CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Embedded System: A Solution for Home Security

A thesis submitted in partial fulfillment of the requirements
For the degree of Master of Science
in Computer Science

By

Roland Lee Hern

December 2013
The graduate thesis of Roland Lee Hern is approved:

______________________________________  ________________
Professor R. Covington  Date

______________________________________  ________________
Professor J. Noga  Date

______________________________________  ________________
Professor J. Wiegley (Chair)  Date

California State University, Northridge.
Acknowledgements

I would like to acknowledge my first programming teacher, Ron Miller for the logic and the fundamentals of computer programming that I still utilize today. I am forever grateful to you, Kiddo.

I wish to thank my chair, Professor Wiegley for your support and guidance.

I also wish to thank my committee members, Professor Noga, and Professor Covington for your patience and contributions to my education. They have always been greatly appreciated.

Thank you to Dr. Melissa Midzor for opportunity and encouragement that know no limits.

Thank you to my mother-in-law, Claire, for your support and encouragement.

Thank you to my father, Ben. Your support made this possible.

Finally and most importantly, I wish to thank my wife, Deb. I have accomplished a difficult goal but I did not accomplish it alone. Sharing a brain, love, sense of humor, and life with you made this possible.
# Table of Contents

Signature page.......................................................................................................................... ii  
Acknowledgements.................................................................................................................. iii  
List of Figures ............................................................................................................................ v  
Abstract .................................................................................................................................... vi  
Introduction ............................................................................................................................... 1  
Communication Technologies .................................................................................................... 3  
  X10 / INSTEON......................................................................................................................... 3  
  Z-Wave..................................................................................................................................... 10  
  ZigBee / XBee .......................................................................................................................... 14  
Hardware ................................................................................................................................... 17  
  iDigi® Gateway Development Kit ........................................................................................... 17  
    ConnectPort X4 ...................................................................................................................... 17  
    XBee Smart Plug ................................................................................................................... 18  
    XBee Light/Temperature Sensor ........................................................................................... 18  
  Embedded System .................................................................................................................... 21  
    Rabbit® MiniCore® RCM6700 ............................................................................................... 21  
  External Hardware .................................................................................................................... 24  
Cloud Environments .................................................................................................................. 27  
  iDigi® Device Cloud ................................................................................................................. 27  
  Google Appspot.com Cloud .................................................................................................... 33  
Rabbit Security System ............................................................................................................ 36  
Conclusion ................................................................................................................................. 39  
References ................................................................................................................................. 40  
Appendix A – Rabbit Security Controller Code ......................................................................... 42  
Appendix B – ConnectPort X4 Update Code ............................................................................. 56  
Appendix C – ConnectPort X4 Configuration File ..................................................................... 63  
Appendix D – Google Appspot Tutorial Code .......................................................................... 65
List of Figures

Figure 1 – System Architecture ........................................................................................................ 2
Figure 2 – AC Transmit Timing [4] ....................................................................................................... 6
Figure 3 – Two-phase Home INSTEON Installation [4] ....................................................................... 8
Figure 4 – Z-Wave ZM5304 Serial Interface Module with Antenna ................................................. 12
Figure 5 – Z-Wave protocol stack [6] .................................................................................................. 13
Figure 6 – Z-Wave protocol frame structure [6] .................................................................................. 13
Figure 7 – Outline of the ZigBee Stack Architecture [7] ..................................................................... 15
Figure 8 – ZigBee General Application Frame Format [7] .................................................................. 16
Figure 9 – ConnectPort X4 .................................................................................................................. 17
Figure 10 – XBee Sensor and Smart Plug ............................................................................................ 18
Figure 11 – Sample “idigi_dia.html” Local Website ........................................................................... 20
Figure 12 – Rabbit MiniCore RCM6700 ............................................................................................ 21
Figure 13 – Assembled Rabbit with IO board attached ........................................................................ 22
Figure 14 – iDigi Device Cloud – iDigi Manager Pro ....................................................................... 27
Figure 15 – iDigi Device Cloud – XBee Networked Devices ............................................................... 28
Figure 16 – iDigi Device Cloud – Add Device Alarms ....................................................................... 28
Figure 17 – API Explorer ..................................................................................................................... 29
Figure 18 – Sample iDigi Development Kit Website .......................................................................... 33
Figure 19 – Sample Google Chart’s .................................................................................................... 34
Figure 20 – Google App Engine Launcher ........................................................................................... 35
Figure 21 – Rabbit remotely responding to iDigi Cloud commands .................................................. 36
Figure 22 – Rabbit responding to local commands .......................................................................... 38
Abstract

Embedded System: A Solution for Home Security

By
Roland Lee Hern

Master of Science in Computer Science

The objective of this thesis is to present the reader with the latest advances in home automation technologies with an emphasis on security. The X10/INSTEON, Z-Wave and ZigBee/XBee technologies are examined for their usefulness in constructing a subscription based residential security system. The central control unit of this security system is an iDigi Rabbit RCM6700 embedded system. The application running on the Rabbit demonstrates this embedded system’s ability to monitor and control attached devices. The Rabbit application also communicates with another application running on Google’s Appspot.com cloud environment through the iDigi’s device cloud environment to provide the user with system interactivity and system status.
Chapter 1

Introduction

Home automation is the introduction of technology to make a living environment more convenient. The brain of the system is the central control unit that is programmed to monitor and control devices around the home. As the home automation industry has grown, so have the number of devices that can be controlled. Light fixtures, television and audio systems, major appliances, thermostats, cameras, motion sensors, door sensors and many more devices can be purchased with the goal of incorporating them into a home automation system. These devices can be adjusted, turned on/off by the system, or programmatically controlled when other events occur. The user can make changes to these devices using either an application running on a smartphone, a touchpad mounted on the wall, or a website that can be accessed anywhere in the world. These remote controls allow the user to interact with the system to gain system status or make changes to it.

Until the 1990s, the idea of home automation was the stuff of science fiction, but as the price of microprocessors has decreased, and the availability of wireless technology devices has increased, that dream has become a reality in many homes. Currently 1.7 million homes have an automation system installed, so the market has room for growth. According to a poll of online users interested in home automation, 62% of them cited security as their primary purchasing motivation [1]. It is now up to the industry to provide potential customers with the knowledge of how these products can better their lives.
A security provider, utilizing cloud computing, can remotely monitor, manage and configure their client’s central control unit. A monthly subscription fee is charged for system monitoring, client notification and communication with authorities in case an emergency is detected.

Figure 1 – System Architecture

Subscribers log into a secured company website to monitor their home and make adjustments to their home automation system. Depending on the type of devices installed, the user can arm or disarm their security system, lock or unlock door locks, adjust lights and thermostat settings, and view cameras from anywhere. Companies such as ADT, Honeywell, Safemart, SimpliSave, Time Warner Cable, and Vivint are just a few of the currently available providers of such services.
Chapter 2

Communication Technologies

Communication with remote devices is the first hurdle to home automation, and many technologies have been developed to accomplish this; either a wired or a wireless connection needs to be established to allow devices to receive commands from the central control unit. In this section, three types of popular technologies will be discussed. X10 and INSTEON are two of the oldest technologies, but can be limited in their usability due to restrictions on device network size and sensitivity to environmental conditions. Z-Wave is newer, wireless, faster and more reliable, but because it is a proprietary technology with license fees, that could make it cost prohibitive for some applications. And finally, ZigBee and Xbee are also wireless technologies that are nonproprietary and based on the IEEE 802.15.4 standard that provide a very customizable system at a lower cost than Z-Wave.

X10 / INSTEON

History

Pico Electronics, a Scottish engineering company, developed the X10 system during the mid-1970s. They named it X10 because it was the 10th project for the company. In 1978, ready for release in America, it was sold through Radio Shack, Sears, and BSR. The Radio Shack “Plug ‘n Power System” package included a 16 channel Command Console, Lamp Module, and Appliance Module [2]. Soon a Wall Switch module allowed for more complex system designs. A year later, a timer module was introduced. The timer module
was the first of its kind, so Pico/X10 hired an ad agency to create the product’s name; they came up with “The Timer.” It was sold under that name by Macy’s [3].

Maintaining a low system cost was an important factor from the beginning. To keep production costs low, Pico/X10 opened manufacturing facilities in China. To reduce installation costs, X10 utilizes electrical wires already installed in the home, so a system can be placed in a home without running additional wires. To reduce manufacturing unit costs, each unit only has a communication receiver, and only the command unit transmits over the single-band mesh network. Confirmations to commands are not sent back to the command module. With X10, it is possible to configure a dual-band mesh network and incorporate RF communication, but those devices cost two to four times the single-band devices. Given the over 35-year history of X10, there are a lot of compatible devices available on the market now.

Technology

Taking advantage of the existing electrical wiring found in most homes, X10 and INSTEON technologies utilize the zero crossing period to broadcast commands one bit at a time. Zero crossing is the quiet period of power oscillation on a 60 Hz AC power line.

The X10 commands format: Home Address, Device Address, and Command.

