Wireless Home Control System

Grant Hernandez, Jimmy Campbell and Joseph Love

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.**

***Keywords*-Home automation, mobile, scalable platform, sensors, power control, wireless.**

1. Introduction

The Wireless Home Control System is meant to be a solution for any homeowner to be able to remotely control core appliances of their home. The WHCS vision is to allow the user to control lights, outlets, doors, and sensors around their home. The system's design philosophy emphasizes ease of use, affordability, and effectiveness. An Android phone application developed for WHCS will allow users to monitor the state of the installed components and activate them remotely. A central base station equipped with a touch-enabled LCD will be present to allow the users to interact with the system without the need of a phone. Peripheral control modules will be installed into the targeted appliances such as lights, outlets, and doors for WHCS to control. Together the phone, base station, and peripheral control modules will constitute WHCS.

The implementation of such a system requires research in a myriad of fields to produce a full-fledged product. Android software development is the core for creating the users first impression with WHCS. Research must be conducted to conform to the design philosophies of the Android ecosystem. The alternative interface offered for WHCS will be the base station's display, as a result the best solution for a touch capable LCD will be examined. Communication devices will form the foundation for the wireless aspect of WHCS, thus an investigation into the advantages of different communication modules is required to realize the system. A network protocol will need to be planned out and implemented in order to form a unified system from the independent modules. The activation of appliances around the home requires high voltage control, so methodologies for properly harnessing the power provided by homes are a requirement. To extend upon harnessing the home's power, our individual control modules and base station's logic level voltage depend upon the creation of an efficient way to step down the high voltage supplied to the home. Efficient forms of rectification and power provision will need to be researched for WHCS to be self-sufficient. Important research and critical design decisions will go into the development of the printed circuit boards that will provide the foundation for all the hardware of WHCS.

Wireless Home Control System is a solution targeting the masses and designed by few. Naturally such a system will suffer from the constraints imposed upon the creators. Most prominent of all constraints are those stemming from economics. The development and production of WHCS will have to conform to the low budget available. Design decisions will be made to minimize overall cost of the system to satisfy this constraint. WHCS has the potential for mass implementation if user reception is positive so the design will adhere to manufacturing constraints. The parts used in the system will be chosen so that they are widely available. Our boards and parts will be designed so that they are easy to replicate and manufacture. With a product such as WHCS health and safety is clearly an issue. The system is meant to be installed inside the home where the user should feel at ease with the system installed. Thus it is ethical for us to put effort into making the system safe to use. Things such as controlling the home's high voltage will have to be done in a safe manner. Security is another strong constraint attributed to products targeting the home. The system will be designed to maximize security for our users.

2. Base Station

The heart of WHCS resides with the base station (BS). When users think of WHCS, they will think of the base station as it is the most visible part of the system. The base station is responsible for managing, collecting, and displaying information from all of the control modules. If the BS were to fail, WHCS would cease to function.

When a new control module is introduced in to the system, it must first pair with the BS, which will authenticate it on to the network. Once it joins, the control module can be abstracted as an "API" meaning, the specific hardware details do not have to be considered. This abstraction will be excellent in keeping the system clean from one-off cases and allow for a Domain Specific API to be formed.

1) Software Flow: The main software flow for the base station is the most complicated in WHCS. It has to manage three separate devices simulatenously and be able to service each one in a timely manner. The LCD, NRF radio, and Bluetooth module are all being controlled and commanded by one ATMega32-A chip. There isn't much room for busy waiting or any expensive operations as everything has to be running as fast as possible. Given this, the BS is the least point of failure for the WHCS.

In Figure 1, we have constructed the general architecure of the main loop for the BS, including some early initialization. Most of the implementation details are left out as they are very specific to the final drivers.

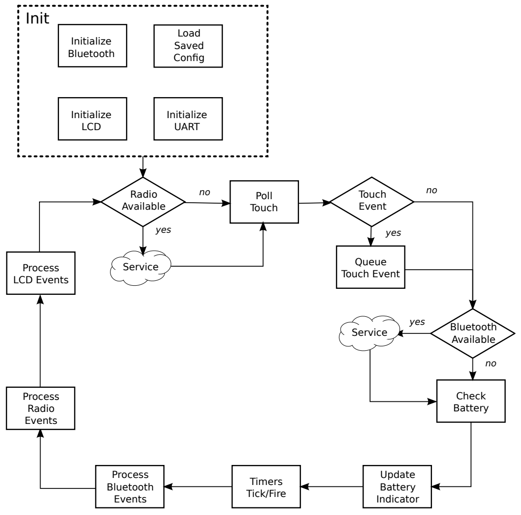


Fig. 1: the high level software flow for the base station

The large amount of tasks that are required of the BS become clear when viewing Figure 1.

The base station from reset would first load any saved settings from the EEPROM (saved control modules, behavior settings, LCD settings, etc.), then it would bring up all of the main modules (radio, LCD, BlueTooth, and UART). If any of these initialization steps were to fail, the BS would indicate through an LED or from a UART debug message. An initialization failure shouldn't be handled as it's a critical failure of the system along with its assumptions. We believe that by assuring that the initialization sequences for all modules is well defined, we will be able to diagnose hardware issues quickly and without strenuous debugging. One of the issues we found while prototyping is that it's difficult to determine if the issue with module initialization is with the wiring or code. Having a golden model for code that is know to work on our target setup would allow us to have a good working state.

Each module has a unique sequence of "commands" with parameters that are required to configure the device. The NRF radio has to have its power, channel, payload length, and other parameters set before usage. Once these basic options, any further configuration is done at run time. This would include switching from listening to transmitting mode and enabling or disabling the automatic acknowledgement feature. The built-in Atmel UART only needs to know the baud rate at which data will be sent and received. There are also other options relating to other bits used (parody, stop, etc.). In our case, we are using the default 8 bit data, 1 stop bit, which is simple enough for our needs. The LCD happens to have the most extensive initialization sequence due to required screen configuration, gamma settings, pixel order and other more archaic options.

