in/out. The remaining three are utilized for the menu option, which provides the user with controlling the set temperature and CO2 parameters which varies based on tissue requirements. The three menu buttons allocated for Enter, Up and Down selection. Another higher-rated toggles will be used to provide an overall on/off for the entire system. To indicate to the user that a system in on and functioning, LEDs will be incorporated into the display. Two millimeter green LEDs were chosen to indicate flow of media. Another red LED was used as a power on indicator (Figure 7)

Figure 7. Schematic of SGI system 1

Schematic of the microcontroller, LEDs, heater, CO2 sensor, Temperature and USB-PC communication.
In order to provide electronic control via the microcontroller, MOSFETs and relays are required. MOSFETs are commonly used to provide on/off control via a microcontroller, as they can be operated using a 5VDC signal with low amperage, which can be easily provided by any of the digital output pins of a microcontroller. Relays are used when controlling higher voltages or currents is necessary, or when a signal needs to be switched from one line to another. Controlling a relay via a microcontroller also normally requires a MOSFET, because microcontrollers typically cannot produce enough current to excite the induction coil that operates the relay. Therefore using a MOSFET to control an independent source is necessary for controlling the relay. The MOSFETs must be N-channel MOSFETs and have a gate voltage of 5VDC and be able to control at least 12VDC. The relays will need to be rated for at least 12 VDC and 5 Amps with at most a 12VDC coil. The MOSFETs selected were purchased from Sparkfun and were rated to control 60VDC and 10A, with a gate voltage of 5VDC. One of the relays provides control of the CO2 injection, while the other is for controlling the temperature. Diodes are necessary to protect the circuit from back EMF generated around the motors being used and the relay’s coil. Resistors of various ohms will also be necessary for many different areas of the circuitry (Figure 8).
Figure 8. Schematic of SGI system (2)

Schematic of solenoid, peristaltic pumps, LCD screen.

**Circuit Customizability:**

SGI adds features to their circuit board by enhancing its setup to give the system maximum customizability for future application add-ons. This innovative system provides a wide range of functions that allow users to easily integrate devices, sensors and any electronics into the design. In effort to enhance circuit customizability, additional circuit board space will be provided either internally hardwired or as a clip-on shields boards. Such shields can be plugged on top of the PCB.

Figure 9. Add-on Circuit feature.
extending its capabilities (Figure 9). Many different shields can be easily mounted, electronically wired and applied to design.

Once the add-on feature has been added, it can be programmed via pc software using Arduino IDE. Arduino IDE allows users to add, edit or delete programming code (ANSI C) with a click of a button. To protect the SGI programming, an internal switch must be deactivated in order to allow program editing.

If additional I2C, serial, digital and analog pins are required due to the complexity of the add-on feature, the core operating system can be upgraded. SGI uses atmega644P a AVR microcontroller (40-pin) but can be upgraded to Atmega2560 which consist of has 54 digital pins, 16 analog pins, 4 serial ports, one I2C port, 256KB of flash memory and a clock speed of 16MHz. It is 100% compatible with the Atmega644P, no code modifications necessary except for pin configuration.

**B. SGI Housing Unit:**

**Sterility Derived Design:**

Effective sterilization processes reduces bio-burden, contamination, and increase the cell chances for survival. By minimizing airborne pathogens and other bioburden, the environment within will be more suitable to maintaining viability and encouraging cell growth. Cleaning is the removal of contaminants from a device to the extent necessary for further processing or use. Implementation of the ISO 17664 protocol, which is imposed on medical devices to ensure that devices branded are sterile and suitably decontaminated for use in human medical procedures.

Various sterilization methods can be used, including autoclave, UV/gamma ray, and ethylene oxide sterilization. Autoclaving is a common steam sterilization method that
does not involve harsh chemicals. In order for a device to be autoclavable it must be capable of withstanding temperatures of 250° to 375° F (121°-190° C). Ethylene Oxide (Eta) sterilization does not have the temperature constraints of autoclaving; but it has potential for hazardous residues, which can decrease the efficacy of this sterilization method since contact with these residues may have an adverse effect on the tissue [90]. Gamma Ray and UV sterilization are other methods, which can be coupled with Eta sterilization to increase efficacy. Sterility is necessary for any component that comes into direct contact with living tissue; aseptic standards are applicable for components that do not come into direct contact with the tissue, where contamination is less of a concern. Striving for complete sterility will reduce the risk of contamination, therefore increasing chances for cell viability.

**SGI Housing Unit Design:**

Based on the above criteria, the SGI housing is composed of three sub-units, the Cell/Tissue Housing (CTH) unit, Heating unit and Gas unit. The Cell/Tissue Housing (CTH) unit, like the name implies, houses cells/tissue constructs and maintains sterility, pH, CO2, humidity and Temperature. Due to interfacing devices, especially electronic or other sensitive electronics, must be reasonably protected from the humidity [91]. Therefore, separate units for the heating and gas compartments have been developed, that is, the Heating unit and The Gas unit (Figure 10).

An air tight seal, gaskets-like scheme was needed for all the contact surfaces between the sub-units. This prohibits any airborne pathogens and leaks that may compromise sterility and therefore cause cells/tissue to be contaminated. Thin gasket rubber mats were used due their superior gas sealant, nontoxicity and simplicity of
trimming different sizes/shapes property. With this design, the SGI apparatus can have a regulated environment capable of maintaining cell viability for days if not weeks.

Figure 10. Overall SGI Housing Unit Design.
**C. SGI Casing overall design:**

The SGI casing plays an important role in preventing hazardous accidents. Firstly it providers researchers and engineers with the capability to lock the compartment. This feature will prohibit unauthorized personal from misuse. Secondly the SGI casing can be mounted and un-mounted onto a surface as a precaution to falling off the bench due to its light weight and mobility. Thirdly it encompasses the “all-in-one” feature, no external equipment necessary. Lastly, it protects the Housing Unit from accumulating dust/debris which will definitely lessen the risk of contamination. Figures 12-17 demonstrate the overall design.
Figure 12. SGI Casing Front and Side View.

Figure 13. SGI Rear View.
Figure 14. SGI Top View.
Figure 15. SGI Isometrical View.
Figure 16. SGI 3D View: Front, isometric and bird eye views.

**SGI Housing Unit Compartment:**

Figure 17. The Housing Unit Compartment
**Devices and equipment Compartment:**

This compartment houses small-size instruments, equipment and devices. Such mechanisms include: pH meter, solenoid valve, peristaltic pumps, CO2 tank, and air pressure regulator as observed in figure 18.

![Figure 18. 3D View of the Device and Equipment compartment.](image-url)
**Electronics Compartment:**

The electronic compartment, as the name implies, houses all the controls, LCD display, electronics and power required for the system to operate. This compartment involves two main features, the front panel and the back main power unit. The front panel allows users to interact with the system. Such interaction involves inputs for modifying desired temperature/CO\textsubscript{2} based on the types of tissue culture required. This is accomplished using the menu options available (enter, up and down). In addition, user input for controlling culture media dispensing. Lastly, an LCD display that prints the temperature, CO\textsubscript{2}, Humidity, menu options for the user to observe (Figure 19).

![Figure 19. 3D view of the front panel](image-url)
To cool down the heat generated from power conversion (120 VAC to ±12, 5, 3 VDC) and power usage a 50mmx50mm fan is located on the lower back left side. Also a switch to completely shut down the power source. A 120 VAC cable from the outlet to the ATX power supply. Lastly, a USB serial port, which connects the SGI system to the PC. This permits SGI’s operation system to be customizable. Hence, devices can be added or removed without damaging the system peripherals. Also logging experimental data, which can be written in to text file or excel format. (Figure 20)
**SGI coming to life:**

Figure 21. SGI front and back view.

Figure 22. SGI Left/Right side views.
Figure 23. SGI Electronic and Equipment compartments.
Figure 24. SGI's Housing Unit.

(a) Housing Unit with three distinct stackable layers (Cell/Tissue Housing (CTH), Heating and Gax Units.  (b) Inside the Heating Unit, a waterproof temperature sensor, Aqua gel heating medium and the orange OEM heater.  (c) Gas chamber with the CO$_2$ sensor inside.
**Overall materials:**
The overall structure’s was driven by the design requirements and the necessary equipment. The SGI is constructed using HDPE (High-Density Polyethylene) is an extremely versatile product with outstanding properties and good chemical resistance for a wide variety of applications. HDPE has a low coefficient of friction and can be easily cut, machined, welded, and thermoformed for easy fabrication. This material will not splinter, rot or retain harmful bacteria, and is extremely resistant to cleaning agents. HDPE can be bonded with Poly-Weld and/or mechanically fastened. To provide a see-through surface, utilization of clear UV radiation blocker acrylic material is employed to observe the Housing Unit inside the casing compartment. The acrylic sheeting was chosen due to its relative inexpensive, machine-able and filtering capability of 99.93% of UV rays. Glass sheets on the other hand are heavier, more expensive and more difficult to machine.

**Expenses Summary:**
The established devices, equipment, sensors, electronics and material that have been purchased during the initial fabrication steps are listed in billing material (Table 1) of the SGI project totaled at $1,311.83. The majority of those costs comes from experimenting with different sensors, electronics, hardware, microcontrollers, serial port to test for optimal system operation as intended in the design process. Extra electronics, ICs, sensors were also purchased as backup, which bumped the price.