- 4-bit – Home Address (Represented by A-P)
- 4-bit – Device Address (Represented by 1-16)
- 4-bit – Command (Some examples include: All Devices Off, All Lights On, All Lights Off, Single Device On, Single Device Off, Dim Light, Brighten Light)
A command like “Select Code: K3” followed by “Turn On” would turn on the device at Home Address K, and Device Address 3. Daisy chaining the commands is also possible, as in “Select Code: K3”- “Select Code: K4”- “Select Code: L10” followed by “Turn On” would turn on all three of the selected devices at the same time. Daisy chaining is possible because of the waterfall method in which commands are received. When the “Select Code:” is received, all devices on the network listen for the next command. When the “K3” command is received, the device at network address K3 continues listening for a command but the other devices stop listening. By transmitting multiple “Select Code XX” commands, all of those devices continue to listen for a command; in the example above, “Turn On”. Once received, all of those listening devices will respond by turning on.

Smartlab’s INSTEON system was released in 2000 and is similar to X10 in that it communicates over AC power lines. However, INSTEON was also designed with RF communication ability, and a more reliable dual-band mesh network from the start. Each device acts as a peer-to-peer router and rebroadcasts all received messages three times. This provides a wider range of connectivity and increases the chance of the command reaching the intended device.

Network data throughput is another advantage INSTEON technology has over X10. This is achieved by beginning data transmission 800 microseconds before the zero crossing (see Figure 2) for a total transmit time of 1823 microseconds at 131.65 KHz, while X10 starts at zero crossing and transmits for 1023 microseconds at 120 KHz. This gives INSTEON a significant increase in data throughput per cycle compared to X10.
INSTEON data packets are also larger than X10’s 4-bit data packet, and this allows for a larger variety of commands. A standard INSTEON message has one start packet with 12 data bits followed by 4 body packets with 18 data bits each. The second half of the last body packet is ignored, giving a standard message a total of 80 bits or 10 bytes of usable data. Extended messages can be used for encrypting data when security is needed. An extended message consists of one start packet followed by 10 body packets for a total of 192 bits or 24 bytes of usable data.

![AC Transmit Timing](Image)

**Figure 2 – AC Transmit Timing [4]**

INSTEON RF messages also have a standard and extended message form. They contain the same amount of usable data but the sync and start code bytes are slightly different than wired messages. When an RF message is received by devices attached to a wired network, it is rebroadcast like commands received from another wired devices.
Most American homes have 2 phases of electrical wiring installed; this can lead to network messages being cut off from devices not on the same phase. To deal with the split electrical, INSTEON uses RF Dual-Band devices to bridge the two phases and allow devices on either phase to communicate.

“Electrical power is most commonly distributed to homes in North America as split-phase 220-volt alternating current (220 VAC). At the main electrical junction box to the home, the single three-wire 220 VAC powerline is split into a pair of two-wire 110 VAC powerlines, known as Phase 1 and Phase 2. Phase 1 wiring usually powers half the circuits in the home, and Phase 2 powers the other half.

Powerline signals originating on the opposite powerline phase from a powerline receiver are severely attenuated, because there is no direct circuit connection for them to travel over. A traditional solution to this problem is to connect a signal coupling device between the powerline phases, either by hardwiring it in at a junction box or by plugging it into a 220 VAC outlet. INSTEON automatically solves the powerline phase coupling problem through the use of INSTEON dual-band devices capable of both powerline and RF messaging. INSTEON RF messaging bridges the powerline phases whenever at least one INSTEON DUAL-BAND device is installed on each powerline phase.” [4].

A typical installation can be seen in Figure 3, with INSTEON Dual-Band devices joining power line phases and allowing network-wide communication. As each device attached to the network rebroadcasts all received commands, the more devices used increases the chance a command will reach the intended device.

Some of the complaints about X10 technology include its susceptibility to environmental interference, lack of acknowledgements, and command response times. Installations in
congested large cities, and factories where florescent lights or appliances with large electric motors may be present, may result in transmissions disrupted by line noise.

![Figure 3 – Two-phase Home INSTEON Installation](image)

Noise filters may lessen the impact of line noise and allow commands to reach their intended device more consistently, but that increases the installation cost of the network. INSTEON is less susceptible to line noise because it uses RF broadcasting as a backup, and each device rebroadcasts all received commands three times over the entire network.
Unlike X10, when an INSTEON command is issued, the receiver sends back an acknowledgement. If the acknowledgement is not received, the transmitter will automatically rebroadcast the command and use other nodes on the network to reach the intended receiver.

Another complaint about X10 is the device limit (256 possible) due to the addressing structure. More complex home automation installations can run out of possible addresses. To avoid this, INSTEON assigns a unique home identification to each device when it is added to the network. This gives the network over 16 million possible device addresses. In congested areas with multiple networks nearby, these identifiers stop devices from responding or rebroadcasting commands from other networks.

While many companies produce inexpensive do-it-yourself security systems based on the X10 technology, those systems lack the reliability, responsiveness, and robustness needed to create a professional looking security system.
Z-Wave

Unlike X10 and INSTEON with their mixture of wired and wireless connections, Z-Wave and ZigBee are strictly wireless RF solutions. Z-Wave communicates in the 900 MHz range, which gives it a greater throughput of 9600 bits/s compared to the zero crossing technique of X10. Z-Wave has become one of the most widely used communication technologies in the home automation field, and some consider it to be an industry standard. Recently, the International Telecommunications Union (ITU) included the Z-Wave PHY and MAC layers as an option in its new G.9959 standard, which defines a set of guidelines for sub-1-GHz narrowband wireless devices [5]. The Z-Wave Alliance is a collection of companies that design and sell home and office automation solutions around the world.

History

Conceived in the late 1990s, The Z-Wave standard is a proprietary protocol created and licensed by Zensys. Z-Wave technology targets home and small business automation applications such as light, security and climate controls. “In 2005, as more companies adopted the protocol, the founding members of the Z-Wave Alliance; Intermatic, Leviton, Wayne Dalton, Danfoss, and Universal Electronics met with Zensys to discuss their needs. Today the Z-Wave alliance has over 150 member companies. Zensys was acquired by Sigma Designs in 2008.” [5] Z-Wave has grown into one of the leading technologies for communicating between automated devices with over 12 million devices sold around the world.
The Z-Wave Alliance controls the licensing of all Z-Wave devices, and requires that all Z-Wave devices be certified by the Z-Wave Alliance Labs before being sold on the market. Manufacturers may also hire the lab to provide device-to-device interoperability testing, single device capability testing, and to manage the certification process for any newly created devices. When dealing with RF communication, it is important for manufacturers to know the transmission range and capabilities of their devices as this directly impacts the overall reliability and performance of the Z-Wave network, and how the customer perceives the quality of their devices.

**Technology**

Broadcasting in the Part 15 unlicensed ISM band, in the less-cluttered frequency range of 908.42 MHz, Z-Wave does not need to compete against other technologies like Wi-Fi and Bluetooth. This frequency also gives it better range and penetration in home and office environments than other technologies.

With an effective range of 30 meters, Z-Wave utilizes a mesh network to daisy chain devices together to improve device communication in homes and commercial office environments where structural interference is present. A device on a mesh network, also known as a self-healing network, keeps track of which devices it can communicate with through broadcasts. If a device is moved out of range, it may still be able to communicate via another device on the network. This increases the effective distance of the network and circumvents obstacles that cause the environmental interference.

“The Z-Wave protocol is an interoperable wireless RF-based communications technology designed specifically for control, monitoring and status reading applications in residential
and light commercial environments. Mature, proven and broadly deployed (12 million products sold worldwide), Z-Wave is by far the world market leader in wireless control, bringing affordable, reliable and easy-to-use 'smart' products to many millions of people in every aspect of daily life.” [5]

The Z-Wave ZM5304 is the latest release from Sigma Designs. It is a fully contained serial interface module that can be integrated into a newly manufactured product for roughly $5.00. It contains a Z-Wave modem and antenna. The target application for this device would be Gateways, TVs and Set-top Boxes. [6]

Figure 4 – Z-Wave ZM5304 Serial Interface Module with Antenna
The proprietary protocol stack of the Z-Wave can be seen in Figure 5, and the protocol frame structure is shown in Figure 6.

**Figure 5 – Z-Wave protocol stack [6]**

The Z-Wave protocol stack is similar in structure to TCP/IP, but simplified to reduce lag and network collisions during rebroadcasts. The transmissions may be directed to a particular device or broadcast over the network.

**Figure 6 – Z-Wave protocol frame structure [6]**
**ZigBee / XBee**

Unlike Z-Wave, which is a proprietary protocol, ZigBee is based on the IEEE 802.15 standard (Wireless Personal Area Network) and is available from multiple vendors. Like Z-Wave, ZigBee is also based on a mesh network and gains robustness as more devices are added to the network.

While similar in many ways, these two standards appear to be finding niche markets of their own. Z-Wave’s focus remains in home and office automation. While ZigBee still remains active in that market, their focus has transitioned to include more industrial controls, utility and medical data collection applications. The 2007 adoption of the ZigBee Pro standard demonstrates the shift in their focus.