Once all of the modules are brought up correctly, the BS would begin the main loop. Radio events, BlueTooth events, and LCD state would all be processed as required. For both the BlueTooth and radio, new packets would be checked for and serviced as needed. From these packets, internal state would updated and any response packets would be generated and sent. Internally, WHCS will have an internal event queue with different event types. These events will be processed and any responses will be generated, if any. These responses could include a confirmation packet over the NRF radio, or a status update through BlueTooth. The base station may also initialize actions despite not receiving events. These could be triggered from timers firing.

In terms of the NRF radio, the module we have chosen exports an interrupt pin which will fire on the reception of a new packet. This feature is great for the WHCS architecture as the radio requires a significant amount of power while actively receiving SPI commands and transmitting. By avoiding the polling of the NRF, the main loops for both the control module and base station will have more cycles to process more important and intensive tasks. Essentially, the NRF will be kept in the listening mode at all times, waiting for a packet from a control module to be received. This is in contrast to the control modules which will try to avoid the transmitting and listening states to save power. The power down mode is great for saving watts, but has very poor performance for quickly responding to new packets.

The BlueTooth module won't require as much time to service as the other modules due to the low data rate. It will be connected directly to the hardware UART of the base station, instead of the normal serial debugger. This will have to be connected to some free GPIO ports and software serial will be used. This is unfortunate as "bit banging" serial isn't efficient and will take up many more cycles than just using the hardware UART. If needed, a packet over BlueTooth could be sent asynchronously from the main loop due to the hardware UART. If the micro loops fast enough, then it would be as if there was no delay in transmission. Due to the real time nature of the NRF radio and LCD, WHCS will take steps to avoid blocking too long on a single activity. Blocking too much will lower the overall performance and responsiveness of the system. At worst, touch events could be lost and packet buffers could overflow. Until the system's code is ready, this worry cannot be confirmed, but an effort will be made to profile the main loop's average time using hardware timers.

One of the most important parts of WHCS is its usage of timers. A global list of timers will be constantly maintained and updated to schedule events in the future, instead of having to handle them immediately. The granularity of the timers will depend on the average execution time of the main loop along with how many timers are processed at once. If the main loop is slow on average, then timers will have to wait for extended periods to be serviced (starvation). Depending on the criticality of the timer and the event associated with it, there may need to be a mechanism for assigning a priority for a timer. This will have to be determined during actual system programming as it is heavily dependent on the task required.

All of the above tasks will be executing in the same way as single core CPU would: in pseudo-parallel. The faster the whole system runs, the better the appearance of everything executing at once.

2) Control Module Abstraction: For WHCS to function smoothly and scale well, a neat and abstracted interface must be defined to accept *any* type of control module. New control module types should be easily added to the system without affecting older types and there should be a set of generic data structures for managing and storing information on modules. These structures must be carefully defined to wrap more specific control module packets in all of the shared metadata. Think of it like a hierarchy where all of the common attributes and actions shared by control modules have packets that can be sent to any module. Whereas the more specific packets (get temperature, engage door, etc.) would be wrapped up in the generic ones (essentially a derived object from the generic control module.) This can be visualized in Figure 2. The details of a network structure that would enable this clean interface is further described in **??**.

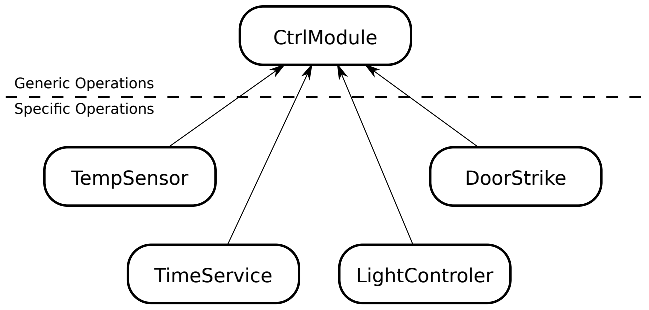


Fig. 2: showing the control module hierarchy for WHCS

Beyond sending packets, the base station must accurately record and update state for all of the control modules. Depending on the control module, additional state will need to be stored and functions will need to be written to query that state. Each control module would have its own state machine that would control its function in relation to the base station.

3) Subsystems: The base station has the hardest job in the entire WHCS architecture. It has to juggle a lot of data with limited memory and processing speed. Packets need to be processed and queued to keep the pipeline flowing smoothly. The following sections will break down the individual subsystems and how they work in concert to make the base station a well oiled microcontroller.

NRF24L01+: The NRF radio will be directly connected to the Atmega32-A microcontroller which will control its state. This module will be constantly listening for new packets from the control modules and sending responses in turn. Due to this link being the most critical for WHCS, it needs to have the most attention to detail when constructing the layout and software design. Also, besides the expensive drawing operations for the LCD, this may take up the most CPU time to process packets and perform data transfers. This is partly due to the slow SPI interface that will be used to transfer the data. the NRF24LO1+ breakout board does not offer any other options for transferring data to and from a microcontroller. The overall system speed will be limited by the maximum transfer speed of this radio. When the considerations for thw WHCS topology are brought forward, its is obvious that the organization relies solely on the speed of the base station. We have considered this fact throughout the system design - for example, the LCD could have *also* been controlled over SPI, but we made a trade off for pin count in order to use the parallel interface. This frees up the SPI bus for programming (infrequent) and the NRF. Essentially, the NRF has full control over the SPI bus and will receive the full attention of the microcontroller.

Assuming the NRF is not limited by the transfer rate of the SPI bus, the additional considerations can now be focused on. The microcontroller needs a way to quickly assess the state of the NRF chip. This state query could include the status of the radio's transfer queues - meaning if there is a packet to be received or if a packet has been successfully sent. This is something that will need to be checked very frequently as the radio usage will increase linearlly with the amount of control modules that are apart of WHCS. If any algorithms or tight loops in the microcontroller were to **not** have linear performance, then the system speed would suffer as more control modules were brought in to the network. We are focusing on WHCS' scalability for more complex households that a small demo setup. Of course, once the network becomes too crowded, the somewhat weak Atmega32-A will no longer be able to sustain the flood of packets that are required to maintain WHCS. At this point, more base stations will need to be operating simultaneously in order to handle the load. Our implementation of WHCS briefly considers this fact, but due to the prototype nature of this endeavour, we have left these more complicated details to another, more scalable revision of WHCS.