On the flip side, the initial fabrication phase permitted the SGI apparatus to be obtainable at reduced cost (Billing Material Table 2) subtotaling $284.55. Price quotes from Chinese venders have greatly condensed the cost analysis. These estimates were sources from Aliexpress.com and its third party affiliates.
**Bill of Material:**

(Table 1)

<table>
<thead>
<tr>
<th>Vendors</th>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sparkfun:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOL-10242</td>
<td>Solder</td>
<td>1</td>
<td>$29.95</td>
<td>$29.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAB-00065</td>
<td>Serial Cable</td>
<td>1</td>
<td>$3.95</td>
<td>$3.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAB-00064</td>
<td>Parallel Cable</td>
<td>1</td>
<td>$4.95</td>
<td>$4.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEV-09718</td>
<td>FTDI Cable</td>
<td>1</td>
<td>$17.95</td>
<td>$17.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEV-09215</td>
<td>Cable</td>
<td>4</td>
<td>$1.95</td>
<td>$7.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-00097</td>
<td>switch</td>
<td>4</td>
<td>$0.35</td>
<td>$1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-09456</td>
<td>Op-amp</td>
<td>4</td>
<td>$0.95</td>
<td>$3.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-08860</td>
<td>LED</td>
<td>2</td>
<td>$1.50</td>
<td>$3.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-08861</td>
<td>LED</td>
<td>2</td>
<td>$1.50</td>
<td>$3.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-08862</td>
<td>LED</td>
<td>2</td>
<td>$1.50</td>
<td>$3.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-09590</td>
<td>LED</td>
<td>4</td>
<td>$0.35</td>
<td>$1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-09592</td>
<td>LED</td>
<td>4</td>
<td>$0.35</td>
<td>$1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM-09594</td>
<td>LED</td>
<td>4</td>
<td>$0.35</td>
<td>$1.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOL-09235</td>
<td>SPI shortcut</td>
<td>1</td>
<td>$24.95</td>
<td>$24.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEN-10972</td>
<td>pH sensor</td>
<td>1</td>
<td>$94.95</td>
<td>$94.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOB-09981</td>
<td>I2c to UART</td>
<td>1</td>
<td>$14.95</td>
<td>$14.95</td>
<td>$10.29</td>
<td></td>
</tr>
<tr>
<td>TOL-08775</td>
<td>Solder Wick</td>
<td>1</td>
<td>$16.95</td>
<td>$16.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOL-08967</td>
<td>Liquid Flux Pen</td>
<td>3</td>
<td>$4.95</td>
<td>$14.85</td>
<td>$8.52</td>
<td></td>
</tr>
<tr>
<td>LCD-00256</td>
<td>LCD</td>
<td>1</td>
<td>$17.95</td>
<td>$17.95</td>
<td>$3.81</td>
<td></td>
</tr>
<tr>
<td>SEN-11050</td>
<td>Temp. sen</td>
<td>1</td>
<td>$9.95</td>
<td>$9.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEN-00245</td>
<td>Temp. sen</td>
<td>4</td>
<td>$4.25</td>
<td>$17.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOB-00099</td>
<td>Real time clock</td>
<td>1</td>
<td>$14.95</td>
<td>$14.95</td>
<td>$3.98</td>
<td></td>
</tr>
</tbody>
</table>

| **Mouser:** | | | | | | |
| | Item # | Description | # of units | Price | Cost | S&H |
| | | | | | | |
| 595-ULN2003AINSR | Current buffer | 6 | $0.70 | $4.20 | | |
| 579-PIC18F4553-I/P-771 | 8-bit AVR | 2 | $7.56 | $15.12 | | |
| SC16IS740IPWQ90 | I2c to UART | 6 | $2.28 | $13.68 | | |
| 579-PIC18F4550-I/P-863-VN2222LLG | 8-bit AVR | 2 | $5.59 | $11.18 | $4.99 | |
| 595-PCA9306DCTR | Mosfet | 10 | $0.46 | $4.60 | | |
| 523-115101-19-180-771 | BiDir i2c bus | 8 | $0.59 | $4.72 | | |
| SC16IS740IPWQ90 | UART Interface IC | 3 | $2.28 | $6.84 | | |
| 651-2799474 | Contact Cable | 1 | $52.85 | $52.85 | | |

**Total Cost:**

$1,311.83

$336.10

$336.10
<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>968-HIN232ACPZ</td>
<td>Rs-232 IC</td>
<td>3</td>
<td>$1.43</td>
<td>$4.29</td>
<td>↓</td>
</tr>
<tr>
<td>511-ULN2003A</td>
<td>NPN Transistor</td>
<td>6</td>
<td>$0.58</td>
<td>$3.48</td>
<td>$16.98</td>
</tr>
<tr>
<td>579-MCP3204</td>
<td>ADC-12 BIT</td>
<td>4</td>
<td>$3.34</td>
<td>$13.36</td>
<td>$4.99</td>
</tr>
<tr>
<td><strong>Total Cost:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$179.17</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Adafruit:

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS922</td>
<td>Op-am</td>
<td>4</td>
<td>$2.95</td>
<td>$11.80</td>
<td>↓</td>
</tr>
<tr>
<td>MCP23008</td>
<td>I2c 8 I/O</td>
<td>2</td>
<td>$1.95</td>
<td>$3.90</td>
<td>$9.60</td>
</tr>
<tr>
<td>74LVC245</td>
<td>8-bit logic shifter</td>
<td>6</td>
<td>$1.50</td>
<td>$9.00</td>
<td>↓</td>
</tr>
<tr>
<td>USB FTDI</td>
<td>TTL-232</td>
<td>1</td>
<td>$20.00</td>
<td>$20.00</td>
<td>↓</td>
</tr>
<tr>
<td></td>
<td>4-channel I2c Bi-direct.</td>
<td>2</td>
<td>$3.95</td>
<td>$7.90</td>
<td>$14.22</td>
</tr>
<tr>
<td><strong>Total Cost:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$76.42</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Gringer:

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>6WTU3</td>
<td>Solenoid Valve</td>
<td>1</td>
<td>$51.65</td>
<td>$51.65</td>
<td>↓</td>
</tr>
<tr>
<td>4HL96</td>
<td>Tubing Male</td>
<td>1</td>
<td>$16.79</td>
<td>$16.79</td>
<td>↓</td>
</tr>
<tr>
<td>4DHJ6</td>
<td>Connectors</td>
<td>2</td>
<td>$2.98</td>
<td>$5.96</td>
<td>$23.41</td>
</tr>
<tr>
<td><strong>Total Cost:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$97.81</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Anko Products Inc.

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>P/N is 810-205-12113-1</td>
<td>Peristaltic Motor</td>
<td>1</td>
<td>$69.70</td>
<td>$69.70</td>
<td>?</td>
</tr>
<tr>
<td><strong>Total Cost:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$69.70</strong></td>
<td></td>
</tr>
</tbody>
</table>

### CO2 meters:

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>K33-ICB</td>
<td>CO2 Sensor</td>
<td>1</td>
<td>$125.00</td>
<td>$125.00</td>
<td>?</td>
</tr>
<tr>
<td>SenseAir K33-ICB</td>
<td>CO2 Sensor</td>
<td>1</td>
<td>$75.00</td>
<td>$75.00</td>
<td></td>
</tr>
<tr>
<td><strong>Total Cost:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$200.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

### All Electronics:

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost</th>
<th>S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMT-364</td>
<td>5-wire Step motor</td>
<td>1</td>
<td>$4.35</td>
<td>$4.35</td>
<td>↓</td>
</tr>
<tr>
<td>SMT-134</td>
<td>4-wire Step Motor</td>
<td>1</td>
<td>$3.25</td>
<td>$3.25</td>
<td>↓</td>
</tr>
<tr>
<td>MD-57</td>
<td>Motor Driver</td>
<td>1</td>
<td>$8.95</td>
<td>$8.95</td>
<td>↓</td>
</tr>
<tr>
<td>LLC-45</td>
<td>Logic Level</td>
<td>2</td>
<td>$1.95</td>
<td>$3.90</td>
<td>↓</td>
</tr>
<tr>
<td>Item #</td>
<td>Description</td>
<td># of units</td>
<td>Price</td>
<td>Cost+S&amp;H</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
<td>------------</td>
<td>--------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>SOL-132</td>
<td>Solenoid Valve</td>
<td>2</td>
<td>$2.50</td>
<td>$5.00</td>
<td></td>
</tr>
<tr>
<td>CDS-1</td>
<td>CO2 sensor</td>
<td>2</td>
<td>$10.00</td>
<td>$20.00</td>
<td>$7.00</td>
</tr>
</tbody>
</table>

**Total Cost:** $52.45

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost+S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT1027BCN8-5#PBF</td>
<td>5V Ref PDIP</td>
<td>4</td>
<td>$6.17</td>
<td>$24.68</td>
</tr>
</tbody>
</table>

**Total Cost:** $34.69

<table>
<thead>
<tr>
<th>Item #</th>
<th>Description</th>
<th># of units</th>
<th>Price</th>
<th>Cost+S&amp;H</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDPE sheets</td>
<td>1’x3’ sheets</td>
<td>4</td>
<td>$20.00</td>
<td>$80.00</td>
</tr>
<tr>
<td>Poly-Weld</td>
<td>Tap-poly weld</td>
<td>2</td>
<td>$15.00</td>
<td>$30.00</td>
</tr>
</tbody>
</table>
(Table 2)

<table>
<thead>
<tr>
<th>Items</th>
<th>price</th>
<th># Item</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(CO2-Sensors)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO2 sensor (HM-Z14)</td>
<td>$33.00</td>
<td>1</td>
<td>$33.00</td>
</tr>
<tr>
<td><strong>(pH sensor)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ph sensor</td>
<td>$18.00</td>
<td>1</td>
<td>$18.00</td>
</tr>
<tr>
<td>ph-probe</td>
<td>$8.56</td>
<td>1</td>
<td>$8.56</td>
</tr>
<tr>
<td><strong>(Peristaltic Pumps)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peristaltic pump</td>
<td>$6.99</td>
<td>2</td>
<td>$13.98</td>
</tr>
<tr>
<td>Peristaltic pump hose/meter</td>
<td>$1.15</td>
<td>1</td>
<td>$1.15</td>
</tr>
<tr>
<td><strong>(CO2-Injection System components)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CO2 Flow Regulator (or)</td>
<td>$9.27</td>
<td>1</td>
<td>$9.27</td>
</tr>
<tr>
<td>CO2 Adapter Gauge (or)</td>
<td>$6.99</td>
<td>1</td>
<td>$6.99</td>
</tr>
<tr>
<td>UP CO2 Adjuster pierce (88 co2 cartridge tanks only)</td>
<td>$18.99</td>
<td>1</td>
<td>$11.00</td>
</tr>
<tr>
<td></td>
<td>$11.00</td>
<td>1</td>
<td>$11.00</td>
</tr>
<tr>
<td><strong>(Solenoid Valves)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solenoid Valve</td>
<td>$10.00</td>
<td>1</td>
<td>$10.00</td>
</tr>
<tr>
<td>Pressure Regulator</td>
<td>$4.50</td>
<td>1</td>
<td>$4.50</td>
</tr>
<tr>
<td><strong>Temperature and Heater system</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature Sensor (DS18B20)</td>
<td>$1.60</td>
<td>1</td>
<td>$1.60</td>
</tr>
<tr>
<td>Silicon Heater (12 v, 50w) 100mmX100mm</td>
<td>$3.50</td>
<td>1</td>
<td>$3.50</td>
</tr>
<tr>
<td><strong>Electronics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>$11.00</td>
<td>1</td>
<td>$11.00</td>
</tr>
<tr>
<td>LCD Display</td>
<td>$1.88</td>
<td>1</td>
<td>$1.88</td>
</tr>
<tr>
<td>5 Volt relays</td>
<td>$0.56</td>
<td>5</td>
<td>$2.80</td>
</tr>
<tr>
<td>5 volt power regulators (100 pcs)</td>
<td>$0.10</td>
<td>3</td>
<td>$0.30</td>
</tr>
<tr>
<td>Heat sinks</td>
<td>$0.09</td>
<td>3</td>
<td>$0.27</td>
</tr>
<tr>
<td>Switching Transistors</td>
<td>$0.01</td>
<td>5</td>
<td>$0.07</td>
</tr>
<tr>
<td>buttons</td>
<td>$0.15</td>
<td>10</td>
<td>$1.50</td>
</tr>
<tr>
<td>LEDs</td>
<td>$0.10</td>
<td>5</td>
<td>$0.50</td>
</tr>
<tr>
<td>Power Connectors</td>
<td>$0.10</td>
<td>10</td>
<td>$1.00</td>
</tr>
<tr>
<td>LCD-Ribbon cables/ft</td>
<td>$0.07</td>
<td>1</td>
<td>$0.07</td>
</tr>
</tbody>
</table>