**History**

The ZigBee Alliance started in 2002 with 25 members. As of 2012, the ZigBee Alliance had grown to over 400 members with over 600 certified products.

Adopted in 2007, the ZigBee Pro standard focuses on the growing demand from the power and utility industries [8] and adds additional features to facilitate ease-of-use and advanced support for larger networks:

- Network scalability – Improved support for larger networks offering more management and flexibility.
- Fragmentation – New ability to divide longer messages and enable interaction with other protocols and systems.
- Frequency agility – Networks dynamically change channels should interference occur.
- Automated device address management – Optimized for large networks with added network management and configuration tools.
• Group addressing – Offers additional traffic optimization needed for large networks.
• Wireless commissioning – Enhanced with secure wireless commissioning capabilities.
• Centralized data collection – Tuned specifically to optimize information flow in large networks.

With the addition of these features, ZigBee Alliance recognizes the growing market of utility companies and allows home automation or building control systems to easily connect wirelessly to an exterior utility meter.

XBee is the trademarked brand name of ZigBee devices sold by iDigi International.

Technology

ZigBee broadcasts in the 2.4 GHz range globally and 915 MHz in North America, and uses multiple channels to reduce interference from other Wi-Fi and Bluetooth devices.

![Figure 7 – Outline of the ZigBee Stack Architecture][7]

Because multiple vendors create radios for the IEEE 802.15 standard, certification that they meet the standard is crucial; one radio has the potential to disrupt the entire network.
Layer 1 (Physical) and layer 2 (MAC) are provided by IEEE 802.15.14 while the ZigBee stack software provides the Application and Network layers.

ZigBee uses a simpler general application frame format than TCP/IP, with the goal of keeping packet size small.

![ZigBee General Application Frame Format](image)

**Figure 8 – ZigBee General Application Frame Format [7]**
Chapter 3

Hardware

iDigi® Gateway Development Kit

The iDigi Gateway Development Kit is a relatively inexpensive way to get started with home automation. The kit includes a ConnectPort X4, XBee Smart Plus, and XBee Temperature and Light Sensor. The ConnectPort X4 is the hub of the XBee network and connects the other devices together. As new devices are added to the network, they will join the XBee network via the ConnectPort X4. The ConnectPort X4 also connects the XBee network to the Internet.

The iDigi ESP for Python software used to configure these devices may be downloaded from the iDigi website. It is based on an Eclipse Project IDE and provides configuration control for all of the devices in the kit.

![ConnectPort X4](image)

**Figure 9 – ConnectPort X4**

ConnectPort X4

The ConnectPort X4 is a coordinator device that provides connectivity between the ZigBee network and its attached devices to the iDigi Cloud environment. The
coordinator device cannot sleep; it is the central device of the network and the network cannot function without it. It also serves as a local webserver to allow the manipulation and monitoring of the devices attached to the XBee network.

It has ports for Ethernet, Serial and USB connections, and a wireless connection for XBee devices.

![XBee Sensor and Smart Plug](image)

*Figure 10 – XBee Sensor and Smart Plug*

**XBee Smart Plug**

The Smart Plug is a routing XBee network device that will reroute commands from one device to another device if there is interference blocking those two devices from communicating directly. Routing devices cannot go into sleep mode as they need to be available to relay network traffic. The Smart Plug plugs into an AC outlet and contains a power outlet that it can control. Light fixtures or other appliances can have their power controlled by the Smart Plug. It also contains a light sensor and temperature sensor.

**XBee Light/Temperature Sensor**

The Light and temperature sensor is an end-node device that does not forward messages like a routing device. End-node devices can go into sleep mode to conserve power. It is powered by two AA batteries, and if configured to sleep when not sending updates to the
network, it should be able to function on those batteries for a year. Low power consumption is one of the major goals of ZigBee’s End Nodes.

**ConnectPort X4 Configuration and Local Website**

The ConnectPort X4 is configured by uploading a dia.yml file (See Appendix C for the complete dia.yml file) and rebooting the device. The following changes were made to this file and their results can be seen in the lower section of Figure 11 in the “transform0” section:

- The following configuration change produces the average room temperature based on the data collected from both switch and sensor temperature sensors. The result is converted to the Fahrenheit scale.
- The c[x] naming convention is used in the expression (expr) line to represent the user defined channels (device.sensor).

- name: "Average Sensor Temp in F"
  unit: "F"
  channels:
    - "rpm0.temperature"
    - "sensor0.temperature"
  expr: "((int)(((c[0] + c[1]) / 2) * 9) / 5) + 32"

- The following changes test the switch’s light sensor to determine whether the light level brightness is greater than 40.
- The sensor reports brightness on a scale from 0 to 1201.

- name: "Switch_Low_Light"
  unit: "boolean"
  channels:
    - "rpm0.light"
  expr: "c[0] < 40"

- The following changes test the sensor’s light sensor to determine whether the light level brightness is greater than 40.
- The sensor reports brightness on a scale from 0 to 1201.

- name: "Sensor_Low_Light"
unit: "boolean"
channels:
  - "sensor0.light"
expr: "c[0] < 40"

- The following changes compute an AND on the results from the previous two light sensor tests to determine if the overall light level in the room is below 40.
- Originally, this test was used to turn on and off a light connected to the Smart Plug’s power outlet, based on the brightness in the room. That functionality was removed, but the test was left to demonstrate how the results from other tests can be used to create more complicated logic.

- name: "Room_Low_Light"
  unit: "boolean"
  channels:
    - "transform0.Switch_Low_Light"
    - "transform0.Sensor_Low_Light"
  expr: "c[0] & c[1]"

The following website is hosted by the ConnectPort X4 device and is accessible at http://(ConnectPort X4 IP Address)/idigi_dia.html and it shows the above changes in the transform0 section of the website.

![Web Configuration](image)

Figure 11 – Sample “idigi_dia.html” Local Website
Embedded System

The iDigi Rabbit RCM6700 system was chosen for this project because it contains all of the connectivity needed to allow a security system to interact with external hardware: sensors and alarm lights, as well as the ability to connect to the Internet to report status. There are also versions of Rabbit that connect to the Internet via cellular phone communication. For security systems, most companies now use cellular phone connections instead of wired connections. This reduces a potential weakness in the system’s ability to communicate in times of emergency.

Rabbit® MiniCore® RCM6700

![Figure 12 – Rabbit MiniCore RCM6700](image)

The Rabbit MiniCore RCM6700 includes the following features:

- Rabbit 6000 running up to 200 MHz
- 10/100 Ethernet and pin-compatible with MiniCore 802.11b/g wireless modules
- 1 MB of internal RAM for program storage
- 1 MB of serial Flash for data logging or web page storage
- Available iDigi Application Development Services
- Secure, anywhere management using iDigi Manager Pro
Attaching the RCM6700 to the lower interface board provides AC power, programmable I/O ports, one user input button, and one programmable LED. Attaching the RCM56/57XX Digital I/O board includes port replication of the lower interface board as well as 4 user input buttons and 4 programmable LED lights that are used for direct user interaction with the Rabbit system.

![Image of assembled Rabbit with IO board attached]

**Figure 13 – Assembled Rabbit with IO board attached**

**Digi Rabbit: Dynamic C®**

The iDigi Rabbit comes with a modified version of C++ called Dynamic C. It is well suited to creating state machine applications on embedded systems like the Rabbit. Once the I/O ports have been configured, the application’s main function contains a loop that the Rabbit remains in as it reacts to changes in the external hardware and requests from the Internet. During the loop, the developer can use the costate command to create states of their state machine based on global variable values. The below pseudo code shows the syntax of this main loop. It is important to keep the costate sections of code simple to
avoid one of them taking up all of the system’s resources or starving the other costate functions.

While(TRUE)
{
  Costate
  {
    if (StateA)
      DoSomethingA();
    else
      DoSomethingElseA();
  }
  Costate
  {
    if (StateB)
      DoSomethingB();
    else
      DoSomethingElseB();
  }
}

As these embedded systems tend to be located where it is difficult to access them, these main loops usually contains code to deal with system status requests and system reboot requests.
External Hardware

Using the I/O ports of the iDigi Rabbit to communicate with external hardware requires those ports to be defined as either input or output ports. In the example below, the port’s registry is written to in order to define the function of those pins on the I/O board.

```c
// Set Port E pins 0, 1 to switch inputs
// Set Port C pins 0, 1 to switch outputs
BitWrPortI(PEDDR, &PEDDRShadow, 0, 0);
BitWrPortI(PEDDR, &PEDDRShadow, 0, 1);
BitWrPortI(PCDDR, &PCDDRShadow, 1, 0);
BitWrPortI(PCDDR, &PCDDRShadow, 1, 1);
```

The I/O port C pins 0 and 1 have been connected to a light. The light is made to flash off and on with the following code:

```c
BitWrPortI(PCDR, &PCDRShadow, 0, 1);// Light off
waitfor(DelayMs(50));
BitWrPortI(PCDR, &PCDRShadow, 1, 1);// Light on
waitfor(DelayMs(50));
```

Reed Switch – Door and Window Sensor

A reed switch is being utilized as a sensor to determine if a door or window is opened or closed. For a security system, one half of the switch’s plastic housing would be mounted on the door and the other half to the door frame. One of the plastic housings contains a magnet and the other contains a two position circuit with three exterior screws attached to contacts within the housing. When the two plastic housings are placed within 1” of each other, the magnetic field causes the contacts to move, and the screw for the closed position is connected to the live screw. When the two plastic housings are moved apart by more than 1”, the lack of the magnetic field causes the contacts to move again, and the
screw for the opened position is connected to the live screw. The three screws of the reed switch are hardwired to three ports on the RCM56/57XX Digital I/O board that is attached to the Rabbit daughterboard. On the Rabbit, connection 1 (the live screw) is attached to a 3.3V port; connection 2 (the opened screw) is attached to PE0 port; and connection 3 (the closed screw) is attached to PE1 port. By monitoring the bits of ports PE0 and PE1, the Rabbit can determine the status of the switch.