In WHCS' architecture, the main loop of the base station will not be interacting directly with the radio. This is due to the abstraction we plan on building around the low-level radio driver. All driver specific functions will be wrapped in to a façade pattern, network library. This would allow WHCS to swap out the underlying network hardware for another, similar radio. This would encourage code reuse and prevent massive, expensive rewrites of the net code. The high-level interface that the BS will know about will be sufficient and feature rich enough to carry out all of the actions required for WHCS to come to fruition.

HC-05: The HC-05 BlueTooth module is quite simple in its operations. Data is sent over a two line serial bus and if there is an active connection to a bluetooth enabled device, it will be able to easily receive the data and handle it. In this case the device on the other end is expected to be a phone, but not limited to one. As long as the device on the other end of the BlueTooth link follows the WHCS BlueTooth application protocol, then WHCS will be able to receive commands from arbitrary devices. Assuming that the only thing that will connected to WHCS is a phone, then a suitable protocol for querying and changing WHCS' state will need to be derived. This set of functions is important for more than just the BlueTooth link - if written correctly, it could scale to many different consumers and producers of commands. A simple diagram showing a very high level interaction of a BT device with the base station is for reference in Figure 3. Once again this underlying protocol will be made to be abstracted away from the underlying hardware. For example, if the HC-05 were to fail to meet WHCS' strict requirements, then we would have to switch it for the next best unit - the RN-41. If we write the underlying driver to be "top level" layer that the base station will interact with, then large amount of code and possibly architecture will have to be swapped out to meet the needs of another hardware device.

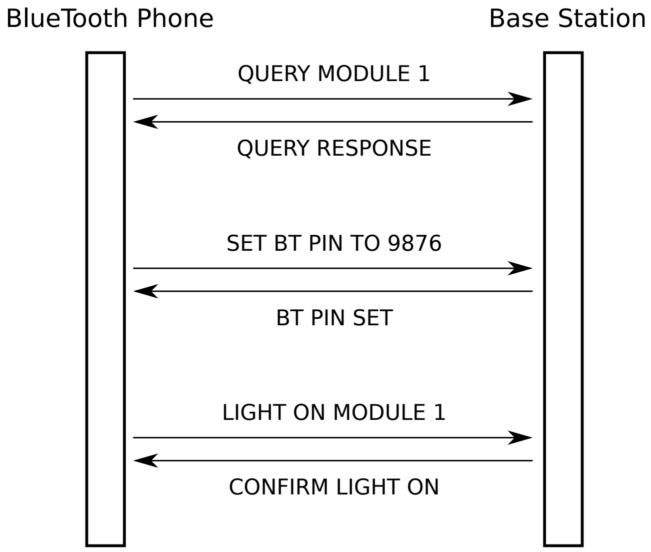


Fig. 3: an example sequence diagram that could occur between a connected bluetooth phone and the base station

In regards to the application level protocol for WHCS, there will need to be a well defined, easy way, for the BlueTooth library to gather information from WHCS' state. This can be handled on the top-level flow of the base station by gluing together two different libraries without them knowing about each other. This is a good approach because it will decouple the two modules from each other, making their individual implementations separate. Two tightly coupled modules may start to take on the appearance of a "ball of mud." A connected phone will be able to accomplish any task that manually interacting with the LCD could handle. This would include controlling the function of individual modules and querying their current state. The BT connection would try to avoid generating too many packets over the NRF radio in response to user events. Instead it would merely lookup the cached state from the base station's memory. This would be faster and the round trip time would be quick. Also, if a bluetooth packet did require a packet to be generated over the NRF radio, this interaction would have to asynchronously tracked until the request was fulfilled. This would complicate the system, but would allow for more advanced queries and commands.

LCD: In what could be considered the "face of WHCS", the LCD module, which will be situated directly over the Atmega32-A, will have the tough job of accurately and quickly conveying any desired information about the state of WHCS' control modules. This is no simple task as not only does it have to display, but with an attached touchpanel, it has to react to user touches. What functionality is exported to the LCD is only limited by the underlying processor speed and the UI library. The high level design of the WHCS LCD will only have to worry about what the end goals are for its usage. An example of this abstraction may be viewed in Figure 4.

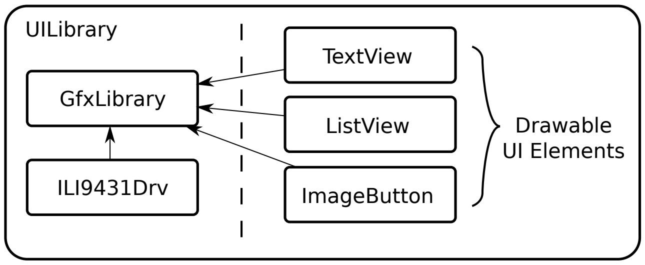


Fig. 4: the level of abstractions for the LCD subsystem

As the state of the WHCS network changes, the base station will have to fire off redraw events in order to keep the LCD up-to-date. These redraws will sync the internal state of WHCS with the user viewable interface. The physical connections to the LCD will consist of data signaling and an 8-bit wide parallel data bus. There is an optional reset pin that WHCS will opt to use for emergency resets and debugging. The high level interface with the LCD will occur directly with the high-level UI library and if necessary the underlying graphics library. The base station should never use the direct driver interface as this is subject to change in an emergency (if the LCD fails to meet WHCS' requirements.) In additiona, a subtle feature that WHCS may choose to implement would be dynamic power saving through screen dimming. Although we assume the base station will have wall power at all times, there may come a time where the system may migrate over to a lower wattage current source, such as power stealing from an HVAC unit. In this case, the system would most certainly have to be power efficient. Despite not needing to worry about power, this function would be simple to implement as only one microcontroller pin is required to control the screen brightness.

Touchpanel: In order to provide a way for an end user to be able to control WHCS from the LCD unit, there is a requirement to poll for touch events. This subsystem can be considered a part of the LCD, but the driver is independent from the graphics and ILI9431 drivers. These events will be dispatched to the appropriate UI element based on the X and Y position of the touch event. There is also an optional Z "position" which represents the pressure of the touch event. This could be used to gather more fine grained information about the touch itself. One of the unfortunate properties of the touchpanel is that it must be actively polled for new touches. This requires that the ADC be constantly providing conversions, which raises the dynamic power of the MCU. This isn't a major concern as the base station is expected to have power from the wall most of the time.