Total Cost $284.55
<table>
<thead>
<tr>
<th>Item Description</th>
<th>Quantity</th>
<th>Unit Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Air Filter</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Screw Caps with HEPA .22um filter</td>
<td>1</td>
<td>$7.50</td>
<td>$7.50</td>
</tr>
<tr>
<td>Plated Brass Male Luer Lock To 1/8” NPT (M)</td>
<td>1</td>
<td>$2.55</td>
<td>$2.55</td>
</tr>
<tr>
<td><strong>NPT fittings</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/8” Male x 1/4” Male Adapter</td>
<td>2</td>
<td>$0.59</td>
<td>$1.18</td>
</tr>
<tr>
<td>1/8”Threaded Plastic NPT to Hose Barb (100 pcs)</td>
<td>3</td>
<td>$0.21</td>
<td>$0.63</td>
</tr>
<tr>
<td><strong>Housing Design</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDPE 6”x6”x2”</td>
<td>3</td>
<td>$20.00</td>
<td>$60.00</td>
</tr>
<tr>
<td><strong>Case Design</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDPE 1’x3’x1/16”</td>
<td>15x4</td>
<td></td>
<td>$60.00</td>
</tr>
<tr>
<td><strong>Optional:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check Valve regulator</td>
<td>1</td>
<td>$3.68</td>
<td>$3.68</td>
</tr>
<tr>
<td>RTC</td>
<td>1</td>
<td>$2.95</td>
<td>$2.95</td>
</tr>
<tr>
<td>(Humidity sensor + Temp system)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Humidity Sensor + Temp (SHT25)</td>
<td>1</td>
<td>$11.26</td>
<td>$11.26</td>
</tr>
<tr>
<td>Breakout board for SHT25</td>
<td>1</td>
<td>$0.45</td>
<td>$0.45</td>
</tr>
</tbody>
</table>
Chapter 3 Results: Validation of the SGI system

Temperature

In order to characterize the efficiency of the temperature control, data was collected during the heating cycle. The heating rate, range of temperatures and accuracy of maintaining the set temperature was analyzed over one hour interval. The highest and lowest temperature reached was 37.26°C and 36.81°C respectively, as demonstrated in figure 25. The heater is set to trigger at 36.9°C and off at 37.0°C. This meets the design requirements, therefore, should sustain environment thermal regulation required for cells viability.

![SGI Temperature Fluctuations Graph](image)

Figure 25. Temperature Fluctuations Graph

CO2

The SGI system is capable of raising the CO2 percentage up to 30%, however to maintain viable cells, CO2 must be kept at 5% for mammalian cells to survive. This caused the 20 g CO2 tank to become empty after two months of maintaining the CO2 percentage at 5%, running several experiments. The rate of raising the CO2 percentage and accuracy of maintaining a certain percentage was also studied. The CO2 system is set
to inject step-like increments, starting at 0 and ending at 4.9%. One single injecting is capable of inducing 0.15% CO2 gas concentration as seen in figure 26. This could be adjusted based on the pressure set by the regulator. The accuracy of maintaining 5% CO2 was found to be ± 0.0975%. Which is well within acceptable range for maintaining pH levels for cell viability.

Figure 26. CO2 Fluctuations Graph

**Humidity:**

SHT15 humidity sensor was used to determine the RH inside the Cell/Tissue Housing (CTH) unit. The humidity reading was gathered. The sensors were found to have an accuracy of ± 2% RH. According to the statistics, maximum humidity reached approximately 84% at a temperature of 37°C as shown in Figure 27. This indicates that Tissue/Cell Housing unit can maintain humidity for cells to be feasible for producing tissue engineering constructs.
Figure 27. Humidity, Temperature vs Time Graphical Analysis.
Application of SGI using bone tissue engineering as a model system:

**Abstract:**
Fracture nonunion, osteoarthritis and other traumatic injuries of underlying bone represent a major problem in regenerative medicine. Autologous bone harvested from other anatomic locations and inserted into the defect site is a representative therapy. Bone regeneration via tissue engineering (TE) has become a focus in regenerative medicine. Bone tissue engineering of autologous osteoprogenitor cells combined with biodegradable scaffold has emerged as an alternative approach in treatment of large bone defects[1-3]. Adult human mesenchymal stem cells (MSCs) are easier to obtain and show a strong capacity for expansion in vitro while maintaining their multi-lineage differentiation potential, including cartilage, bone, adipose, muscle, tendon, stroma, and ligament[4]. MSCs have been widely used in tissue engineering and gene therapy[5].

**Materials and Methods**

**Cell culture**
Human Mesenchymal stem cells (hMSCs) isolated from fresh adult human bone marrow (CD105+, CD166+, CD29+, CD44+, CD14-, CD34-, CD45-) were commercially available from Lonza (Lonza Walkersville, Inc., Walkersville, MD, USA). Cells were cultured in a basic culture medium composed of Dulbecco’s Modified Eagle’s Medium (Gibco, Carlsbad, CA) supplemented with 10% fetal bovine serum (Gibco, Carlsbad, CA), 1% antibiotics/antimycotics (Invitrogen, Carlsbad, CA) at 37 °C and 5% CO2. The osteogenic culture medium was created by adding 10 nM dexamethasone, 10 mM β-glycerophosphate and 50 μg/ml ascorbic acid-2-phosphate (Sigma, St. Louis, MO) to the basic culture medium. The control group was cultured in only the basic culture medium.
**Gelatin sponge scaffold preparation**

A biodegradable and biocompatible sterile gelatin (Pharmacia & Upjohn, Kalamazoo, MI) was used as the biological scaffold. The gelatin sponge was punched at 5mm diameter, 3mm thickness, and prewetted with basic medium for 1 h. Tissue-engineered osteogenic constructs were prepared by immersing the prewetted sponges into a prepared hMSCs suspension with a cell density of $1 \times 10^5$ cells/scaffold in a 1.5 ml centrifuge tube. A slight vacuum was created in the tube by using a 20-ml syringe to accelerate cell seeding evenly throughout the sponges [6]. Then the mixture of sponges and cell suspension was incubated at 37 °C for 2 h. The cell-seeded constructs were subsequently transferred to a 24-cell culture plate, and osteogenic medium was added and allowed to grow in vitro at normal incubator and SGI incubator for two weeks in a time course study. Replace with fresh media twice a week.

**Scanning Electron Microscopy (SEM)**

For SEM, constructs were fixed for 30 m in the 10% neutral buffered formalin (VWR international, West Chester, PA). The constructs were subsequently dehydrated using a series of graded alcohols, and dried using critical point drying. The constructs were gold-sputtered prior to evaluation.

**RNA extraction and cDNA synthesis**

Fresh construct (weekly intervals up to 4 weeks) were transferred into 1.5 ml centrifuge tubes. Samples were homogenized in 1.5 ml Trizol (Life technologies) using mortar plus liquid nitrogen and RNA was extracted according to the single-step acid-phenol-guanidinium method [7]. The RNA samples were reverse transcribed into cDNA using oligo (dT)-selection according to the manufacture’s protocol (Reverse Transcription kit, Qiagen). A 300 ng total RNA sample was used for the single strand cDNA synthesis. The
reverse transcription reaction was incubated at 42 °C for 30 min, and terminated at 95 °C for 3 min.

**Real time reverse transcription-polymerase chain reaction (real-time RT-PCR)**

Alkaline phosphatase (ALP), Bone Sialo-protein (BSP) transcript level were quantified using Fast SYBR® Green Master Mix (Life Technologies) and the ABI Prism 7000 real time PCR system (Applied Biosystems, Barlsbad, CA). After a 10 min denaturation step at 95°C, cDNA was amplified by performing two-step PCR cycles: a 3 s step at 95°C, followed by a 15 s step at 60°C. The transcript data were normalized to the housekeeping gene, glyceraldehydes-3-phosphate-dehydrogenase (GAPDH). Reactions were performed in triplicate. Expression of target genes was normalized to GAPDH and expressed as the fold ratio relative to the control group, using the 2-ΔΔCT method [8]. Specific genes and primer sequences are listed in Table 3.
<table>
<thead>
<tr>
<th>Gene</th>
<th>Primer Sequence</th>
</tr>
</thead>
</table>
| **BSP** | F: 5’- AATTTCCAGTTCCAGGGCAGTAG-3’  
          | R: 5’- GTGTGGTAATTTCAGCCTCAG-3’  
          | F: 5’-GACCATTCCCACGTCTTCACA-3’  
          | R: 5’-GGGCCAGACCAAAGATAGGTGTTG-3’ |
| **ALP** | F: 5’-TCCACTGGCGTCTTCACC-3’  
          | R: 5’-GGCAGAGATGATGACCCCTTT-3’ |
| **GAPDH** | |

**SGI Results Summary:**
MSCs are multipotent stromal cells that can differentiate into a variety of cell type, including osteoblasts, chondrocytes, and adipocytes[4]. MScs are characterized morphologically by a small cell body with a few cell processes that are long and thin. The cell body contains a large and round nucleus with an obvious nucleolus (Figure 28). MSCs cultured in osteogenetic medium on the gelatin scaffolds under normal incubator and SGI incubator were observed by SEM (Figure 29). MSCs cultivation resulted in the formation of continuous sheets of cells on surfaces of gelatin scaffolds after 2 weeks in culture.
Figure 28. Human Mesenchymal Stem Cells Morphology.

Human bone marrow derived Mesenchymal stem cell showing fibroblast like morphology seen under phase contrast microscope at 20 x magnification.

Figure 29. Cell Morphology and Attachment.

Scanning electron micrographs of hMSC cultured in osteogenic medium on gelatin scaffolds under normal incubator and education incubator. Face view images of hMSC cultured for 2 weeks under (A, C) normal incubator and (B, D) SGI incubator. 200 Am (A, B), and 100 Am (C, D).

Transcript levels of osteogenetic markers among the SGI incubator and normal incubator at two weeks of incubation were assessed using RT-PCR to determine the levels of osteogenetic differentiation. Levels of ALP and BSP, osteogenetic related genes, were analyzed and normalized to GAPDH within the linear range of amplification. As
shown in Figure 30, transcript levels of ALP and BSP in cells within gelatin constructs cultured under SGI incubator were upregulated in comparison to the control, implied the osteogenetic differentiation under SGI incubator culture. However, the expression of ALP and BSP cultured under SGI culture were lower than normal incubator culture.