The following code allows the Rabbit to monitor the reed switch and update the variable DoorStatus accordingly.

```c
// Monitor door switch
if (BitRdPortI(PEDR, 0) && DoorStatus == CLOSE) // Wait for the door to open.
{
    waitfor(DelayMs(50)); // Wait 50 milliseconds for switch bounce to clear.
    if (BitRdPortI(PEDR, 0)) // If the switch is still pressed, change state.
    {
        DoorStatus = OPEN;
        ...
    }
}
```

A 50 millisecond delay is used to cope with switch bounce, the flutter of a mechanical switch as it changes states from opened to closed. The program waits for the bounce to clear and then checks the value of the switch again before changing the value of the DoorStatus variable.

The DoorStatus variable is consulted again when the user attempts to arm the system, as it will not arm while the door is open. Once the system is armed, the door status is used to determine whether someone has entered the home and now must enter the security code.
Swan – PIR Motion, Light and Alarm System

The Swan security system was chosen for its simple but wide variety of security devices. The system contains a light sensor, motion detector, two speakers, two LED lights, mode switch and activation remote. The light sensor determines the amount of ambient light on the face of the unit. The motion detector detects motion in a horizontal arc of 100º in front of the system and a vertical arc of 70º with a range of 3’ to 18’ of detection when mounted between 4’ to 6’ from the ground. The two speakers are capable of generating an alarm sound of 100 dB when activated. However, due to the voltage change when driving the speakers with the Rabbit’s 3.3V port, the volume is significantly lower. In a production unit, the required 6V needed to drive the speakers would be utilized, producing the 100 dB levels. There is one white LED light and one red LED light. A toggle switch is used to change the system mode from off to alarm and light modes. The remote is used to activate the system when placed in alarm mode with the toggle switch.

Lead wires have been soldered to the speakers and two LED lights. These wires are attached to ports on the Rabbit’s IO board. When the Rabbit goes into Alarm mode, the white LED will be used to determine if the motion detector has become active. When the Rabbit is in AlarmHome mode, the motion detector will be ignored. If an intrusion is detected, the user is given the chance to enter their security code. If they fail to do so in the given time, the system goes into Call the Police mode, and the speakers and red LED light are activated. See Appendix A for the complete application.
Chapter 4

Cloud Environments

iDigi® Device Cloud

The iDigi Device Cloud is part of Amazon Elastic Compute Cloud (Amazon EC2). When you purchase the iDigi Gateway Development Kit, you may create a free iDigi Device Cloud account that includes iDigi Manager Pro to monitor up to 5 devices, with unlimited iDigi Web Services and 30 iDigi SMS per month.

![iDigi Device Cloud - iDigi Manager Pro](image)

**Figure 14 – iDigi Device Cloud – iDigi Manager Pro**

iDigi Manager Pro displays all registered devices with their network information and connection status. Figure 14 shows an example of two connected devices: The Rabbit RCM6700 at IP Address 192.168.1.205 and the ConnectPort X4 at IP Address 192.168.1.214. Devices can be assigned to Group Paths. Two groups were created and would represent the devices at a client’s home.
Figure 15 shows all of the individual XBee devices attached to the network. Device information is displayed here including Gateway Device ID, Product Type, and Role.

![Image of XBee Networked Devices](image)

**Figure 15 – iDigi Device Cloud – XBee Networked Devices**

Six different alarm types may be assigned to monitor and respond to changes in a device’s status. The alarm types include excessive disconnections, data point condition match, and offline detection. A security company configuring a client’s devices could define an alarm to watch for excessive device network disconnections. If this alarm is triggered, it could mean the user’s security system has gone offline, possibly indicating that a break-in is in progress, and that the customer should be contacted by a company representative.

Figure 16 shows the parameters that can be customized for the excessive disconnects alarm.

![Image of Edit Alarm](image)

**Figure 16 – iDigi Device Cloud – Add Device Alarms**
If a device detaches from the network for more than a minute, the alarm is triggered. The trigger will be reset once the device is reconnected to the network for five minutes. On the Admin tab of the website, notifications for both triggered alarms and reset alarms can be created to define how the system will respond when an alarm or reset has been triggered.

API Explorer

On the Documentation tab of the iDigi Device Cloud website, an API Explorer may be used to customize, test and transmit commands to devices attached to the network. In the example below, the command to request device status from a ConnectPort X4 is shown.

![API Explorer](image.png)

**Figure 17 – API Explorer**

The Send button immediately sends the command to the user defined Device ID and the response is shown in the Web Services Response window. Below is a partial listing of the UML response generated by the ConnectPort X4 device to the status request commands. It shows all of the statistics and settings for the device.
<sci_reply version="1.0">
<send_message>
<device id="00000000-00000000-00409DFF-XXXXXXXX">
<rci_reply version="1.1">
<query_state>
<device_info>
<mac>XX:XX:XX:XX:XX:XX</mac>
<product>ConnectPort X4</product>
<model>ConnectPort X4</model>
<digiproduct>ConnectPort X4</digiproduct>
...
</device_info>
</query_state>
<serial_stats></serial_stats>
<net_stats></net_stats>
<device_stats>
<cpu>26</cpu>
<uptime>614068</uptime>
<datetime>Sun Nov 24 02:53:57 2013 UTC</datetime>
<adjdatetime>Sat Nov 23 19:53:57 2013</adjdatetime>
...
</device_stats>
<boot_stats>
<dhcp>on</dhcp>
:autoip>on</autoip>
<static>off</static>
<ip>192.168.1.207</ip>
...
</boot_stats>
<device_registry></device_registry>
<ppp_stats></ppp_stats>
<zigbee_state></zigbee_state>
<camera_stats></camera_stats>
</query_state>
</rci_reply>
</device>
</send_message>
</sci_reply>
In addition, the website may generate Python code to execute these commands. In the below example, the status request command from Figure 17 has been generated. The code creates a secure connection to the target device, requests status, and processes the response.

**Exported as Python**

```python
# The following lines require manual changes
username = "YourUsername" # enter your username
password = "YourPassword" # enter your password
# Nothing below this line should need to be changed
# -----------------------------------------------
import httplib
import base64

# create HTTP basic authentication string, this consists of
# "username:password" base64 encoded
auth = base64.encodestring("%s:%s"%(username,password))[:-1]

# message to send to server
message = """"<sci_request version="1.0">
    <send_message cache="false">
        <targets>
            <device id="00000000-00000000-00409DFF-XXXXXXXX"
        </targets>
        <rci_request version="1.1">
            <query_state/>
        </rci_request>
    </send_message>
</sci_request>
"""

webservice = httplib.HTTP("my.idigi.com",80)

# to what URL to send the request with a given HTTP method
webservice.putrequest("POST", "/ws/sci")

# add the authorization string into the HTTP header
webservice.putheader("Authorization", "Basic %s\%auth")
```
webservice.putheader("Content-type", "text/xml; charset=UTF-8")
webservice.putheader("Content-length", "%d" % len(message))
webservice.endheaders()
webservice.send(message)

# get the response
statuscode, statusmessage, header = webservice.getreply()
response_body = webservice.getfile().read()

# print the output to standard out
print (statuscode, statusmessage)
print response_body
Google Appspot.com Cloud

The iDigi Gateway Development Kit contains a sample website that is hosted on the Google Appspot.com Cloud at: http://rf-gateway.appspot.com. The source code is also available for download and provides an excellent example of how to interact with network connected XBee devices over the Internet.

![Image of the iDigi Gateway Development Kit website]

**Figure 18 – Sample iDigi Development Kit Website**

Once the user has connected and configured their iDigi Gateway Development Kit’s devices, they can use this sample website. The user logs onto their ConnectPort X4 with a user name, password, and the MAC address of their device. The website is divided into three sections. In the first section, the XBee Sensor’s brightness and temperature
readings are shown. In the second section, the XBee Smart Plug’s AC current reading is shown along with the control button for the plugs power outlet that may be turned on or off. In the last section, telemetry readings from both XBee Sensor and XBee Smart Plug are shown as a graph with historical data.