The extent of the touchpanel interaction will occur from a `getTouch()` method. This method will return the latest touch event, if any. The base station will have full control over where this touch event is dispatched to. Depending on the LCD scene (i.e main menu, boot screen), this event will be handled in different ways. Further details are discussed in **??** for the UI library.

Timers: Accurate timing is one of the tasks microcontrollers excel at. The WHCS base station will be using timers to schedule periodic events, wake up from sleep modes, to provide automated send and resend delays, and much more. A simple Timer class will serve as the mechanism and state for a single Timer object. These objects will b e kept track of in a global list and ticked every loop. When a timer reaches the number of ticks or has simply expired, then it will be fired. This firing may cause an action to occur, which is programmable to the specific need. This action may be something like scheduling a network health check or to increase the step of an LCD graphics animation. A UML representation of the WHCS timer class is shown in Figure 5.

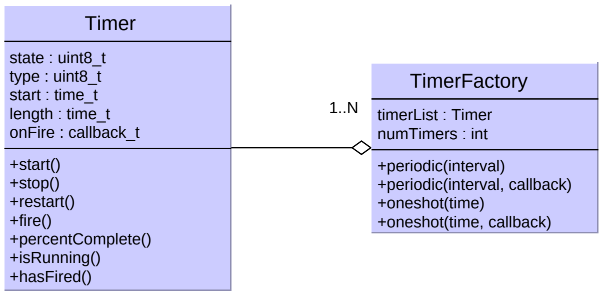


Fig. 5: A UML representation of the Timer class

4) Schematic Breakdown: To tie the whole design of the base station together, the schematic, created in KiCad is broken down below. The full schematic is available for viewing in **??**.

In Figure 6 we see a focused view of the Atmega32-A microcontroller with an attached 16MHz crystal and power passives. The crystal has two capacitors that are dependent on the target crystal. These are required to get the correct oscillation for the external crystal. Also the AREF, VCC, and AVCC lines of the MCU have decoupling capacitors. These are used to make sure that the base station performs well under a large current spike. When designing the board, these capacitors should be placed as close as possible to the

MCU to avoid a long high-current path through the ground plane. The capacitor on the analog reference pin (AREF) is used to stabilize the reference to make ADC conversions more accurate.

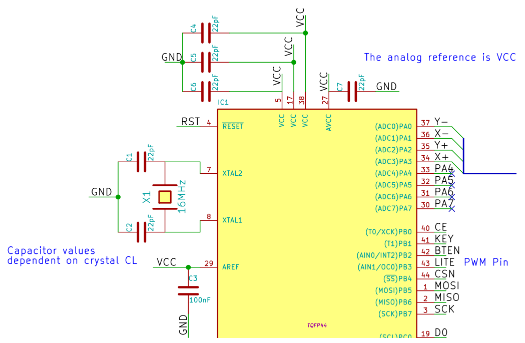


Fig. 6: Base Station crystal and decoupling capacitors

In Figure 7 we see the buses used to connect the LCD to the MCU. This is the most pin heavy component and care must be taken not to mix up any of the signal paths. All of the data control signals are connected to PORTD of the MCU, the touchpanel signals to PORTA (ADC), and the 8-bit parallel data bus completely uses PORTC. There are a few one off signals such as LITE which is a PWM input to control the LCD's backlight brightness.

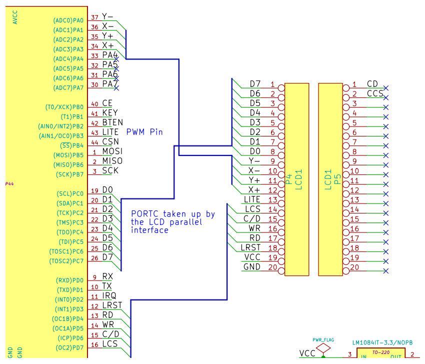


Fig. 7: Base Station LCD header to MCU

In Figure 8 we see the 3.3V 3 terminal regulator converting the 5V VCC line down. Additionally, we see the external RESET pull-up resistor and a manual reset push button. The left corner has the pinout for the NRF breakout board we will be using.

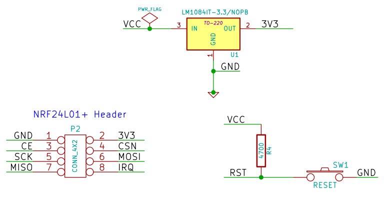


Fig. 8: Base Station power schematic and NRF header

In Figure 9 we see the header for the HC-05. We examine this further because of the unique electrical characteristics of the HC-05 module. The module only accepts 3.3V power and logic. Our MCU will be running at 5V, which means we need a 5V to 3.3V logic shifter. To simply implement this, we used a resistor voltage divider which will provide the required logic level for the TX pin. The RX pin does not need a shifter because 3.3V is still above the V_{IH} minimum for the MCU.

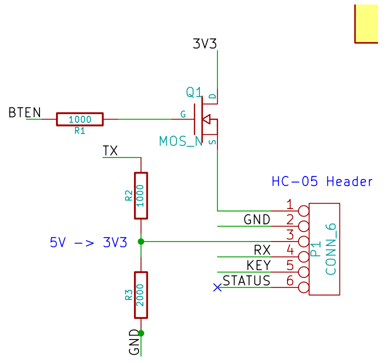


Fig. 9: Base Station HC-05 header

Finally, in Figure 10 we see the standard ICSP header that most AVR line microcontrollers use. This pin array will serve as a quick and easy way to connect an external programmer to our base station while in the field. In this schematic revision, this header can provide power directly to the 3.3V regulator and MCU.

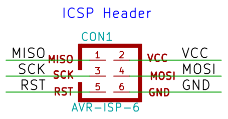


Fig. 10: Base Station ISP header

3. Control Modules

What can be thought of as the "arms of WHCS", the control modules serve as the main devices that seed the network with data. This data is specific to the control module that is emitting it. The base station is more more complex than an individual control module because it needs to be. The control modules should be as lightweight as possible to save cost and keep power usage down. If the control modules were too complex, then the entire cost of WHCS would increase proportionally to the number of control modules.