![Figure 30. RT-PCR Analysis of ALP and BSP.](image)

Transcript levels for osteogenetic markers in MSCs cells within the gelatin construct cultured under the Second Generation Incubator (SGI) apparatus and commercial incubator over 2 weeks. Data are shown relative to the expression of the respective transcript by undifferentiated MSCs at week 0 and represented as the average ± standard deviation.
Conclusion:

To summary, the main objective of the thesis is to design, build and test Second Generation Incubator (SGI) culturing apparatus that can simulate in vivo conditions under standard culture frames and sustain cell life to produce a viable tissue engineered construct.

The SGI apparatus is designed to provide the optimum homeostatic environment for cells to thrive and produce quality tissue engineering constructs equivocal to that of commercial incubators. Amongst many features such as; ease of use, customizability, compatibility, portability and low cost, special attention was given to the design aspect of providing proper sterile conditions. SGI’s purpose is to provide students, engineers and scientist with a tool that would revolutionize traditional incubators in terms of the topologies mentioned above. In addition, benefiting the tissue engineering field by providing an apparatus that is capable of being intertwined with conventional bioreactors and produce patient-specific, disposable, tissue engineered constructs at low cost. This will offset the lag that is hindering tissue engineering products from the research phase to the clinical trial applications.

Even though SGI established great sterile environment and maintained homeostatic conditions for cells to thrive, there were some difficulties in producing viable tissue engineered bone constructs for application use. According to the ALP and BSP gene expressions observed for both SGI and commercial incubator, the commercial incubator provided better tissue engineered constructs. Meaning that, there tends to be less live cells in the SGI. Important note must be taken, SGI is in its toddler phase and
much work needs to be done to bring lifesaving tissue engineered constructs to the people you love.
References:


SGI References:


Appendices:

Programming Code:

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <LiquidCrystal.h>
#include <LiquidCrystalFast.h>
#include <OneWire.h>
#include <Wire.h>
#include <EEPROM.h>
#define MOVECURSOR 1  // constants for indicating whether cursor should be redrawn
#define MOVELIST 2  // constants for indicating whether cursor should be redrawn
byte totalRows = 4;  // total rows of LCD
#define TEMPR_STORE   500 // Where in the EEPROM we store the set temperature
#define CO2_STORE   1000

#include <SoftwareSerial.h>
#include <Chronodot.h>
SoftwareSerial mySerial(20, 21);// RX, TX

// Arduino Mega Digital I/O pings>
// Rs, E, DB4,DB5,DB6,DB7>
// 1  2  3   4   5  6   7   8   9  10  11  12  13 14 15 16 <- LCD Pins
//GND +5 GND RS GND E DB0 DB1 DB2 DB3 DB4 DB5 DB6 DB7 5 GND <- LCD Connections
// 22 23 24 25 26 27 <LCD to Mega interface
/*
***************************************************
* Timer Variables
***************************************************/
    double presetTemp=37.0;
    double CurrTemp=25.5;
    double temp=(presetTemp-CurrTemp)/1.25;
    int minute=temp;
    int Seconds=(temp-minute)*100;
    int second=((Seconds*60 /100));
    int hour=0; // declare time variables

// variables
    double currTemp = 0;    // The currently read temperature
    double setTemp = 0;    // The set temperature
    float setCO2=0;
    byte hasReset = 0;
    long lastUpdate = 0;
    boolean stillSelecting =false;

//variables
    byte present = 0;
    byte data[12];
    byte addr[8];
    boolean CO22State=0;
    boolean CO22indicator=0;
```
boolean Mstate1=0;
boolean Mstate2=0;
const int  LedMediaOut = A3;  //led
const int  BMediaOut = A1;  //button
const int  LedMediaIn = A2;  //led
const int  BMediaIn = A0;  //button
const int  MEDIAin=1;
const int  MEDIAout=2;
boolean state1;
boolean state2;
boolean LEDstate1 = 0;    //state1 of button pin
boolean LEDstate2 = 0;    //state1 of button pin

unsigned long button_time1 = 0;
unsigned long last_button_time1 = 0;
unsigned long button_time2 = 0;
unsigned long last_button_time2 = 0;
unsigned long Timer = 0;    // will store last time LED was updated
unsigned long lastTimer = 0;         // interval at which to blink (milliseconds)

// Buttons
byte lastButton[2];                     // Track the button states
byte buttonPins[2];                     // Which pins the buttons are on

long lastPress = 0;                    // When the last button was pressed
byte buttonUp=A4;
byte buttonDown=A5;
byte buttonSelect=A6;

byte totalCols = 20;                   // total columns of LCD
unsigned long timeoutTime = 0;         // this is set and compared to millis to see when the user last input
const int menuTimeout = 10000;         // time to timeout in a menu when user doesn't do anything.
unsigned long lastButtonPressed;      // this is when the last button was pressed. It's used to debounce.
const int debounceTime = 200;          //this is the debounce and hold delay. Otherwise, you will FLY
//through the menu by touching the button.

unsigned long BMenu_Time=0;
unsigned long BMenu_LastTime= 0;
boolean scroll=true;
boolean UDscroll=false;
boolean CO2SubMenu=0;
boolean TempSubMenu=0;

LiquidCrystalFast lcd(6,5,12,13,14,22);
Onewire ds(23);                      // temp probe pin connection
//Chronodot RTC;
byte cmdRead[] = {0xFE, 0X44, 0X00, 0X08, 0X02, 0X9F, 0X25};
int co2Addr = 0x68;
int Heater =19 ;
int CO2LED=3;
double Temp1;
double Temp;
int count = 0;
double CO2;
double CO22;
double pH1;
int seconds;
int Blink=21;
int Debug=13; //debug

/**************************************************
* Initialize system’s inputs and outputs
**************************************************/

void setup(){
  Serial.begin(9600);
  Serial1.begin(38400);
  //mySerial.begin(9600);
  Wire.begin();
  /*
   RTC.begin();
   */
  pinMode(Blink,OUTPUT);
  pinMode(Debug, OUTPUT);
  pinMode(Heater,OUTPUT);
  pinMode(CO2LED,OUTPUT);
  pinMode(CO2LED2,OUTPUT);
  pinMode(LedMediaOut, OUTPUT);  //led output
  pinMode(LedMediaIn, OUTPUT);
  pinMode(MEDIAout, OUTPUT);
  pinMode(MEDIAin, OUTPUT);
  pinMode(BMediaOut, INPUT);  //button input
  pinMode(BMediaIn, INPUT);
  pinMode(buttonUp, INPUT);
  pinMode(buttonDown, INPUT);
  pinMode(buttonSelect, INPUT);
  setCO2=5.0;
  setTemp=37.0;  // Default temperature is 37.0 C
  // See if we can read the set temperature out of the EEPROM
  byte gCO2=EEPROM.read(CO2_STORE);
  if (gCO2 > 0) setCO2=(float)gCO2 /2;
  byte gTemp = EEPROM.read(TEMPR_STORE);
  if (gTemp > 0) setTemp = (float)gTemp / 2; // Temperature is stored doubled

  boolean Menu=0;
  lcd.begin(20,4);   // columns, rows. use 16,2 for a 20x4 LCD, etc.
  lcd.clear();      // start with a blank screen
  lcd.setCursor(5,0); // set cursor to column 0, row 0 (the first row)
  lcd.print("INCUBATOR");  // change this text to whatever you like. keep it clean.
lcd.setCursor(0,1);
lcd.print("Temp: ");
lcd.setCursor(12,1);
lcd.print("C");
lcd.setCursor(0,2);
lcd.print("CO2 Sensor:");
//lcd.setCursor(10,2);
//lcd.print("rH:");
lcd.setCursor(15,2);
//lcd.setCursor(17,2);
//lcd.print("%");
lcd.setCursor(0,3);
lcd.print("pH Sensor:");
MCUCR|= (1<<JTD);            //disable jtag on atmega644p
MCUCR|= (1<<JTD);
delay(5000);
}
long currentMillis3 = 0;       // will store last time LED was updated
long interval3 = 5000;          // interval at which to blink (milliseconds)
unsigned long previousMillis3=0;
boolean Settings=0;

void loop(){
  if (digitalRead(buttonSelect)==HIGH)
  {
    Settings=1;
    run_Menu();
    countdown();
    TimerOutput();
  }
  if (digitalRead(buttonSelect)==LOW && Settings==1)
  {
    MenuHome();
    Settings=0;
    //Runsys();
  }
  if (digitalRead(buttonSelect)==LOW && Settings==0)
  {
    Runsys();
  }
}

/*
 ********************************************
* System Operation
 *********************************************/
void Runsys()
{
    unsigned long currentMillis3 = millis();
    if(currentMillis3 - previousMillis3 > interval3)
    {
        MediaOut();
        Medialn();
        readTempSensor();
        Heat();
        lcd.setCursor(5,1);
        lcd.print(Temp1,2);
        CO2I2C();
        //rHI2C();
        CO2inc();
        readpH();
        lcd.setCursor(11,3);
        lcd.print(pH1,2);
        //MIOSafty();
        //Time();
        logData();
        //LEDBlink();
        //readCO2();
        //CO2Display();
        previousMillis3 = currentMillis3;
    }
}

/*
*****************************************************************************************/

* Menu
*****************************************************************************************/

void MenuHome(){
    stillSelecting=false;
    //Serial.print(" ");
    //Serial.print(setTemp);
    //Serial.print(" ");
    //Serial.print(setCO2);
    lcd.begin(20,4);     // columns, rows. use 16,2 for a 20x4 LCD, etc.
    lcd.clear();        // start with a blank screen
    lcd.setCursor(5,0);   // set cursor to column 0, row 0 (the first row)
    lcd.print("iNCUBATOR"); // change this text to whatever you like. keep it clean.
    lcd.setCursor(0,1);
    lcd.print("Temp:");
    lcd.setCursor(12,1);
    lcd.print("C");
    lcd.setCursor(0,2);
    lcd.print("CO2 Sensor:");
    //lcd.setCursor(10,2);
    //lcd.print("rH");
}
lcd.setCursor(15,2);
lcd.print("%");
lcd.setCursor(17,2); //lcd.print("%");
lcd.setCursor(0,3);
lcd.print("pH Sensor:");
delay(1000);
}

void DefaulthomeScreen()
{
stillSelecting=false;
lcd.setCursor(5,0); // set cursor to column 0, row 0 (the first row)
lcd.print("iNCUBATOR"); // change this text to whatever you like. keep it clean.
lcd.setCursor(0,1);
lcd.print("Temp:");
lcd.setCursor(12,1);
lcd.print("C");
lcd.setCursor(0,2);
lcd.print("CO2 Sensor:");
//lcd.setCursor(10,2);
//lcd.print("rH:");
lcd.setCursor(15,2);
lcd.print("%");
//lcd.setCursor(17,2);
//lcd.print("%");
lcd.setCursor(0,3);
lcd.print("pH Sensor:");
}

/*************************************************************************
**                                 Data Logging
*****************
********************************************************/
void logData ()
{
Serial.print("#S|DATALOGS|[");
Serial.print("t");
Serial.print("Temp (C):");
Serial.print(Temp1);
Serial.print("t");
Serial.print("pH:");
Serial.print(pH1);
Serial.print("t");
Serial.print("CO2(%):" );
Serial.print(CO22);
Serial.print("t");
Serial.println("#]");
}

/*************************************************************************
**                                 CO2 Readi
**                                  UART
*************************************************************************/
void CO2Display()
{
}
```cpp
lcd.setCursor(11,2);
lcd.print(CO2);
}