A security system could give users access to their installed devices by developing a similar website.

**Google Chart**

The iDigi Gateway Development Kit website utilizes Google Charts to display the results of the sensor readings. A hyperlink passes the parameters of the desired chart to the chart application, and the chart is returned as an image file. The below sample chart is from the iDigi Gateway Development Kit website.

![Sample Google Chart](image)

**Figure 19 – Sample Google Chart's**

The following URL was used to create this chart:

http://chart.apis.google.com/chart?cht=gom&chd=t:2.97333333333&chs=400x200&chco=444444,FF3333&chl=0.446%20A&chtt=Current&chxt=y&chxr=0,0,15.0,5

The parameters are defined here:  cht = Chart Type, chd = Chart Dimensions, chs = Chart Scale, chco = Chart Color Scale, chl = Chart Data Point Label, chtt = Chart Title, chxt = Chart X Axis Text, chxr = Chart X Axis Range.
The Google Appspot.com website also contains a comprehensive tutorial that walks the user through the process of developing, structuring and deploying a Python application onto the Appspot.com host. The tutorial includes:

- How to build an App Engine application using Python.
- How to use the webapp2 web application framework.
- How to use the App Engine datastore with the Python modeling API.
- How to integrate an App Engine application with Google Accounts for user authentication.
- How to use Jinja2 templates with your app.
- How to upload your app to App Engine using Google App Engine Launcher.

**Google App Engine Launcher**

The Google App Engine Launcher is free software provided by Google to create a local host environment during development of Python applications and a publishing tool for deploying applications to the AppSpot.com website.
Chapter 5

Rabbit Security System

Developed on the iDigi Rabbit embedded system, the security system connects to the iDigi Cloud at startup and will respond to commands sent from the API Explorer. The user may interact with the system via the four buttons on the I/O board. The screen shot below is from the iDigi Dynamic C IDE during execution. The terminal window is used during development for feedback from the system as it changes states.

![Screen Shot of IDE during Execution](image)

Figure 21 – Rabbit remotely responding to iDigi Cloud commands

Sending the follow command from the iDigi Cloud’s API Explorer will cause the Rabbit’s armed status to be toggled:

```
<sci_request version="1.0">
  <send_message>
    <targets>
```

36
<device id="00000000-00000000-0090C2FF-XXXXXXXX"/>
</targets>
<rci_request version="1.1">
<do_command target="SystemArm">
<code index="1">1</code>
<code index="2">2</code>
<code index="3">3</code>
<code index="4">4</code>
<code index="5">1</code>
</do_command>
</rci_request>
</send_message>
</sci_request>

When the Rabbit receives the request, the do_command target contains the function name to be called of SystemArm. When action_SystemArm is called, it confirms the submitted password from the code index parameter and, if valid, toggles the value of AlarmStatus. The system cannot be armed while the door is open and returns error code -2 if this occurs. If the password is invalid, error code -1 is returned and no change is made to the system. A value of 0 is returned if the system is disarmed and a value of 1 is returned if the system is armed.

Local commands may be entered using the four buttons on the I/O board. Button 1 represents Arm/Disarm the system. By pressing this button, the system enters a state of waiting for the user to enter their security code. The user has 10 seconds to enter the correct code. Once in this state, the four buttons represent security code numbers 1 through 4. If the wrong code is entered, the user may try again until time has expired. If the system’s original state was disarmed and the user fails to enter the security code, the system ends the attempt and nothing else happens. If the system’s original state was armed and the user fails to enter the security code, the system go into the alarm state called Call the Police. In the Call the Police state, the siren is heard and the red light
flashes. To exit the Call the Police state, the user must press the 1 button again and successfully enter the security code. If the door is opened while entering the code, the attempt is ended as the system cannot be armed while the door is open. Button 2 represents Arm/Disarm in stay at home mode. Like Button 1 Arm/Disarm, the user must enter their security code but once successful, the system will be armed with the motion detector inactive.

If the system is armed in either mode and the door is opened, the user must successfully enter the security code or the system will go into Call the Police mode. If the system is armed in stay at home mode, the motion detector is ignored. However, in normal armed mode, if movement is detected, the system will require a user to enter the security code.

Figure 22 – Rabbit responding to local commands
Chapter 6

Conclusion

Given the technologies and hardware presented in this thesis, a security company could be established to provide customers with a subscription-based home or office automation and security system. The central control unit based on the iDigi Rabbit embedded system can be connected to a cloud-based network capable of communicating with all of the devices installed at a customer’s location. These connected devices can be monitored by the security company to allow them to respond to triggered events, and to update software remotely as needed. Customers can be given secure access to their system using any device capable of connecting to the Internet and displaying a webpage. This allows them to monitor and interact with their system from anywhere in the world. As more companies advertise and install these technologies, they will become less expensive. Customers will come to expect these devices in their daily life, much like smart phones and smart televisions; and more homes will have an automation system to control their living environments, access, and security.
References


2010.


Appendix A – Rabbit Security Controller Code

Digi’s Rabbit Dynamic C language was used to create this application for the Rabbit RCM 6700 embedded system. It allows the Rabbit to connect to the Internet, respond to commands and information requests sent over the Internet, control the attached hardware, and monitor sensors that are attached to it.

/*******************************************************************
 Rabbit Security Controller

 By Lee Hern

 California State University, Northridge

 Master of Computer Science

 Copyright 2013. All rights reserved.

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

// False = 0; True = Any non-zero value
// LED values: On = 0; Off = 1
// Button 1 = Set Arm/Disarm in Normal mode (with motion detection)
// Button 2 = Set Arm/Disarm in Home mode (no motion detection)

#define PORTA_AUX_IO // required to enable external I/O bus
#define CLOSE 0
#define OPEN 1
#define DISARMED 0
#define ARMED 1
#define FALSE 0
#define TRUE 1
#define DS1 4
#define DS2 5
#define DS3 6
#define DS4 7
#define S1 4
#define S2 5
#define S3 6
#define S4 7

// Global variables
int SecCodeSource[5];
int SecCode[5];
int SecCodeIndex = 0;
int AlarmStatus = DISARMED;
//int AlarmAtHome = FALSE;
int EnterCode = FALSE;
int DoorStatus = OPEN;
int LightStatus = OPEN;
int BreakIn = FALSE;
int Sound = FALSE;
int FlashingLight = FALSE;
int ErrorStatus = FALSE;
int FlashDelay2 = 50;
int FlashDelay3 = 50;
int FlashDelay4 = 50;

/******************************
Web responses specific code
******************************/
#define IDIGI_USE_ADDP     // Required to include ADDP support
#define IDIGI_PRODUCT "IDIGI_DO_COMMAND.C"
#define IDIGI_VENDOR "Digi International Inc."
#define IDIGI_VENDOR_ID "1234"
#define IDIGI_FIRMWARE_ID "1.01.00"
#define IDIGI_CONTACT "support@digi.com"
#define IDIGI_LOCATION "Planet Earth"
#define IDIGI_DESCRIPTION "Simple iDigi demo"
#define IDIGI_SERVER "sd1-na.idigi.com"

// Define an offset in the Rabbit UserID block at which to store network
// configuration data.
#define IDIGI_USERBLOCK_OFFSET 0
#define ADDP_PASSWORD"rabbit"

// Comment this out if the Real-Time Clock is set accurately.
#define X509_NO_RTC_AVAILABLE

// This prints interface status when it changes.
#define IDIGI_IFACE_VERBOSE

// Required because we're not using any static Zserver resources. This
// definition is required in any iDigi application which does not include
// an HTTP server.
#define SSPEC_NO_STATIC
#define "idigi.lib"
// System Arm/Disarm Via the Internet code
#define TARGET_NAME_1  "SystemArm"
#define REQ_VAR_NAME_1  SystemArmCode
#define REPLY_VAR_NAME_1  SystemArmResult

struct {
    int code[5];
} REQ_VAR_NAME_1;
#web REQ_VAR_NAME_1

struct {
    int result;
} REPLY_VAR_NAME_1;
#web REPLY_VAR_NAME_1

void action_SystemArm(void);
#web_update REQ_VAR_NAME_1 action_SystemArm

// prototype of SecCodeMatched()
int SecCodeMatched();

void action_SystemArm(void)
{
    int i, x;

    printf("\n\nSystem connected via online user account\n");
    printf("Validating security code:\n");

    for (i = 0; i < 5; i++)
    {
        SecCode[i] = REQ_VAR_NAME_1.code[i];
    }

    if (SecCodeMatched())
    {
        if (AlarmStatus)
        {
            printf("System Disarmed Remotely\n");
            AlarmStatus = DISARMED;
            //AlarmAtHome = FALSE;
            BreakIn = FALSE;
            Sound = FALSE;
            FlashingLight = FALSE;
            EnterCode = FALSE;
            REPLY_VAR_NAME_1.result = AlarmStatus;
        }
    }
}
else
{
    if (DoorStatus == CLOSE)
    {
        printf("System Armed Remotely\n");
        AlarmStatus = ARMED;
        //AlarmAtHome = FALSE;
        REPLY_VAR_NAME_1.result = AlarmStatus;
    }
    else
    {
        printf("\nCannot Arm system while door is open\n");
        REPLY_VAR_NAME_1.result = -2;
    }
}
else
{
    printf("Invalid Code\n");
    REPLY_VAR_NAME_1.result = -1;
}