A. *Software Flowchart*

The general flow for the control modules is much simpler than the base station just due to the requirements of the system. There isn't as much that needs to be done on each loop iteration. The only main module that the control module needs to work with is the NRF radio. This can be seen in Figure 11. Due to the capabilities for the NRF radio to provide an interrupt signal on the reception of a packet, the control module actually has the ability to sleep when not doing anything. Control modules should be as mobile as possible, which limits their overall functionality and processing power. Without these limits, any battery attached would quickly be drained.

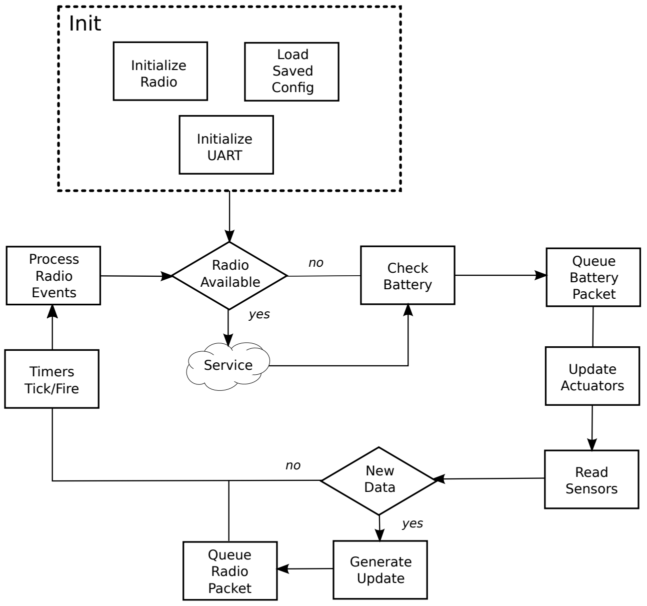


Fig. 11: the high level software flow for a generic control module

Beyond scheduling packets and receving responses, the sensor or actuator that the control module is responsible needs to be serviced. The rate at which the sensor is polled is depending on the required data rate. For instance, if this control module is controlling the door, then it needs to be actively listening for a packet to open or close it. This is quite different from the requirements of a control module that is gathering temperature data. Like the base station, the control module will also have a global list of timers that will govern the timing of events and actions to be taken. The timer list can be associated to any thing that is required to run in the future or periodically.

Events spawned from receiving packets from the base station will change the internal state of the control modules. This state is going to be transitioned between using a specific set of functions that act as an API to the base station. This API will extend across the network layer through a network protocol that will enable the control and querying of the control module's state from remote locations. It is important that this is done right in and in a sustainable manner in order to have clean API for usage across the entire network. With out a clean and well thought API, each control module will have duplicate functionality to handle common tasks. This has already been in further detail in the base station and network library sections.

The mobile nature of WHCS' control modules means that they need to be aware of their power state. We want our modules to consume as little power as possible in order to sustain an isolated power outage. Also, some modules may opt to be battery only if their power requirements are low enough. For instance, a temperature module may be a low enough power consumption that it could last on a battery long enough to be feasible. This requires further field testing and research in order to prove right or wrong. Regardless of the end goals for WHCS control modules, the state of the battery for each control module should be known to the base station for analytics and tracking.

B. *Electronic Strike*

1) Normally Open or Normally Closed: The first thing we considered was whether we wanted a normally open lock or a normally closed lock. Normally opened means that the door requires power to be unlocked and is otherwise locked without power, while normally closed means that the door needs power in order to be locked and is only unlocked when the power is shut off. It was pretty easy for our group to decide that normally open was the better design choice because it would allow the door to be locked most of the time without wasting power. The only issue we saw initially was that it might be a potential safety hazard to have doors locked while there is a power failure. In the case of an emergency this could be a huge problem. We did however find an easy way to have a mechanical alternative (this will be discussed in the next Section 3-B2) so that the safety hazard was no longer present.

2) Strike vs Deadbolt: There are two main types of electric locks to choose from, electric strikes and deadbolts. While some may argue that deadbolts are more secure, electric strikes have the advantage that it they can be used with a regular door knob. This is an advantage because it allows us to include a door knob with a mechanical lock.

That way if there is ever a power failure the mechanical lock can still be used. This gets rid of the safety hazard that could arise if for example a fire where to occur. It also allows for a backup system in case you were to lose your phone or if there were some sort of failure in the electronics that give the command to unlock the door. While perhaps there is a higher level of security that could result from using a deadbolt, the advantage that comes from using an electric strike outweighs the benefit or an electric deadbolt.

C. Sensor Data Collection

The temperature sensor that we have decided to use for WHCS is the TMP36. The TMP36 is widely available and comes in through hole and smd packages. The temperature sensor is simple in design as it has only three pins that require connection. The schematic shown in Figure 12 shows how the temperature sensor would be connected to the ATmega328 on the control modules. The VOUT pin of the TMP36 outputs a voltage signal that varies based on the temperature surrounding the component. The voltage range is between 2.7 to 5.5 volts which will be suppliable through or logic level voltage lines. This model of temperature sensor is capable of sensing temperatures within the range of -40 to 125 degrees Celsius. This covers all the temperatures that one would encounter in a home and more. Connecting to the analog sensor will require the use of one of the ATmega328's ADC (Analog to Digital Conversion) pins for converting the analog signal to a digital signal. There are plenty of ADC pins available on the microcontroller and even when one pin on the ADC port is connected for conversion the other pins can still be used as GPIO pins. The schematic shows the output of the temperature sensor going to pin ACD0 but any ADC pin will be able to accomplish converting the signal to digital.