double readCO2()
{
  mySerial.write(cmdRead,7);
  while (mySerial.available()<7)
    CO2=(getCO2() /1000);
  CO2indicator=1;
  return CO2;
}

double getCO2()
{
  byte i;
  byte packet[7];
  for(int i=0; i<7; i++)
    {
      packet[i] = mySerial.read();   //create array from packet
    }
  int low = packet[4];            //low byte for value is 5th byte in the packet
  int high = packet[3];           //high byte for value is 4th byte in packet in the packet
  int val = high*256 + low;      //combine high byte and low byte
  return val;
}

double readCO2()
{
  Serial1.write(cmdRead,7);
  while (Serial1.available()<7)
    CO2=(getCO2() /1000);
  CO2indicator=0;
  return CO2;
}
```
void wakeSensor() {
// This command serves as a wakeup to the CO2 sensor, for K33-ELG/BLG Sensors Only
// You’ll have the look up the registers for your specific device, but the idea here is simple:
// 1. Disabled the I2C engine on the AVR
// 2. Set the Data Direction register to output on the SDA line
// 3. Toggle the line low for ~1ms to wake the micro up. Enable I2C Engine
// 4. Wake a millisecond.
TWCR &= ~(1<<2); // Disable I2C Engine
DDRC |= (1<<4); // Set pin to output mode
PORTC &= ~(1<<4); // Pull pin low
delay(1);
PORTC |= (1<<4); // Pull pin high again
TWCR |= (1<<2); // I2C is now enabled
delay(1);
}

void initPoll() {
Wire.beginTransmission(co2Addr);
Wire.write(0x11);
Wire.write(0x00);
Wire.write(0x60);
Wire.write(0x35);
Wire.write(0xA6);
Wire.endTransmission();
delay(20);
Wire.requestFrom(co2Addr, 2);
byte i = 0;
byte buffer[2] = {0, 0};
while(Wire.available()) {
buffer[i] = Wire.read();
i++;
}

double readCo2() {
int co2_value = 0;
// We will store the CO2 value inside this variable.
//digitalWrite(13, HIGH);
// On most Arduino platforms this pin is used as an indicator light.
SSID://Begin Write Sequence */
SSID://Begin Write Sequence */
Wire.beginTransmission(co2Addr);
Wire.write(0x22);
Wire.write(0x00);
Wire.write(0x08);
Wire.write(0x2A);
Wire.endTransmission();
/*
We wait 10ms for the sensor to process our command.
The sensor's primary duties are to accurately
measure CO2 values. Waiting 10ms will ensure the
data is properly written to RAM
*/
delay(20);

/*****************************/
/* Begin Read Sequence */
/*****************************/
/*
Since we requested 2 bytes from the sensor we must
read in 4 bytes. This includes the payload, checksum,
and command status byte.*/
Wire.requestFrom(co2Addr, 4);
byte i = 0;
byte buffer[4] = {0, 0, 0, 0};
/*
Wire.available() is not necessary. Implementation is obscure but we leave
it in here for portability and to future proof our code
*/
while(Wire.available()) {
  buffer[i] = Wire.read();
i++;
}
co2_value = 0;
co2_value |= buffer[1] & 0xFF;
co2_value = co2_value << 8;
co2_value |= buffer[2] & 0xFF;
byte sum = 0; //Checksum Byte
sum = buffer[0] + buffer[1] + buffer[2]; //Byte addition utilizes overflow
if(sum == buffer[3]) {
  // Success!
  //digitalWrite(13, LOW);
  return ((double) co2_value / (double) 1);
}
else {
  // Failure!
  /*
  Checksum failure can be due to a number of factors,
fuzzy electrons, sensor busy, etc.
  */
  //digitalWrite(13, LOW);
  return (double) -1;
}

/*****************************/
// Function : double readTemp()
// Returns : The current Temperature Value, -1 if error has occurred
///////////////////////////////////////////////////////////////////
double readTemperatures() {
    int tempV = 0;
    Wire.beginTransmission(co2Addr);
    Wire.write(0x22);
    Wire.write(0x00);
    Wire.write(0x12);
    Wire.write(0x34);
    Wire.endTransmission();
    delay(20);
    Wire.requestFrom(co2Addr, 4);
    byte i = 0;
    byte buffer[4] = {0, 0, 0, 0};
    while(Wire.available()) {
        buffer[i] = Wire.read();
        i++;
    }
    tempV = 0;
    tempV |= buffer[1] & 0xFF;
    tempV = tempV << 8;
    tempV |= buffer[2] & 0xFF;
    byte sum = 0;  // Checksum Byte
    sum = buffer[0] + buffer[1] + buffer[2];  // Byte addition utilizes overflow
    if(sum == buffer[3]) {
        return ((double) tempV / (double) 100);
    } else {
        return -1;
    }
}

///////////////////////////////////////////////////////////////////
// Function : double readRh()
// Returns : The current Rh Value, -1 if error has occurred
///////////////////////////////////////////////////////////////////
double readRh() {
    int tempV = 0;
    Wire.beginTransmission(co2Addr);
    Wire.write(0x22);
    Wire.write(0x00);
    Wire.write(0x14);
    Wire.write(0x36);
    Wire.endTransmission();
    delay(20);
    Wire.requestFrom(co2Addr, 4);
    byte i = 0;
    byte buffer[4] = {0, 0, 0, 0};
    while(Wire.available()) {
        buffer[i] = Wire.read();
        i++;
    }
tempV = 0;
tempV |= buffer[1] & 0xFF;
tempV = tempV << 8;
tempV |= buffer[2] & 0xFF;
byte sum = 0; //Checksum Byte
sum = buffer[0] + buffer[1] + buffer[2]; //Byte addition utilizes overflow
if(sum == buffer[3]) {
  //digitalWrite(13, LOW);
  return (double) tempV / (double) 100;
}
else {
  //digitalWrite(13, LOW);
  return -1;
}
}

void CO2I2C(){
  wakeSensor();
  initPoll();
  delay(100);
  double tempV = readTemperatures();
  Serial.print(tempV);
  delay(20);
  wakeSensor();
  double rhValue = readRh();
  Serial.print(rhValue);
  delay(20);
  wakeSensor();
  double co2Value = readCo2();
  delay(20);

  if(co2Value >= 0) {
    CO22 = (co2Value/1000);
    lcd.setCursor(11,2);
    lcd.print(CO22);
  }
}

void rHI2C(){
  wakeSensor();
  initPoll();
  delay(100);
  double rhValue = readRh();
  if(rhValue >= 0) {
    lcd.setCursor(13,2);
    lcd.print(rhValue);
  }
}

/********************************************
*************************
**                                 CO2 injection Algorithm
*************************************************************************/
long previousMillis1 = 0;    // will store last time LED was updated
long interval1 = 30000;  // interval at which to blink (milliseconds) 40 secs

void CO2inc()
{
  double minCO2=CO22;
  unsigned long currentMillis1 = millis();

  if(currentMillis1 - previousMillis1 > interval1)
  {
    previousMillis1 = currentMillis1;

    if (CO22 < 3.10 && minCO2 >= 0 && Mstate1==0 && Mstate2==0)
    {
      digitalWrite(CO2LED, HIGH);
      delay(18);
      digitalWrite(CO2LED, LOW);
    }
    if (CO22 < 4.90 && minCO2 >= 3.10 && Mstate1==0 && Mstate2==0)
    {
      digitalWrite(CO2LED, HIGH);
      delay(17);
      digitalWrite(CO2LED, LOW);
    }
  }
  if (CO22 >= 4.90)
  {
    digitalWrite(CO2LED, LOW);
    //CO22State=0;
    }
}

/*
long previousMillis1 = 0;  // will store last time LED was updated
long interval1 = 10000;  // interval at which to blink (milliseconds) 40 secs

void CO2inc()
{
  double minCO2=CO2;
  unsigned long currentMillis1 = millis();

  if(currentMillis1 - previousMillis1 > interval1)
  {
    previousMillis1 = currentMillis1;

    if (CO2 < 4.10 && minCO2 >= 0.01 && Mstate1==0 && Mstate2==0)
    {
      digitalWrite(CO2LED, HIGH);
      delay(20);
      digitalWrite(CO2LED, LOW);
    }
  }*/
if (CO2 < 4.90 && minCO2 >= 4.10 && Mstate1==0 && Mstate2==0) {
    digitalWrite(CO2LED, HIGH);
    delay(20);
    digitalWrite(CO2LED, LOW);
}

if (CO2 >= 4.90) {
    digitalWrite(CO2LED, LOW);
}
}

/***************************************************************************/
/**                  Temp Reading                                        **
***************************************************************************/

void readTempSensor()
{
    byte j;
    if ( !ds.search(addr)) {
        return;
    }
    present = ds.reset();
    ds.select(addr);
    ds.write(0x44,1);    // start conversion, with parasite power on at the end
    //delay(1000);      // maybe 750ms is enough, maybe not
    // we might do a ds.depower() here, but the reset will take care of it.
    for ( j = 0; j < 11; j++) {
        data[j] = ds.read();    // we need 12 bytes
    }
    Temp=(data[1]<<8)+data[0];   //take the two bytes from the response relating to temperature
    Temp1=Temp/16;  //divide by 16 to get pure celcius readout
    //Serial.print("P=");
    //Serial.print(present,HEX);
    //Serial.print(" ");
    for ( j = 0; j < 11; j++) {
        // we need 12 bytes
        data[j] = ds.read();
    }
    Temp=(data[1]<<8)+data[0];   //take the two bytes from the response relating to temperature
    Temp1=Temp/16;  //divide by 16 to get pure celcius readout
    //Serial.print("T=");  //output the temperature to serial port
    //Serial.print(Temp1,1);
    // Serial.print(" ");
}
/**
 * Heater
 */

void Heat()
{
    if (Mstate1==1 || Mstate2 == 1)
    {
        digitalWrite(Heater, LOW);
    }
    else if (Temp1 < 36.90 && Temp1 >= 36.0 && Mstate1==0 && Mstate2==0 )
    {
        digitalWrite(Heater, HIGH);
        delay(5000);
        digitalWrite(Heater, LOW);
        delay(2000);
    }
    else if (Temp1 < 36.0 && Temp1 > 0.0 && Mstate1==0 && Mstate2==0 )
    {
        digitalWrite(Heater, HIGH);
        delay(10000);
        digitalWrite(Heater, LOW);
        delay(3000);
    }
    else
        digitalWrite(Heater,LOW);
}

/*************************************************************************
 **                                 pH Reading
 *************************************************************************/

void readpH()
{
    char inData[7];
    byte index = 0;
    Serial1.print('R');
    Serial1 .print('r');
    while(Serial1.available() > 0 && index<6 )
    {
        for (int i=0; i < 7; i++)
        {
            inData[index] = Serial1.read();
            index++;
            inData[index] = '\r';
        }
        pH1=(atof(inData));
        //Serial.print(pH1);
        index=0;
        return;
    }
}
/*************************************************************************/  
**                                Media Exchange                     */  
*************************************************************************/

void MediaOut(){
  button_time1 = millis();
  if (button_time1 - last_button_time1 > 1000 )
  {
    state1 = (digitalRead(BMediaOut));
    state2 = (digitalRead(BMediaIn));
    if (state1 == 1){
      switch (LEDstate1){
        case 0:
          digitalWrite(LedMediaOut, HIGH);
          digitalWrite(MEDIAout, HIGH);
          digitalWrite(LedMediaIn, LOW);
          digitalWrite(MEDIAin, LOW);
          Mstate1=1;
          LEDstate1=1;
          break;
        case 1:
          digitalWrite(LedMediaOut, LOW);
          digitalWrite(MEDIAout, LOW);
          LEDstate1=0;
          Mstate1=0;
          break;
      }
    }
    last_button_time1 = button_time1;
  }
}

void MIOSafty()
{
  if (Mstate1==1 || LEDstate1==1 || LEDstate2==1 || Mstate2==1)
  {
    Timer=millis();
    if (Timer - lastTimer > 10000)
    {
      digitalWrite(LedMediaIn, LOW);
      digitalWrite(MEDIAin, LOW);
      digitalWrite(MEDIAout, LOW);
      digitalWrite(LedMediaOut, LOW);
    }
    lastTimer=Timer;
  }
}
void MediaIn(){
    button_time2 = millis();