/**
* Code to access the system over the Internet to ARM/DISARM
*
* <sci_request version="1.0">
*   <send_message>
*     <targets>
*       <device id="00000000-00000000-0090C2FF-FFE82AB0"/>
*     </targets>
*     <rci_request version="1.1">
*       <do_command target="SystemArm">
*         <code index="1">1</code>
*         <code index="2">2</code>
*         <code index="3">3</code>
*         <code index="4">4</code>
*         <code index="5">1</code>
*       </do_command>
*     </rci_request>
*   </send_message>
* </sci_request>
*
* Result: -2 = Door Open, -1 = Invalid Security Code, 0 = System Disarmed, 1 = System Armed
// Check System Arm/Disarm Status Via the Internet code
#define TARGET_NAME_2  "SystemStatus"
#define REQ_VAR_NAME_2 SystemStatusCode
#define REPLY_VAR_NAME_2 SystemStatusResult

struct {
  int code[5];
} REQ_VAR_NAME_2;
#web REQ_VAR_NAME_2

struct {
  int result;
} REPLY_VAR_NAME_2;
#web REPLY_VAR_NAME_2

void action_SystemStatus(void);
#web_update REQ_VAR_NAME_2 action_SystemStatus

void action_SystemStatus(void)
{
  int i;

  printf("\n\nSystem connected via online user account\n");
  printf("Validating security code:\n");

  for (i = 0; i < 5; i++)
  {
    SecCode[i] = REQ_VAR_NAME_2.code[i];
  }

  if (SecCodeMatched())
  {
    printf("Sending Alarm Status\n");
    REPLY_VAR_NAME_2.result = AlarmStatus;
  } else
  {
    printf("Invalid Code\n");
    REPLY_VAR_NAME_2.result = -1;
  }
}

/*******************************
Code to access the system over the Internet to see system status
*******************************
<sci_request version="1.0">
  <send_message>
    <targets>
      <device id="00000000-00000000-0090C2FF-FFE82AB0"/>
    </targets>
    <rci_request version="1.1">
      <do_command target="SystemStatus">
        <code index="1">1</code>
        <code index="2">2</code>
        <code index="3">3</code>
        <code index="4">4</code>
        <code index="5">1</code>
      </do_command>
    </rci_request>
  </send_message>
</sci_request>

Result: -1 = Error, 0 = System Disarmed, 1 = System Armed
*******************************************
End of Web responses specific code
*******************************************

void InitIO()
{
    // Initialize ports for read and write

    // Set Security Code
    SecCodeSource[0] = 1;
    SecCodeSource[1] = 2;
    SecCodeSource[2] = 3;
    SecCodeSource[3] = 4;
    SecCodeSource[4] = 1;

    // Make Port B pins for switch inputs
    BitWrPortI(PBDDR, &PBDDRShadow, 0, S1);
    BitWrPortI(PBDDR, &PBDDRShadow, 0, S2);
    BitWrPortI(PBDDR, &PBDDRShadow, 0, S3);
    BitWrPortI(PBDDR, &PBDDRShadow, 0, S4);

    // Set Port A pins for LEDs low
    BitWrPortI(PADR, &PADRShadow, 1, DS1);
    BitWrPortI(PADR, &PADRShadow, 1, DS2);
    BitWrPortI(PADR, &PADRShadow, 1, DS3);
    BitWrPortI(PADR, &PADRShadow, 1, DS4);
// Make Port A bit-wide output
BitWrPortI(SPCR, &SPCRShadow, 1, 2);
BitWrPortI(SPCR, &SPCRShadow, 0, 3);
BitWrPortI(PCDR, &PCDRShadow, 1, 0);

// Set Port E pins 0, 1 to switch inputs
// Set Port C pins 0, 1 to switch outputs
BitWrPortI(PEDDR, &PEDDRShadow, 0, 0);
BitWrPortI(PEDDR, &PEDDRShadow, 0, 1);
BitWrPortI(PCDDR, &PCDDRShadow, 1, 0);
BitWrPortI(PCDDR, &PCDDRShadow, 1, 1);

void SecCodeReset()
{
    int k;

    SecCodeIndex = 0;
    for (k = 0; k < 5; k++)
    {
        SecCode[k] = 0;
    }
}

int SecCodeMatched()
{
    int j;
    int Matched = TRUE;

    for (j = 0; j < 5; j++)
    {
        if (SecCode[j] != SecCodeSource[j])
        {
            Matched = FALSE;
            break;
        }
    }
    return Matched;
}

main()
{
    /**************************************************
Web responses specific code
***********************************************************************/

int rc, i, x, y;
if (idigi_init())
    exit(1);

// Register do_commands
#define MKS(x) #x
#define MKSTRING(x) MKS(x)

idigi_register_target(TARGET_NAME_1, MKSTRING(REQ_VAR_NAME_1), MKSTRING(REPLY_VAR_NAME_1));
idigi_register_target(TARGET_NAME_2, MKSTRING(REQ_VAR_NAME_2), MKSTRING(REPLY_VAR_NAME_2));

/***********************************************************************/
End of Web responses specific code
***********************************************************************/

InitIO();
SecCodeReset();

while (TRUE)
{
    _restart:
        rc = idigi_tick();
        if (rc)
        {
            printf("Final rc = %d\n", rc);
            if (rc == -NETERR_ABORT)
            {
                // RCI reboot request was received. Normally we would use this
                // to shut down cleanly and reboot the board.
                printf("Rebooting via exit(0)!\n");
                exit(0);
            }
            goto _restart;
        }

    // External Connection Controls
costate
    { // Monitor door switch
        if (BitRdPortI(PEDR, 0) && DoorStatus == CLOSE)
        {
            waitfor(DelayMs(50));          // Wait for switch bounce
            if (BitRdPortI(PEDR, 0))
{  DoorStatus = OPEN;
   printf("Door Open\n");
   BitWrPortI(PADR, &PADRShadow, 0, DS1);  // LED on
   if (AlarmStatus == ARMED && !BreakIn)
      EnterCode = TRUE;
 }
}

if (BitRdPortI(PEDR, 1) && DoorStatus == OPEN)
{
   waitfor(DelayMs(50));                  // Wait for switch bounce
   if (BitRdPortI(PEDR, 1))
   {
      DoorStatus = CLOSE;
      printf("Door Closed\n");
      BitWrPortI(PADR, &PADRShadow, 1, DS1);  // LED off
      if (AlarmStatus == DISARMED)
         ErrorStatus = FALSE;
   }
}

/* costate
   {  // Motion detection switch
      if (AlarmStatus == ARMED && AlarmAtHome == FALSE)
      {
         waitfor(BitRdPortI(PCDR, 5));
         if (AlarmStatus == ARMED && !BreakIn)
         {
            printf("Motion Detected");
            EnterCode = TRUE;
         }
      }
   } */
}
costate
{  // Controls Siren
   if (Sound)
   {
      BitWrPortI(PCDR, &PCDRShadow, 0, 1);  // Speaker on
      waitfor(DelayMs(0));
      BitWrPortI(PCDR, &PCDRShadow, 1, 1);  // Speaker off
      waitfor(DelayMs(0));
   }
   else
      BitWrPortI(PCDR, &PCDRShadow, 1, 1);  // Speaker off
costate
{
    // Controls Flashing Light
    if (FlashingLight)
    {
        BitWrPortI(PCDR, &PCDRShadow, 0, 0); // Light on
        waitfor(DelayMs(50));
        BitWrPortI(PCDR, &PCDRShadow, 1, 0); // Light off
        waitfor(DelayMs(50));
    }
    else
    {
        BitWrPortI(PCDR, &PCDRShadow, 1, 0); // Light off
    }
}

// Control Rabbit board buttons

// Monitor Button 1 = Arm/Disarm System with Motion detection
waitfor(BitRdPortI(PBDR, S1));
waitfor(DelayMs(50));                // Wait for switch unpressed
if (!BitRdPortI(PBDR, S1))          // Unpressed switch detected
{
    if (EnterCode)
    {
        printf("1");
        SecCode[SecCodeIndex++] = 1;
    }
    else
    {
        //AlarmAtHome = FALSE;
        EnterCode = TRUE;
    }
}
}