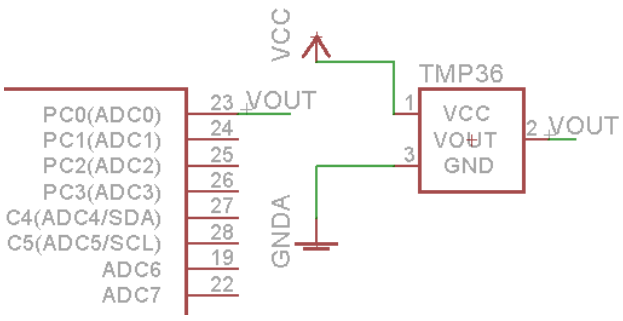


Fig. 12: Temperature Sensor Connection Schematic

D. Light and Outlet Control

Once we narrowed down our solution to controlling lights and outlets through the use of a SSR, we searched for a chip that had the electrical characteristics we were searching for. The GPIO pins of the ATmega328 that we are using for the control modules are capable of outputting a maximum of 40mA DC current. We needed a solid state relay that had a tolerance for 120V AC as a load voltage, could be supplied by 5V, and needed less than 40mA for activation/forward current. We found a solid state relay made by Sharp Microelectronics with the part number S108T02F that met all the requirements that we set. We confirmed that the chip is in stock and suppliable by digikey. The chip is available at an affordable price of \$5.10. Figure 13 Shows a schematic using this solid state relay. The schematic for the control module would mimic the one shown in this figure. The activation input would be directly connected to the microcontroller's GPIO pin in order to toggle on and off the state of the relay. This particular relay has an activation voltage of 1.2V DC which means that when the relay is on this is the voltage across the diode shown in the schematic. Thus the forward current for this schematic can be calculated through the equation (5V-1.2V)/R1. The SSR in the figure requires at least 15mA for activation. The microcontroller's pins are adequate for supplying this low current. When the microcontroller's activation pin, pin PB1 in the figure, is set to high, the 120V AC is free to flow through the triac of the SSR and power the component in place of the load resistor. In WHCS the load resistor RL in the figure will be replaced with an outlet or a light. Whenever the microcontroller pin goes high, the light or outlet will receive the power it normally would from the household main.

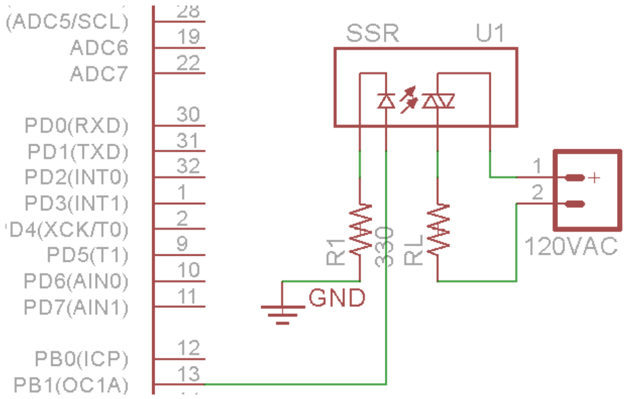


Fig. 13: Wiring Schematic for Solid State Relay

E. Schematic Breakdown

The control modules for WHCS have to be able to support communication via a radio transceiver as well as interaction with their target endpoints. Figure 14 shows the schematic for the control modules that will be implemented in WHCS. The full schematic is available for viewing in **??**. The main component of the schematic is the ATmega328.

Everything in the schematic is connected to the microcontroller in some way. In the schematic three different VCC lines are shown. This is because the control modules will have to access to a 3.3V line, a 5V line, and a 12V line. The power board will supply these power lines to the control module. The 5V and 3.3V lines are necessary because they provide power to the logic chips like the microcontroller and the radio transceiver. The 12V line is necessary solely for the electronic strike that we have chosen.

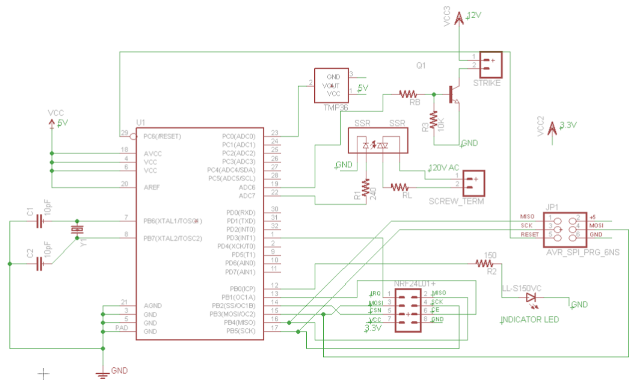


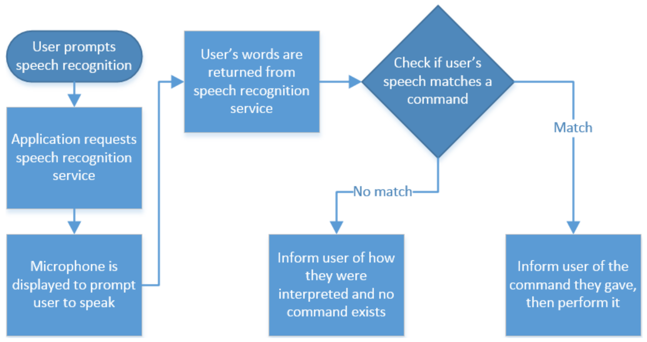
Fig. 14: WHCS Control Module Schematic

4. Android Application

For most WHCS users the mobile application will be the only physical interaction they have with the application. When we set out for development we wanted to make an easy to use application that would attract users to stick with our system. Operability and usability were emphasized in our design process. We wanted an appealing U.I. without complexity, after all we are targeting a simple solution to home automation.

A. Speech Recognition

The Android application for WHCS will offer speech activation capabilities. These will be on top of GUI activation capabilities. The speech activation sequence begins with the press of a button to start the speech recognition. The user will be prompted with a microphone and can then give his command. The commands will be formatted like "light one on." When the user gives commands using the speech method, a notification will be given indicating the success of interpreting the speech into a known command. If the user's speech does not match a known command, the speech will be shown back to the user to show what went wrong. We are predicting that the most frequent cause of this will be the Android phone mishearing the user. In the event that the speech matches a command, the application will display the command to the user and then perform it. The following flow chart in Figure 15 displays the sequence of events happening when a user performs speech activation.