    if (button_time2 - last_button_time2 > 1000 )
    {
        state2 = (digitalRead(BMediaIn));
        if (state2 == 1){
            switch (LEDstate2){
            case 0:
                digitalWrite(LedMediaIn, HIGH);
                digitalWrite(MEDIAin, HIGH);
                digitalWrite(MEDIAout, LOW);
                digitalWrite(LedMediaOut, LOW);
                LEDstate2=1;
                Mstate2=1;
                break;
            case 1:
                digitalWrite(LedMediaIn, LOW);
                digitalWrite(MEDIAin, LOW);
                LEDstate2=0;
                Mstate2=0;
                break;
            }
        }
        last_button_time2 = button_time2;
    }
}

/***************************************************************************/
**                                Real Time Clock (RTC)                      **
***************************************************************************/
//long currentMillis4 = 0;        // will store last time LED was updated
//long interval4 = 5000;           // interval at which to blink (milliseconds)
//unsigned long previousMillis4=0;

/*
void Time(){

    //unsigned long currentMillis4 = millis();
    //if(currentMillis4 - previousMillis4 > interval4) //
    //  {
    //    DateTime now = RTC.now();
    //    //lcd.setCursor(5,0);
    //    Serial.print(now.year(), DEC);
    //    Serial.print('/');
    //    if(now.month() < 10) Serial.print("0");

*/
Serial.print(now.month(), DEC);
Serial.print('/');
if(now.day() < 10) Serial.print("0");
Serial.print(now.day(), DEC);
Serial.print(' ');
if(now.hour() < 10) Serial.print("0");
Serial.print(now.hour(), DEC);
Serial.print(':');
if(now.minute() < 10) Serial.print("0");
Serial.print(now.minute(), DEC);
Serial.print(':');
if(now.second() < 10) Serial.print("0");
Serial.print(now.second(), DEC);
Serial.println();

//Serial.print(now.tempC(), 1);
//Serial.println(" degrees Celcius");
//Serial.print(now.tempF(), DEC);
//Serial.println(" degrees Farenheit");
//Serial.println();
// previousMillis4 = currentMillis4;
// }
}
*/

/**************************************************************************
**                                MENU mini functions
**************************************************************************/

void incTemp(){
    // Increase temp.
    setTemp += 0.1;
    if (setTemp > 40) setTemp = 40;
    updateDisplay1();
saveTemp();
}

void DefTemp(){
    setTemp=37.0;
saveTemp();
}

void DefCO2(){
    setCO2=5.0;
saveCO2();
}

void decTemp(){
    // Increase temp.
    setTemp -= 0.1;
if (setTemp < 25) setTemp = 25;
updateDisplay1();
saveTemp();
}

void incCO2(){
  // Increase temp.
  setCO2 += 0.1;
  if (setCO2 > 10) setCO2 = 10;
  updateDisplay2();
saveCO2();
}

void decCO2(){
  // Increase temp.
  setCO2 -= 0.1;
  if (setCO2 < 0) setTemp = 0;
  updateDisplay2();
saveCO2();
}

void run_Menu(){
  read_buttons();
basicMenu();
}

void clearDisplay()
{
  lcd.write(0xFE);  // send the special command
  lcd.write(0x01);  // send the clear screen command
}

void saveTemp()
{
  // Save the temperature into the EEPROM
  byte gTemp = setTemp * 2;
  EEPROM.write(TEMPR_STORE, gTemp);
}

void saveCO2()
{
  // Save the temperature into the EEPROM
  byte gCO2 = setCO2 * 2;
  EEPROM.write(CO2_STORE, gCO2);
}

void updateDisplay1()
{
  lcd.setCursor(14,1);
  lcd.print(setTemp,2);
}
void updateDisplay2()
{
  lcd.setCursor(14,1);
  lcd.print(setCO2,2);
}

void updateDisplay3()
{
  lcd.setCursor(14,2);
  lcd.print(CurrTemp,2);
}

void countdown()
{
  static unsigned long lastTick = 0; // set up a local variable to hold the last time we decremented one second
  // (static variables are initialized once and keep their values between function calls)

  // decrement one second every 1000 milliseconds
  if (second > 0) {
    if (millis() - lastTick >= 1000) {
      lastTick = millis();
      second--;
      TimerOutput();
    }
  }

  // decrement one minute every 60 seconds
  if (minute > 0) {
    if (second <= 0) {
      minute--;
      second = 59; // reset seconds to 59
    }
  }

  // decrement one hour every 60 minutes
  if (hour > 0) {
    if (minute <= 0) {
      hour--;
      minute = 59; // reset minutes to 59
    }
  }

  // close countdown();

  void TimerOutput()
  {
    //clearLCD();
    //Print time on each line
    delay(100);
    //lcd.setCursor(0,0);
    //lcd.print("Time: ");
lcd.setCursor(12,3);
lcd.print(hour, DEC); // the hour, sent to the screen in decimal format
lcd.print(" "); // a colon between the hour and the minute
lcd.print(minute, DEC); // the minute, sent to the screen in decimal format
lcd.print(" "); // a colon between the minute and the second
if (second >=10){
lcd.print(second, DEC); // the second, sent to the screen in decimal format
}
if (second <10){
lcd.setCursor(16,3);
lcd.print(0);
lcd.setCursor(17,3);
lcd.print(second, DEC);
}
// termination condition
if (second == 0 && minute == 0 && hour == 0) {
  lcd.setCursor(12,3);
lcd.print("**Ready**");
}
}

byte read_buttons(){ // you may need to swap "void" with "int" or "byte"
  byte returndata = 0;
  int buttonState; // this might not be the correct declaration.
  // remember to declare what buttonUp, buttonDown, buttonSelect, buttonCancel are
  if ((lastButtonPressed + debounceTime) < millis()){ // see if it's time to check the buttons again
    buttonState = digitalRead(buttonUp);
    if (buttonState == HIGH){
      returndata = returndata + 1;
      lastButtonPressed = millis();
    }
  }
  buttonState = digitalRead(buttonDown);
  if (buttonState == HIGH){
    returndata = returndata + 2;
    lastButtonPressed = millis();
  }
  buttonState = digitalRead(buttonSelect);
  if (buttonState == HIGH){
    returndata = returndata + 4;
    lastButtonPressed = millis();
  }
  return returndata; // this spits back to the function that calls it the variable returndata.
}

/*
 ***************************************************
 * Menu function
 ***************************************************/
void basicMenu() {

    byte topItemDisplayed = 0;  // stores menu item displayed at top of LCD screen
    byte cursorPosition = 0;  // where cursor is on screen, from 0 --> totalRows.

    // redraw = 0 - don't redraw
    // redraw = 1 - redraw cursor
    // redraw = 2 - redraw list
    byte redraw = MOVELIST;  // triggers whether menu is redrawn after cursor move.
    byte i=0; // temp variable for loops.
    byte totalMenuItems = 0; // a while loop below will set this to the # of menu items.

    // Put the menu items here. Remember, the first item will have a 'position' of 0.
    char* menuItems[]=
    {   
        "Main Menu Options",
        "Set Temp (C)",
        "Set CO2 (%)",
        "Default Temp/CO2",
        "PreHeat WaterBath",
        "Home",
        "",
    };

    while (menuItems[totalMenuItems] != ""){
        totalMenuItems++;  // count how many items are in list.
    }
    totalMenuItems--;  // subtract 1 so we know total items in array.

    clearDisplay();  // clear the screen so we can paint the menu.

    stillSelecting = true;  // set because user is still selecting.

    timeoutTime = millis() + menuTimeout; // set initial timeout limit.

    do  // loop while waiting for user to select.
    {

        switch(read_buttons())

        {  // analyze encoder response. Default is 0.

            case 1:  // 'UP' BUTTON PushED

                timeoutTime = millis()+menuTimeout;  // reset timeout timer
                // if cursor is at top and menu is NOT at top
                // move menu up one.
                if(cursorPosition == 0 && topItemDisplayed > 0 && scroll==true) // Cursor is at top of LCD, and there are higher menu items still to be displayed.
                {
                    topItemDisplayed--;  // move top menu item displayed up one.
                    redraw = MOVELIST;  // redraw the entire menu
                }

        } // case 1

    } // do while
/if cursor not at top, move it up one.
if(cursorPosition>0 && scroll==true)
{
    cursorPosition--; // move cursor up one.
    redraw = MOVECURSOR; // redraw just cursor.
}
break;

case 2: // ENCODER ROTATED UP. EQUIVALENT OF 'DOWN' BUTTON PUSHED

timeoutTime = millis()+menuTimeout; // reset timeout timer
// this sees if there are menu items below the bottom of the LCD screen & sees if cursor is at
// bottom of LCD
if((topItemDisplayed + (totalRows-1)) < totalMenuItems && cursorPosition == (totalRows-1) &&
    scroll==true) // cursor is not at bottom of LCD, so move it down
    { // cursor is not at bottom of LCD, so move it down
    topItemDisplayed++; // move menu down one
    redraw = MOVELIST; // redraw entire menu
}
if(cursorPosition<(totalRows-1)&& scroll==true)  // cursor is not at bottom of LCD, so move it down
    { // cursor is not at bottom of LCD, so move it down
    cursorPosition++; // move cursor down one
    redraw = MOVECURSOR; // redraw just cursor.
}
break;

case 4: // ENCODER BUTTON PUSHED FOR SHORT PERIOD & RELEASED.
// EQUIVALENT TO 'SELECT' OR 'OKAY' BEING PUSHED

timeoutTime = millis()+menuTimeout; // reset timeout timer
switch(topItemDisplayed + cursorPosition) // adding these values together = where on menuItems
cursor is.
{
    // put code to be run when specific item is selected in place of the lcd.print filler.
    // the lcd.print code can be removed, but DO NOT change the case & break structure.
    // (Obviously, you should have as many case instances as you do menu items.)
    case 0: // menu item 1 selected
        basicMenu();
        break;
    case 1: // menu item 1 selected
        SubMenu1(); //Temp SubMenu
        break;
    case 2: // menu item 2 selected
        SubMenu2(); //CO2 SubMenu
        break;
    case 3: // menu item 3 selected
        DefTemp();
        break;
}
DefCO2();
break;

case 4: // menu item 4 selected
SubMenu3();
break;

case 5: // menu item 5 selected
MenuHome();
break;
case 6: // encoder button was pushed for long time. This corresponds to "Back" or "Cancel" being pushed.
stillSelecting = false;
break;
}

switch(redraw){  // checks if menu should be redrawn at all.
case MOVECURSOR: // Only the cursor needs to be moved.
redraw = false; // reset flag.
if (cursorPosition > totalMenuItems) // keeps cursor from moving beyond menu items.
cursorPosition = totalMenuItems;
for(i = 0; i < (totalRows); i++){ // loop through all of the lines on the LCD
lcd.setCursor(0,i);
lcd.print(" "); // and erase the previously displayed cursor
lcd.setCursor((totalCols-1), i);
lcd.print(" ");
}
lcd.setCursor(0,cursorPosition); // go to LCD line where new cursor should be & display it.
lcd.print(">");
lcd.setCursor((totalCols-1), cursorPosition);
//lcd.print("<");
break; // MOVECURSOR break.

case MOVELIST: // the entire menu needs to be redrawn
redraw = MOVECURSOR; // redraw cursor after clearing LCD and printing menu.
lcd.clear(); // clear screen so it can be repainted.