// Monitor Button 2 = Arm Home System without Motion detection
waitfor(BitRdPortI(PBDR, S2));
waitfor(DelayMs(50));                // Wait for switch unpressed
if (!BitRdPortI(PBDR, S2))          // Unpressed switch detected
{
    if (EnterCode)
    {
        printf("2");
        SecCode[SecCodeIndex++] = 2;
    }
else
{
    //AlarmAtHome = TRUE;
    EnterCode = TRUE;
}
}
}

costate
{
    // Monitor Button 3 = xxxx
    waitfor(BitRdPortI(PBDR, S3));
    waitfor(DelayMs(50));  // Wait for switch unpressed
    if (!BitRdPortI(PBDR, S3))  // Unpressed switch detected
    {
        if (EnterCode)
        {
            printf("3");
            SecCode[SecCodeIndex++] = 3;
        }
    }
}
}

costate
{
    // Monitor Button 4 = xxxx
    waitfor(BitRdPortI(PBDR, S4));
    waitfor(DelayMs(50));  // Wait for switch unpressed
    if (!BitRdPortI(PBDR, S4))  // Unpressed switch detected
    {
        if (EnterCode)
        {
            printf("4");
            SecCode[SecCodeIndex++] = 4;
        }
    }
}
}

// LED Controls    LED 1 used to show door status
costate
{
    // Flash LED 2: Flash fast for EnterCode; slow for xxxx
    if (EnterCode)
    {
        if (EnterCode)
        {
            FlashDelay2 = 50;
        }
        else
        {
            FlashDelay2 = 250;
        }
    }
}

BitWrPortI(PADR, &PADRShadow, 0, DS2); // LED on
waitfor(DelayMs(FlashDelay2));
BitWrPortI(PADR, &PADRShadow, 1, DS2); // LED off
waitfor(DelayMs(FlashDelay2));
}
else
BitWrPortI(PADR, &PADRShadow, 1, DS2); // LED off
}

costate
{
  // Flash LED 3: Flash fast for BreakIn; slow for door OPEN when ARMING
if (ErrorStatus || BreakIn)
{
  if (BreakIn)
    FlashDelay3 = 50;
  else
    FlashDelay3 = 250;

BitWrPortI(PADR, &PADRShadow, 0, DS3); // LED on
waitfor(DelayMs(FlashDelay3));
BitWrPortI(PADR, &PADRShadow, 1, DS3); // LED off
waitfor(DelayMs(FlashDelay3));
}
else
BitWrPortI(PADR, &PADRShadow, 1, DS3); // LED off
}

costate
{
  // Flash LED 4: Flash fast for ARMED HOME; slow for ARMED
if (AlarmStatus)
{
  if (!AlarmStatus)
    FlashDelay4 = 200;
  else
    FlashDelay4 = 400;

BitWrPortI(PADR, &PADRShadow, 0, DS4); // LED on
waitfor(DelayMs(FlashDelay4));
BitWrPortI(PADR, &PADRShadow, 1, DS4); // LED off
waitfor(DelayMs(FlashDelay4));
}
else
BitWrPortI(PADR, &PADRShadow, 1, DS4); // LED off
}
costate
{  // Enter Code Timer - 10 seconds to enter code
    if (EnterCode)
    {
        SecCodeReset();
        printf("\nEnter Code: ");

        for (i = 0; i < 100; i++)
        {
            if (AlarmStatus == DISARMED && DoorStatus == OPEN)
            {
                printf("\nCannot Arm system while door is open\n");
                EnterCode = FALSE;
                ErrorStatus = TRUE;
                break;
            }
            else if (SecCodeIndex == 5)
            {
                if (SecCodeMatched())
                {
                    EnterCode = FALSE;
                    if (AlarmStatus == DISARMED)
                    {
                        if (TRUE)  // Test AlarmAtHome to send correct text
                            printf("\nAlarm Armed\n");
                        else
                            printf("\nAlarm Armed with Motion Detection On\n");
                        AlarmStatus = ARMED;
                    }
                    else if (AlarmStatus == ARMED)
                    {
                        printf("\nAlarm Disarmed\n");
                        AlarmStatus = DISARMED;
                        //AlarmAtHome = FALSE;
                        BreakIn = FALSE;
                        Sound = FALSE;
                        FlashingLight = FALSE;
                    }
                    break;
                }
            }
            else
            {
                printf("\nError:  Enter code again...
");
                SecCodeReset();
            }
        }
    }
}
else
{
    if (i % 10 == 0 && EnterCode == TRUE)
    {
        printf(".");
    }
    waitfor(DelayMs(100));
}

if (EnterCode)
{
    if (AlarmStatus == ARMED)
    {
        printf("\nCalling Police\n");
        Sound = TRUE;
        FlashingLight = TRUE;
        BreakIn = TRUE;
        EnterCode = FALSE;
    }
    else if (AlarmStatus == DISARMED)
    {
        printf("\nSystem Timed Out\n");
        EnterCode = FALSE;
    }
    }
Appendix B – ConnectPort X4 Update Code

The following code was used by iDigi ESP for Python to update the configuration of the ConnectPort 4X by loading a dia.yml configuration file (See Appendix C) and rebooting the device.

#!/usr/bin/python

#########################################################################
#                                                                             #
# Copyright (c)2008, 2009, Digi International (Digi). All Rights Reserved. #
#                                                                             #
# Permission to use, copy, modify, and distribute this software and its      #
# documentation, without fee and without a signed licensing agreement, is    #
# hereby granted, provided that the software is used on Digi products only   #
# and that the software contain this copyright notice, and the following   #
# two paragraphs appear in all copies, modifications, and distributions as #
# well. Contact Product Management, Digi International, Inc., 11001 Bren    #
# Road East, Minnetonka, MN, +1 952-912-3444, for commercial licensing      #
# opportunities for non-Digi products.                                    #
#                                                                             #
# DIGI SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED    #
# TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A           #
# PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY,   #
# PROVIDED HEREUNDER IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND.  #
# DIGI HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,           #
# ENHANCEMENTS, OR MODIFICATIONS.                                          #
#                                                                             #
# IN NO EVENT SHALL DIGI BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT,       #
# SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS,    #
# ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF    #
# DIGI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.                   #
#                                                                             #
To run this, use command line: python dia.py [config.yml]

# imports
import sys
try:
    import os.path
except ImportError:
    # os comes from python.zip on our gateways.
    print "Unable to import 'os' module. Check that your system has a 'python.zip' file, and that it is the correct one for your device."
    sys.exit(1)

import gc
import time

# constants
BOOTSTRAP_VERSION = "2.1.1.2"
GC_COLLECTION_INTERVAL = 60 * 15 # fifteen minutes

# internal functions & classes

def spin_forever(core):
    """ This routine prevents the main thread from exiting when the framework is run directly from __main__.
    """
    try:
        while not core.shutdown_requested():
            collected_items = gc.collect()
            if collected_items:
                # only print if something accomplished
                print ('GarbageCollector: collected %d objects.' % collected_items)
            core.wait_for_shutdown(GC_COLLECTION_INTERVAL)
    finally:
        core._shutdown()
        print "dia.py is exiting...
"

def setup_path_and_zip():
""" Sets up the paths to import from the appropriate locations. Also detects if a zip file is present and returns the path to the archive """

#Does the dia.zip exist in our local directory?
expected_zip_path = os.path.join(os.path.abspath('.'), 'dia.zip')
if os.path.exists(expected_zip_path):
    #Yes, add it to the path:
    sys.path.append(expected_zip_path)
    #Add the paths internal to the zip file
    for lib_path in ['lib', 'src']:
        sys.path.insert(0, os.path.join(expected_zip_path, lib_path))

    return expected_zip_path
else:
    #We may be operating in an environment that doesn't need dia.zip
    #find files, like /src/core/core_services.py
    if not os.path.exists(os.path.join('src', 'core', 'core_services.py')):
        raise RuntimeError("Unable to find dia.zip or core libraries", please load dia.zip and try again")
    else:
        #We're running in the root of the source tree, directly add the
        #/src and /lib directories, insert them, so they are imported
        #before base libraries, in case of conflict
        for lib_path in ['lib', 'src']:
            sys.path.insert(0, lib_path)

        return None

def locate_configuration_file(expected_zip_path):
    """Locates the settings file, returns the settings in a file like object,
    returns the source name of the settings (the file name used),
    and returns the destination of where the settings could be saved.