User prompts speech recognition
Application requests speech recognition service
Microphone is displayed to prompt user to speak
User's words are returned from speech recognition service
Check if user's speech matches a command
Match
No match
Inform user of how they were interpreted and no command exists
Inform user of the command they gave, then perform it

Fig. 15: Android app speech activation chart

The goal of the speech activation feature is to be easy to use. In order to promote the usage of this feature we will add the ability for users to rename the endpoints that the speech commands will target. For example the user could change "light 1" into "living room light." This way the user could say "living room light on" to the application in order to turn on the living room light. To do this data structures will need to be stored in the application which hold the preferred name of each type of endpoint. Endpoints can be distinguished by the type they are, their individual identifier number and their preferred name. The preferred name should be stored when the application is closed so a permanent source of storage is needed to do this. The file system can be used or possibly a SQLite database.

In the code for our application we will be using the Android speech recognition API (Application Program Interface). Android has a speech recognition service that can be started by requesting it within an application. We will request this service to be run by using an Android construct called an intent, specifically the recognizer intent. Once the request the service to be run it gives us the text that it produced from listening to the user's speech. The code that performs this process ends up bloating up the application so we sought to develop a wrapper class in order to perform the request for the speech service and simply hand back the text. However because of the Android design philosophy, creating a wrapper class to start the speech recognition service was not easy enough to make it a worthwhile endeavour. Thus we concluded the best approach is to keep the calls to the Android speech recognition API within the class we use for our main activity.

B. BlueTooth Software Design

BlueTooth will be the technology that allows WHCS users to interact with the base station from the mobile phone. This means that proper functioning BlueTooth software must be written to ensure that users can interact with WHCS. From the user's standpoint the only knowledge of BlueTooth required will be the ability to perform an initial connection to the base station. Once a user has connected to the base station once through the WHCS app we will be able to cache the base station device and allow for automatic reconnection every time the application is launched. This is an important abstraction for the user because the user should not have to spend time handling BlueTooth connections every time they open the application. Figure 16 shows what the BlueTooth software will be doing whenever the user opens the Android application.

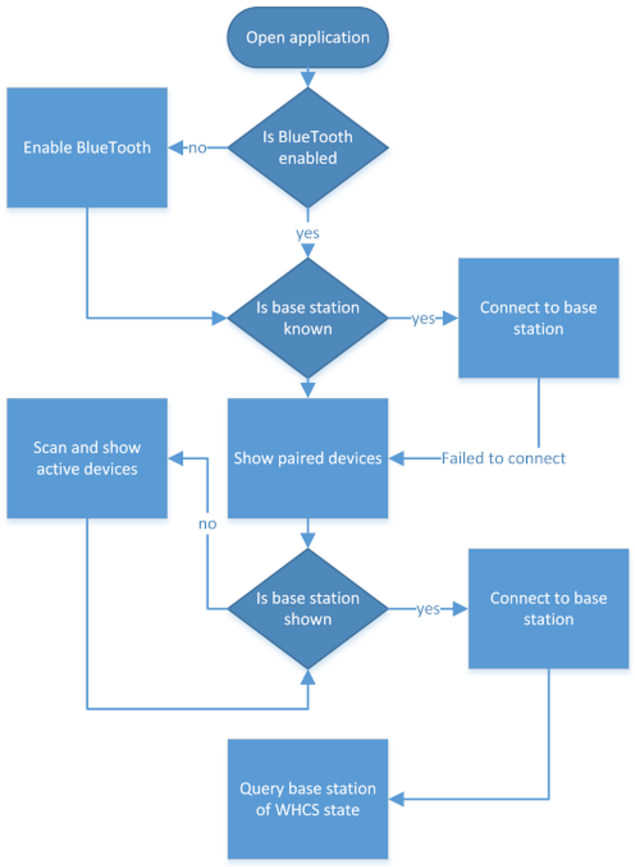


Fig. 16: Android BlueTooth Startup Flowchart

In Figure 16 we see that the first check that is made is to ensure that BlueTooth is enabled. The Android operating system requires applications to ask the user whether they want to activate BlueTooth or not. It cannot just be turned on. If the WHCS application is opened and BlueTooth is off we will prompt to the user to turn it on and if they refuse we will exit the application. When it has confirmed that BlueTooth is on, the application can check to see if it knows the base station device. If the base station device is known then the application can skip asking the user what to connect to and can perform the connection automatically. This is what should be happening most of the time. If the base station is not stored in the applications data then the application will have to prompt the user to connect to a base station. When connecting to a device there are two possibilities for connection, paired devices and non-paired devices. The application will first show the user all devices that their phone has paired with previously, in case the application somehow forgot the base station. If the base station does not show up in the paired devices list, the user will be able to search for active BlueTooth devices and select the base station. At the end of this start up cycle the WHCS application will have an active BlueTooth connection with the base station that can be used for full duplex communication.

Our application will be leveraging the API and design guideline for using BlueTooth from Android phones. The underlying driver for BlueTooth communication utilizes sockets similar to network sockets in other languages. Android offers a class named BluetoothDevice which contains all the address information necessary for opening a socket. When our application scans for devices or asks the user to pick an option from the list of paired devices this will be to get the BluetoothDevice to open a socket from. Once we have obtained that BluetoothDevice we can create a BluetoothSocket through one of its methods. Once a BluetoothSocket has been opened through calling connect, an input and output stream become available that allow us to send and receive raw byte data. This is a primitive form of communication but it is also exactly what we want. All data that we send or receive from the base station over BlueTooth will be in the form of a byte array. This form of primitive data transmission allows us to implement certain communication protocols between the Android base station.

Once a BluetoothSocket has been opened on the Android device the application can begin communicating with the base station. We will use a communication protocol between the Android device to ensure the base station can properly interact with the application. This protocol will allow the Android application to give commands to the base station such as inquire about the state of the control modules or to toggle state within the system. Whenever the Android application wants to send a message to the base station the software will create a packet with a certain structure. The packet will contain a byte for letting the base station know that a command is being given, the command itself, any variables for the command, and then a byte for finishing the command. The base station will receive one byte at a time due to the serial nature of BlueTooth communication but it will be able to parse the packets it receives in order to figure out what action the application is trying to perform. Figure 17 shows a visual representation of the communication between the aplication and the base station.