if(totalMenuitems>((totalRows-1))){ // if there are more menu items than LCD rows, then cycle through menu items.
for (i = 0; i < (totalRows); i++){
lcd.setCursor(1,i);
lcd.print(menuItems[topItemDisplayed + i]);
}
}
else{ // if menu has less items than LCD rows, display all available menu items.
for (i = 0; i < totalMenuitems+1; i++){
lcd.setCursor(1,i);
lcd.print(menuItems[topItemDisplayed + i]);
}
}
break; // MOVELIST break}
if (timeoutTime < millis()) {  // user hasn't done anything in awhile
stillSelecting = false;  // tell loop to bail out.
/*
in my main code, I had a function that
displayed a default screen on the LCD, so
I would put that function here, and it would
bail out to the default screen.
defaultScreen();
*/
}
}
while (stillSelecting == true);  //
}

;/*
******************************
************ Menu Routine
*******************************/

void SubMenu1(){

byte topItemDisplayed = 0;  // stores menu item displayed at top of LCD screen
byte cursorPosition = 0;  // where cursor is on screen, from 0 --> totalRows.

// redraw = 0   - don't redraw
// redraw = 1   - redraw cursor
// redraw = 2   - redraw list
byte redraw = MOVELIST;  // triggers whether menu is redrawn after cursor move.
byte i=0;  // temp variable for loops.
byte totalMenuItems = 0;  // a while loop below will set this to the # of menu items.

// Put the menu items here. Remember, the first item will have a 'position' of 0.
char* SubmenuItems1[]={
"---MODIFY TEMP.---",
"Set Temp (C):",
"+",
"-",
"Return",
""
};

while (SubmenuItems1[totalMenuItems] != ""){
totalMenuItems++;  // count how many items are in list.
}
totalMenuItems--;  // subtract 1 so we know total items in array.

clearDisplay();  // clear the screen so we can paint the menu.

stillSelecting = true;  // set because user is still selecting.

timeoutTime = millis() + menuTimeout;  // set initial timeout limit.
do  // loop while waiting for user to select.
{

switch(read_buttons())
{
  // analyze encoder response. Default is 0.

  case 1:  // 'UP' BUTTON PUSHED
    timeoutTime = millis()+menuTimeout;  // reset timeout timer
    // if cursor is at top and menu is NOT at top
    // move menu up one.
    if(cursorPosition == 0 && topItemDisplayed > 0 && scroll==true)  // Cursor is at top of LCD, and there are higher menu items still to be displayed.
    {
      topItemDisplayed--;  // move top menu item displayed up one.
      redraw = MOVELIST;  // redraw the entire menu
    }

    // if cursor not at top, move it up one.
    if(cursorPosition>0 && scroll==true)
    {
      cursorPosition--;  // move cursor up one.
      redraw = MOVECURSOR;  // redraw just cursor.
    }
    break;

  case 2:  // ENCODER ROTATED UP. EQUIVALENT OF 'DOWN' BUTTON PUSHED
    timeoutTime = millis()+menuTimeout;  // reset timeout timer
    // this sees if there are menu items below the bottom of the LCD screen & sees if cursor is at bottom of LCD
    if((topItemDisplayed + (totalRows-1)) < totalMenuItems && cursorPosition == (totalRows-1) &&
      scroll==true)
    {
      topItemDisplayed++;  // move menu down one
      redraw = MOVELIST;  // redraw entire menu
    }
    if(cursorPosition<(totalRows-1)&& scroll==true)  // cursor is not at bottom of LCD, so move it down one.
    {
      cursorPosition++;  // move cursor down one
      redraw = MOVECURSOR;  // redraw just cursor.
    }
    break;

  case 4:  // ENCODER BUTTON PUSHED FOR SHORT PERIOD & RELEASED.
    // EQUIVALENT TO 'SELECT' OR 'OKAY' BEING PUSHED
    timeoutTime = millis()+menuTimeout;  // reset timeout timer
    switch(topItemDisplayed + cursorPosition) // adding these values together = where on menuitems cursor is.
    {
// put code to be run when specific item is selected in place of the lcd.print filler.
// the lcd.print code can be removed, but DO NOT change the case & break structure.
// (Obviously, you should have as many case instances as you do menu items.)
case 0:  // menu item 1 selected
    SubMenu1();
    updateDisplay1();
    break;
case 1:  // menu item 1 selected
    updateDisplay1();
    break;
case 2:  // menu item 2 selected
    incTemp();
    break;
case 3:  // menu item 3 selected
    decTemp();
    break;
case 4:  // menu item 4 selected
    basicMenu();
    break;
case 5:  // encoder button was pushed for long time. This corresponds to "Back" or "Cancel" being pushed.
    stillSelecting = false;
    lcd.println("Button held for a long time");
    break;
}

switch(redraw){  // checks if menu should be redrawn at all.
case MOVECURSOR:  // Only the cursor needs to be moved.
    redraw = false;  // reset flag.
    if (cursorPosition > totalMenuItems) // keeps cursor from moving beyond menu items.
    cursorPosition = totalMenuItems;
    for(i = 0; i < (totalRows); i++)// loop through all of the lines on the LCD
        lcd.setCursor(0,i);
        lcd.print(" ");  // and erase the previously displayed cursor
        lcd.setCursor((totalCols-1), i);
        lcd.print(" ");
    }
lcd.setCursor(0,cursorPosition);  // go to LCD line where new cursor should be & display it.
lcd.println(" ");
lcd.setCursor((totalCols-1), cursorPosition);
//lcd.print(" ");
break;  // MOVECURSOR break.

case MOVELIST:  // the entire menu needs to be redrawn
redraw=MOVECURSOR;  // redraw cursor after clearing LCD and printing menu.
lcd.clear();  // clear screen so it can be repainted.
if(totalMenuItems>=(totalRows-1)){  // if there are more menu items than LCD rows, then cycle
    for (i = 0; i < (totalRows); i++){  
        lcd.setCursor(1,i);
        lcd.print(SubmenuItems1[topItemDisplayed + i]);
    }
} else{  // if menu has less items than LCD rows, display all available menu items.
    for (i = 0; i < totalMenuItems+1; i++){
        lcd.setCursor(1,i);
        lcd.print(SubmenuItems1[topItemDisplayed + i]);
    }
} else{  // if menu has less items than LCD rows, display all available menu items.
    for (i = 0; i < totalMenuItems+1; i++){
        lcd.setCursor(1,i);
        lcd.print(SubmenuItems1[topItemDisplayed + i]);
    }
} break;  // MOVELIST break

if (timeoutTime<millis()){  // user hasn't done anything in awhile
    stillSelecting = false;  // tell loop to bail out.
    /*
in my main code, I had a function that
displayed a default screen on the LCD, so
I would put that function here, and it would
bail out to the default screen.
defaultScreen();
*/
} while (stillSelecting == true);  //

void SubMenu2(){

    byte topItemDisplayed = 0;  // stores menu item displayed at top of LCD screen
    byte cursorPosition = 0;  // where cursor is on screen, from 0 --> totalRows.

    // redraw = 0  - don't redraw
    // redraw = 1  - redraw cursor
    // redraw = 2  - redraw list
    byte redraw = MOVELIST;  // triggers whether menu is redrawn after cursor move.
    byte i=0;  // temp variable for loops.
    byte totalMenuItems = 0;  //a while loop below will set this to the # of menu items.

    // Put the menu items here. Remember, the first item will have a 'position' of 0.
    char* SubmenuItems2[]={
        "---MODIFY CO2---",
        "Set CO2 (%):",
        "+",
        "-",
        "Return",
        "",
    };
while (SubmenuItems2[totalMenuItems] != ""){
  totalMenuItems++;  // count how many items are in list.
}  

totalMenuItems--;  // subtract 1 so we know total items in array.

clearDisplay();  // clear the screen so we can paint the menu.

stillSelecting = true;  // set because user is still selecting.

timeoutTime = millis() + menuTimeout;  // set initial timeout limit.

do  // loop while waiting for user to select.
{
  switch(read_buttons())
  {
    // analyze encoder response. Default is 0.

    case 1:  // 'UP' BUTTON PUSHED
     (timeoutTime = millis()+menuTimeout;  // reset timeout timer
      // if cursor is at top and menu is NOT at top
      // move menu up one.
      if(cursorPosition == 0 && topItemDisplayed > 0 && scroll==true)  // Cursor is at top of LCD, and
      // there are higher menu items still to be displayed.
      {
        topItemDisplayed--;  // move top menu item displayed up one.
        redraw = MOVELIST;  // redraw the entire menu
      }

      // if cursor not at top, move it up one.
      if(cursorPosition>0 && scroll==true)
      {
        cursorPosition--;  // move cursor up one.
        redraw = MOVECURSOR;  // redraw just cursor.
      }
    break;

    case 2:    // ENCODER ROTATED UP. EQUIVALENT OF 'DOWN' BUTTON PUSHED
      (timeoutTime = millis()+menuTimeout;  // reset timeout timer
      // this sees if there are menu items below the bottom of the LCD screen & sees if cursor is at
      // bottom of LCD
      if((topItemDisplayed + (totalRows-1)) < totalMenuItems && cursorPosition == (totalRows-1) &&
      scroll==true)
      {
        topItemDisplayed++;  // move menu down one
        redraw = MOVELIST;  // redraw entire menu
      }
    if(cursorPosition<(totalRows-1) && scroll==true)  // cursor is not at bottom of LCD, so move it down
    one.
    {
      }
cursorPosition++;  // move cursor down one
    redraw = MOVECURSOR;  // redraw just cursor.
}
break;

case 4:  // ENCODER BUTTON PUSHED FOR SHORT PERIOD & RELEASED.
    // EQUIVALENT TO 'SELECT' OR 'OKAY' BEING PUSHED
    timeoutTime = millis()+menuTimeout;  // reset timeout timer
    switch(topItemDisplayed + cursorPosition) // adding these values together = where on menuItems
cursor is.
{
    // put code to be run when specific item is selected in place of the lcd.print filler.
    // the lcd.print code can be removed, but DO NOT change the case & break structure.
    // (Obviously, you should have as many case instances as you do menu items.)
    case 0:  // menu item 1 selected
       SubMenu2();
        updateDisplay2();
        break;
    case 1:  // menu item 1 selected
        updateDisplay2();
        break;
    case 2:  // menu item 2 selected
        incCO2();
        break;
    case 3:  // menu item 3 selected
        decCO2();
        break;
    case 4:  // menu item 4 selected
        basicMenu();
        break;
    case 5:  // encoder button was pushed for long time. This corresponds to "Back" or "Cancel" being
    pushed.
        stillSelecting = false;
        //lcd.println("Button held for a long time");
        break;
}

switch(redraw){  // checks if menu should be redrawn at all.
case MOVECURSOR:  // Only the cursor needs to be moved.
    redraw = false;  // reset flag.
    if (cursorPosition > totalMenuItems) // keeps cursor from moving beyond menu items.
        cursorPosition = totalMenuItems;
    for(i = 0; i < totalRows; i++)// loop through all of the lines on the LCD
        lcd.setCursor(0,i);
    lcd.print(" ");  // and erase the previously displayed cursor
```cpp
lcd.setCursor((totalCols-1), i);
lcd.print(" ");
}
lcd.setCursor(0,cursorPosition);  // go to LCD line where new cursor should be & display it.
lcd.print(">");
lcd.setCursor((totalCols-1), cursorPosition);
//lcd.print("<");
break;  // MOVECURSOR break.