    The source and destination will differ if the source is the dia.zip file.
    """

    settings_file = None
    settings_flo = None
    dest_file = None

    #To locate the settings file, parse command line
    if sys.argv and len(sys.argv) > 1:
        #Use the one supplied by the args
        settings_file = sys.argv[1]
if not os.path.exists(settings_file):
    # Try again for NDS, doesn't have concept of local directory
    settings_file = os.path.abspath(settings_file)
if not os.path.exists(settings_file):
    raise RuntimeError("Settings file: %s given, but not found" " in path" %(settings_file))

# Settings file found, read it in
settings_flo = open(settings_file, 'r')
dest_file = settings_file
else:
    # No direction from args to find settings file
    # Search local path for dia.pyr, dia.yml in order

    for possible_fname in ['dia.pyr', 'dia.yml']:
        if os.path.exists(os.path.abspath(possible_fname)):
            dest_file = settings_file = os.path.abspath(possible_fname)
            settings_flo = open(settings_file, 'r')
            break

if (expected_zip_path is not None) and (settings_flo is None):
    # Previous attempts at finding the configuration are unsuccessful
    # Search the dia.zip archive for the settings file, dia.pyr/dia.yml

    import zipfile
    import StringIO

    dia_zip_flo = open(expected_zip_path, 'rb')
    dia_zip = zipfile.ZipFile(dia_zip_flo)

    # Find the fname, then break
    for possible_fname in ['dia.pyr', 'dia.yml']:
        if possible_fname in dia_zip.namelist():
            # Set the flo object to the file buffer in the zip file
            # Provide an alternative location, we cannot overwrite the
            # configuration in the zipfile
            settings_file = os.path.join(expected_zip_path, possible_fname)
            dest_file = os.path.join(os.path.abspath('.'), possible_fname)
            settings_flo = StringIO.StringIO(dia_zip.read(possible_fname))
            dia_zip_flo.close()
            dia_zip.close()
            break

return settings_file, settings_flo, dest_file
def do_slowdown_check():
    """ Performs the slow down check and if true, slows down the startup of the Dia. This is done to prevent platforms that have an auto restart on exit of the Dia from spinning fast enough to prevent modification of the platform if something causes the startup to fail immediately. """

    # Check for the file that enables this feature
    stop_fname = os.path.join(os.path.abspath('.'), "nospin.txt")
    if not os.path.exists(stop_fname):
        print("Dia auto-detect of rapid reboot DISABLED")
        return
    print("Dia auto-detect of rapid reboot ENABLED")

    # Continue, check the timestamp file for entries
    slowdown = False
    
    ts_file = open(stop_fname, 'r')
    entries = ts_file.readlines()
    ts_file.close()

    ## Remove all non-float compatible entries
    for ent in entries[:]:
        try:
            float(ent)
        except ValueError:
            entries.remove(ent)

    # If more than 9 entries, find average of last 10 entries
    if len(entries) >= 10:
        curr_time = time.time()
        avg_time = sum(float(x.strip()) for x in entries[-10:]) / 10.0

        # Compare them to the current time, and if less than 20 minutes
        # mark us ready for slow down
        diff_time = curr_time - avg_time
        if diff_time < 1200:
            print("Initiating slow down in order to prevent the Dia from "
                  "spinning too fast")
            slowdown = True

    # Cycle out old timestamps
    entries.append(str(time.time()) + os.linesep)
    ts_file = open(stop_fname, "w")
ts_file.writelines(entries[-10:])

# If need to slow down, do so here
if slowdown:
    print "Slowing down, pausing for 10 minutes"
    time.sleep(600)
    print "Done slowing down."

def main():
    
        Acts as the startup script for the Dia. Sets up the environment (sys.path) and loads the configuration file for use by the core services to load the rest of the system

        
    
    # Perform slow down check in case of reboot cycle
do_slowdown_check()

    # File name of the settings being used
settings_file = None

    # File object derived from opening the settings file
settings_flo = None

    # If using a zip file, will return path to zip
expected_zip_path = setup_path_and_zip()

    # We've found the library files, verify matching version
    try:
        from src.common.dia_version import DIA_VERSION
    except Exception:
        if expected_zip_path:
            print ("Error reading from Zipfile: %s Common cause for error" "is that the files inside the zip file were compiled with " "the incorrect version of python." %expected_zip_path)
        else:
            print ("No dia.zip found and unable to locate the file ", "src/common/dia_version.py in the local path.")
        raise

if DIA_VERSION != BOOTSTRAP_VERSION:
    raise RuntimeError("Library files found, but bootstrap and library " "versions do not match! Expected: %s, Found: %s" %((BOOTSTRAP_VERSION, DIA_VERSION)))

61
# Locate the settings file that we use for starting up
settings_file, settings_flo, dest_file = \n    locate_configuration_file(expected_zip_path)

if settings_file is None or settings_flo is None:
    raise RuntimeError("Unable to locate settings file in local directory,"
           " from command line input, or dia.zip. Please "
           "provide a configuration file in one of these "
           "locations.")

print "Running in environment: %s" % sys.platform
print "iDigi Device Integration Application Version %s" % DIA_VERSION
print "Source settings file: %s" % settings_file
print "Destination settings file: %s" % dest_file

from core.core_services import CoreServices

core = CoreServices(settings_flo=settings_flo,
                     settings_filename=dest_file)

if __name__ == "__main__":
    # Don't exit. If not __main__ then the caller needs to guarantee this.
    spin_forever(core)

return core

if __name__ == "__main__":
    main()
Appendix C – ConnectPort X4 Configuration File

The following dia.yml file was generated to configure the iDigi Gateway Kit hardware: ConnectPort X4, XBee Smart Plug, and XBee Sensor. Once this file is loaded, the ConnectPort X4 reboots and the locally hosted website can be seen at the address defined in the web0 section near the end of this file. In this example, it is defined as idigi_dia.html.

```
devices:
  - name: xbee_device_manager
driver:
devices.xbee.xbee_device_manager.xbee_device_manager:XBeedeviceManager

  - name: edp_upload0
driver: devices.edp_upload:EDPUpload
settings:
  interval: 5
  sample_threshold: 20
  collection: "collection_name"
  filename: "file_name"

  - name: sensor0
driver: devices.xbee.xbee_devices.xbee_sensor:XBeesensor
settings:
  xbee_device_manager: "xbee_device_manager"
  extended_address: "00:13:a2:00:40:a6:17:c5!"
  sample_rate_ms: 5000

  - name: rpm0
driver: devices.xbee.xbee_devices.xbee_rpm:XBeerpm
settings:
  xbee_device_manager: "xbee_device_manager"
  extended_address: "00:13:a2:00:40:6e:43:67!"
  sample_rate_ms: 5000

  - name: transform0
driver: devices.transforms_device:TransformsDevice
settings:
  - name: "TempF"
    unit: "F"
    channels:
      - "rpm0.temperature"
```
- "sensor0.temperature"
  expr: "(((c[0] + c[1]) / 2) * 9) / 5 + 32"
- name: "Switch_Low_Light"
  unit: "boolean"
  channels:
    - "rpm0.light"
      expr: "c[0] < 40"
- name: "Sensor_Low_Light"
  unit: "boolean"
  channels:
    - "sensor0.light"
      expr: "c[0] < 40"
- name: "Room_Low_Light"
  unit: "boolean"
  channels:
    - "transform0.Switch_Low_Light"
    - "transform0.Sensor_Low_Light"
  expr: "c[0] & c[1]"

presentations:
- name: rci
  driver: presentations.rci.rci_handler:RCIHandler

- name: console0
  driver: presentations.console.console:Console

- name: web0
  driver: presentations.web.web:Web
  settings:
    page: "idigi_dia.html"

tracing:
  default_level: "warning"
  default_handlers:
    - "stderr"
Appendix D – Google Appspot Tutorial Code

The following code was created based on the Google Appspot Tutorials. It is currently running at http://guestbook-hern.appspot.com/. The process taught in the tutorial would be used to publish a user interface for connection to the home security system.

```python
import cgi
import datetime
import urllib
import webapp2
import jinja2
import os
from google.appengine.ext import db
from google.appengine.api import users

JINJA_ENVIRONMENT = jinja2.Environment(
    loader=jinja2.FileSystemLoader(os.path.dirname(__file__)))

class Greeting(db.Model):
    """Models an individual Guestbook entry with author, content, and date."""
    author = db.StringProperty()
    content = db.StringProperty(multiline=True)
    date = db.DateTimeProperty(auto_now_add=True)

def guestbook_key(guestbook_name=None):
    """Constructs a Datastore key for a Guestbook entity with guestbook_name."""
    return db.Key.from_path('Guestbook', guestbook_name or 'default_guestbook')

class MainPage(webapp2.RequestHandler):
    def get(self):
        guestbook_name=self.request.get('guestbook_name')
        greetings_query = Greeting.all().ancestor(guestbook_key(guestbook_name)).order('date')
        greetings = greetings_query.fetch(10)
        if users.get_current_user():
            url = users.create_logout_url(self.request.uri)
            url_linktext = 'Logout'
        else:
            url = users.create_login_url(self.request.uri)

        self.response.out.write(jinja2.Environment(loader=self.jinja_loader).get_template('index.html').render(greetings=greetings, url=url, url_linktext=url_linktext))

```

65
url_linktext = 'Login'

template_values = {
    'greetings': greetings,
    'url': url,
    'url_linktext': url_linktext,
}

template = JINJA_ENVIRONMENT.get_template('index.html')
self.response.write(template.render(template_values))

class Guestbook(webapp2.RequestHandler):

def post(self):
    # We set the same parent key on the 'Greeting' to ensure each greeting
    # is in the same entity group. Queries across the single entity group
    # will be consistent. However, the write rate to a single entity group
    # should be limited to ~1/second.
    guestbook_name = self.request.get('guestbook_name')
    greeting = Greeting(parent=guestbook_key(guestbook_name))

    if users.get_current_user():
        greeting.author = users.get_current_user().nickname()

    greeting.content = self.request.get('content')
    greeting.put()

    query_params = {'guestbook_name': guestbook_name}
    self.redirect('/?%s' % urllib.urlencode(query_params))

app = webapp2.WSGIApplication([('/', MainPage),
    ('/sign', Guestbook),
    debug=True)