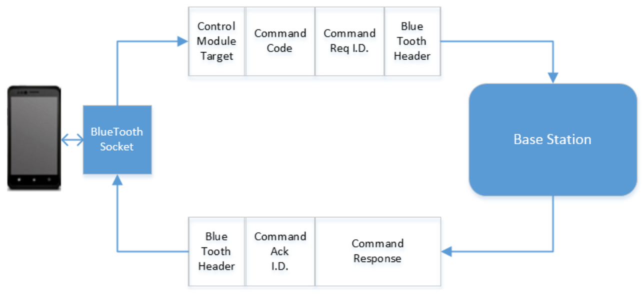


Fig. 17: Visual of Communication Between Android Device and Base Station

5. Power Hardware

Each power board is to convert 120VAC to DC lines of 3.3V 5V and 12V. We will also need to be able to switch on and off 120VAC for the outlet and light switching control modules as well switch on and off the 12V used to operate the strike. The designs for each board will be mostly the same with slight variations depending on the application.

A. *Power Consumption*

In this section we will discuss the amount of power consumed by the boards. First of all it is important to make note of the fact that saving power is not of incredible importance to us. While it is important not to be incredibly wasteful to the point that it becomes a problem, power was not something that we decided we wanted to be competitive on. Had we wanted to be more competitive with power, we would have taken a lot more into account and made different design decisions. For example with the microcontrollers we would have looked more carefully into the amount of current that they drew to help us weigh our decisions. MSP430 boards for example would have been attractive because of the low amounts of current that they draw. With that said we made all of our decisions based on performance in other aspects. The comparisons that we made and the details that were of importance to us can be shown in the sections were we decided on each component that we selected.

In Table I below we have all the information on the amount of current that is to be drawn from the different elements in our design. Note that the information from the datasheet about the current drawn for the microcontrollers is under a clock speed of 16MHz. We choose to run everything at 16 MHz because is was the maximum speed for the Atmega32A. Although the Atmega328P can run at a higher clock speed of 20 MHz, we decided to to make everything consistent to run at 16MHz. The reason why we give ranges for the currents and voltages of the microcontrollers is because there are a number of different current voltage relationships that can achieve the same clock speed. This isn't to say however that any combination will work. If a lower current is selected a higher voltage must be selected in order to maintain the same clock speed of 16MHz.

	Operating Voltages	**Current Drawn**
Atmega32A	4.5-5.5V	12-16mA
Atmega328P	4-5.5V	Active: 7-11mA
Electric Strike	12V	450mA
Adafruit TFT LCD	3.3V	150mA
Bluetooth HC 05	1.8-3.6V	35mA
nRF24L01+	1.9-3.6V	13.5mA
TMP 35 36 37	2.7-5.5V	.05mA

TABLE I: Currents and voltages of devices used in WHCS

Note that the microcontroller of the control modules shows both an active and a power saving mode. When the module is in operation it will run in active mode, however since our microcontroller has the capability switch into a less consuming power mode, we will utilize this mode in order to save power. The microcontroller for the base station also has a low power setting for idle use, yet we will not be running in this mode for the base station because it is impractical. Some of the datasheets gave us specific information on how much current would be drawn while for others we were simply able to estimate the limit. For example the LCD (most of it's current will be drawn from the backlight) only gave a description of the LDO that would be 3.3V at 150mA. Therefore we were able to assume that the current would not surpass 150mA.

As you can see the components in WHCS do not draw a lot of power. This is because the components that constitute the system are all intrinsically low power devices. Based on the fact that Power = Voltage*Current and that not all these devices will be used at a time. We can see that the devices themselves will never really draw that much power.

Now that this data has been presented it can be used as a reference later when the amount of power drawn plays a part in design decisions. Again this section was not made to try and explain why certain decisions were made in order to try and conserve power. Our design is not heavily concerned with testing the limits of low power applications. Rather this section was made to present the data of the power usage that comes from the devices that have already been selected to be used in our design.

6. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

7. References

[1] X10 Website www.x10.com
[2] Fuse Calculated http://www.engbedded.com/fusecalc/
[3] TFT 2.8" LCD https://www.adafruit.com/products/1770
[4] 5V Battery Backup http://www.instructables.com/id/Simple-5v-battery-backup-circuit/
[5] 12V Battery Backup http://electronics.stackexchange.com/questions/96632/12v-battery-backup-supply-for-gprs-tracker
[6] Capacitor Tut http://www.electronics-tutorials.ws/capacitor/cap_2.html
[7] Diode Ref Sheet http://www.diodes.com/datasheets/ds28002.pdf
[8] Solid State Relay http://electronicdesign.com/components/electromechanical-relays-versus-solid-state-each-has-its-place
[9] LDO Dropout http://focus.ti.com/download/trng/docs/seminar/Topic%209%20-%20Understanding%20LDO%20dropout.pdf
[10] ISOCompare http://www.tortech.com.au/isocompare
[11] OSH Park https://oshpark.com/pricing
[12] 4PCB Service http://www.4pcb.com/33-each-pcbs/
[13] Sparkfunhttp://sparkfun.com
[14] Circuit Tutorial http://www.allaboutcircuits.com/vol_5/chpt_2/2.html
[15] Electrical http://homerenovations.about.com/od/electrical/a/artelecbox.htm
[16] GFI Outlet http://diy.stackexchange.com/questions/15684/what-is-a-gfi-outlet-used-for-and-where-should-i-install-them
[17] EAGLE Cadhttp://www.cadsoftusa.com/download-eagle/freeware/
[18] FreeRouter KiCadhttp://www.freerouting.net/
[19] Nest Website http://nest.com
[20] Forbes Article Nest Acquisition http://www.forbes.com/sites/greatspeculations/2014/01/17/googles-strategy-behind-the-3-2-billion-acquisition-of-nest-labs/
[21] Works with Nest https://nest.com/works-with-nest/
[22] ILI9341 http://www.newhavendisplay.com/app_notes/ILI9341.pdf
[23] ILI9341 Adafruit Lib https://github.com/adafruit/Adafruit_ILI9341/tree/master/examples
[24] Adafruit 2.8" TFT https://learn.adafruit.com/adafruit-2-dot-8-color-tft-touchscreen-breakout-v2
[25] ILI9341 Fruit https://github.com/adafruit/Adafruit_ILI9341
[26] Bresenham Line Algo https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

8. Engineering Team

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.