case MOVELIST:  // the entire menu needs to be redrawn
    redraw=MOVECURSOR;  // redraw cursor after clearing LCD and printing menu.
lcd.clear();  // clear screen so it can be repainted.
    if(totalMenuItems>((totalRows-1))){  // if there are more menu items than LCD rows, then cycle through menu items.
        for (i = 0; i < (totalRows); i++){
            lcd.setCursor(1,i);
            lcd.print(SubmenuItems2[topItemDisplayed + i]);
        }
    }
    else{  // if menu has less items than LCD rows, display all available menu items.
        for (i = 0; i < totalMenuItems+1; i++){
            lcd.setCursor(1,i);
            lcd.print(SubmenuItems2[topItemDisplayed + i]);
        }
    }
    break;  // MOVELIST break
}

if (timeoutTime<millis()){  // user hasn't done anything in awhile
    stillSelecting = false;  // tell loop to bail out.
    /*
in my main code, I had a function that
    displayed a default screen on the LCD, so
    I would put that function here, and it would
    bail out to the default screen.
defaultScreen();
    */
}
while (stillSelecting == true);  //

void SubMenu3(){

    byte topItemDisplayed = 0;  // stores menu item displayed at top of LCD screen
    byte cursorPosition = 0;  // where cursor is on screen, from 0 --> totalRows.

    // redraw = 0  - don't redraw
    // redraw = 1  - redraw cursor
    // redraw = 2  - redraw list
    byte redraw = MOVECURSOR;  // triggers whether menu is redrawn after cursor move.
    byte i=0;  // temp variable for loops.
    byte totalMenuItems = 0;  //a while loop below will set this to the # of menu items.
```
// Put the menu items here. Remember, the first item will have a 'position' of 0.
char* SubmenuItems3[] = {
    "PreHeat WaterBath",
    "Set Temp (C):",
    "Curr.Temp(C):",
    "T Remain:",
    "Return",
    ""
};

while (SubmenuItems3[totalMenuItems] != ""){
    totalMenuItems++; // count how many items are in list.
}
totalMenuItems--; // subtract 1 so we know total items in array.

clearDisplay(); // clear the screen so we can paint the menu.

stillSelecting = true; // set because user is still selecting.
timeoutTime = millis() + menuTimeout; // set initial timeout limit.

do // loop while waiting for user to select.
{
    switch(read_buttons())
    {
        // analyze encoder response. Default is 0.

        case 1: // 'UP' BUTTON PUSHED
            timeoutTime = millis()+menuTimeout; // reset timeout timer
            // if cursor is at top and menu is NOT at top
            // move menu up one.
            if(cursorPosition == 0 && topItemDisplayed > 0 && scroll==true) // Cursor is at top of LCD, and there are higher menu items still to be displayed.
            {
                topItemDisplayed--; // move top menu item displayed up one.
                redraw = MOVELIST; // redraw the entire menu
            }
            // if cursor not at top, move it up one.
            if(cursorPosition>0 && scroll==true)
            {
                cursorPosition--; // move cursor up one.
                redraw = MOVECURSOR; // redraw just cursor.
            }
            break;

        case 2: // ENCODER ROTATED UP. EQUIVALENT OF 'DOWN' BUTTON PUSHED
            timeoutTime = millis()+menuTimeout; // reset timeout timer
            // this sees if there are menu items below the bottom of the LCD screen & sees if cursor is at
bottom of LCD
  if((topItemDisplayed + (totalRows -1)) < totalMenuItems && cursorPosition == (totalRows -1) && scroll==true) {
    topItemDisplayed++; // move menu down one
    redraw = MOVELIST; // redraw entire menu
  }
  if(cursorPosition<(totalRows-1)&& scroll=true) // cursor is not at bottom of LCD, so move it down one.
  {
    cursorPosition++; // move cursor down one
    redraw = MOVECURSOR; // redraw just cursor.
  }
  break;

  case 4: // ENCODER BUTTON PUSHED FOR SHORT PERIOD & RELEASED.
     // EQUIVALENT TO 'SELECT' OR 'OKAY' BEING PUSHED
    timeoutTime = millis()+menuTimeout; // reset timeout timer
    switch(topItemDisplayed + cursorPosition) // adding these values together = where on menuItems
cursor is.
    {
      // put code to be run when specific item is selected in place of the lcd.print filler.
      // the lcd.print code can be removed, but DO NOT change the case & break structure.
      // (Obviously, you should have as many case instances as you do menu items.)
      case 0: // menu item 1 selected
        SubMenu3();
        break;

      case 1: // menu item 1 selected
        updateDisplay1();
        break;

      case 2: // menu item 2 selected
        updateDisplay3();
        break;

      case 3: // menu item 3 selected
        countdown();
        TimerOutput();
        break;

      case 4: // menu item 4 selected
        basicMenu();
        break;

      case 5: // encoder button was pushed for long time. This corresponds to "Back" or "Cancel" being pushed.
        stillSelecting = false;
        //lcd.println("Button held for a long time");
        break;
switch(redraw){  // checks if menu should be redrawn at all.
case MOVECURSOR: // Only the cursor needs to be moved.
  redraw = false;  // reset flag.
  if (cursorPosition > totalMenuItems) // keeps cursor from moving beyond menu items.
    cursorPosition = totalMenuItems;
  for(i = 0; i < (totalRows); i++) {  // loop through all of the lines on the LCD
    lcd.setCursor(0,i);
    lcd.print(" ");                      // and erase the previously displayed cursor
    lcd.setCursor((totalCols-1), i);
    lcd.print(" ");
  }
  lcd.setCursor(0,cursorPosition);      // go to LCD line where new cursor should be & display it.
  lcd.print(" ");
  lcd.setCursor((totalCols-1), cursorPosition);
  //lcd.print("<");
  break;  // MOVECURSOR break.

  case MOVELIST:  // the entire menu needs to be redrawn
    redraw=MOVECURSOR;  // redraw cursor after clearing LCD and printing menu.
    lcd.clear(); // clear screen so it can be repainted.
    if (totalMenuItems > (totalRows-1)) {  // if there are more menu items than LCD rows, then cycle through menu items.
      for (i = 0; i < (totalRows); i++){
        lcd.setCursor(1,i);
        lcd.print(SubmenuItems3[topItemDisplayed + i]);
      }
    } else{  // if menu has less items than LCD rows, display all available menu items.
      for (i = 0; i < totalMenuItems+1; i++){
        lcd.setCursor(1,i);
        lcd.print(SubmenuItems3[topItemDisplayed + i]);
      }
    }
    break;  // MOVELIST break
  }
  if (timeoutTime < millis()) { // user hasn't done anything in awhile
    stillSelecting = false;  // tell loop to bail out.
    /*
    in my main code, I had a function that
    displayed a default screen on the LCD, so
    I would put that function here, and it would
    bail out to the default screen.
    defaultScreen();
    */
    }
  while (stillSelecting == true);  //
}