Programmable Computer Remote

Silicon Samurai

Kyle Brown  
John Vogel  
Luke Wilson

March 28, 2008

Abstract: Though remote control devices for computer programs are commercially available, all are either limited-functionality or special-purpose. The PCR (Programmable Computer Remote) is designed as a general-purpose remote control for computer programs. It accomplishes this through use of keyboard shortcuts.
# Table of Contents

Introduction ................................................................................................................................................3
  Problem Statement .................................................................................................................................3
  Features .................................................................................................................................................3
Design ........................................................................................................................................................3
  Hardware ...............................................................................................................................................5
  Firmware .............................................................................................................................................6
  Software ...............................................................................................................................................6
  Alternative Designs ................................................................................................................................9
Conclusion ...............................................................................................................................................10
Appendix A: Parts List and Cost Analysis ...............................................................................................11
  Parts List ..............................................................................................................................................11
  Cost Analysis - Prototype ..................................................................................................................12
  Cost Analysis - Production Model ....................................................................................................13
Appendix B: Communications Protocol Specification .............................................................................14
Appendix C: Keypress Sequence Codes ..................................................................................................16
Appendix D: References ..........................................................................................................................18
Appendix E: Datasheets ...........................................................................................................................19

# Table of Figures

Figure 1: PCR System Architecture .........................................................................................................4
Figure 2: Remote Circuit ............................................................................................................................5
Figure 3: Class Diagram ............................................................................................................................7
Figure 4: PCR Menu ..................................................................................................................................8
Figure 5: PCR Menu with Configurations .................................................................................................8

# Index of Tables

Table 1: Cost Analysis - Prototype ...........................................................................................................12
Table 2: Cost Analysis - Production Model .............................................................................................13
Introduction

If one handheld remote can control your entire entertainment center, from television to sound system to room lights, why should there not be one handheld device that can run many different programs on your computer? This is the purpose of the Programmable Computer Remote (PCR). A wireless, handheld device with eight configurable buttons, the PCR is capable of performing any action on a computer that can be executed with a short sequence of keypresses.

Problem Statement

This project was first conceived in a meeting with Dr. Brian Moore of the University of Nebraska-Lincoln Music Department. He explained that Music Department faculty often have to give presentations in class which may include not only presentation slides but also audio applications such as iTunes®. While giving such presentations, they would prefer to be free to move around the room and interact with students; however, in order to control the software, they need to remain at their computers.

Features

The PCR has the following features:

- Wireless
  - 50 meter range (Appendix E, ez430-RF2500 User Guide, page 12)
  - Not dependent on line-of-sight (unlike infrared technologies)
- Approximately 13 hour battery life
- Visible in dark environments
- Creates keyboard events in the host OS on button presses
  - Events are configurable
  - Configuration can be performed without an extra connection

Design

The design of the PCR can be separated into three major categories:

- Hardware - the physical components that make up the device
- Firmware - the software running on the microcontrollers
- Software - the software running on the host computer
Figure 1 illustrates how the major components of the system are organized. Software makes up the computer side of the system, the firmware bridges the computer and remote by enabling communications between the two microcontrollers, and the hardware makes up the remote itself.

On the computer side, the Driver is responsible for both receiving commands from the "Middleman" and sending new configurations to it, while the Configuration Program is responsible for parsing configuration files and providing the parsed configuration data to the Driver for uploading. The Operating System executes keystrokes when the Driver receives a command from the remote and calls the WSH Script.

According to the original design, the "Middleman" is only responsible for transmitting commands to the computer and configurations to the remote. However, due to limitations in the available RF communication libraries, sending configuration data to the Remote Microcontroller proved to be impossible. Therefore, the "Middleman" stores configuration data itself, and when the Remote indicates that a certain button has been pressed, the "Middleman" looks up the corresponding command and sends it to the Driver.

The LCD on the Remote is intended to provide dynamic labels for each button that change with each configuration. Due to the previously mentioned limitation, however, it cannot be reconfigured and only displays a constant set of labels. The Remote Microcontroller periodically polls the Buttons, and if any button has been pressed, a notification is sent to the "Middleman" identifying which button was pressed.

---

1 Radio Frequency
2 Liquid Crystal Display
Hardware

The following components comprise the PCR:

- TI ez430-RF2500 development kit
  - 2 development boards with MSP430F2274 microcontroller and CC2500 RF transceiver
  - 1 USB³ programmer board with virtual serial interface
  - 2 AAA batteries to power one development board
- SparkFun SerLCD v2.5
- One 5 volt LDO⁴ voltage regulator
- Four AA batteries
- Eight buttons and a power switch

The PCR prototype was assembled on a solderless development board with each of the eight buttons connected to one input terminal on the Remote Microcontroller. As illustrated in Figure 2, the Remote Microcontroller is connected directly by wire to the two AAA batteries that power it, and the LCD is connected to its serial output line. The 5 volt LDO reduces the 6 volts provided by the four AA batteries to 5 volts, the level required to power the LCD. A power switch is included between the batteries and LDO so the LCD can be switched on and off more easily. An additional 3.3 volt LDO was purchased to eliminate the need for the AAA batteries and allow the switch to control the entire device, but it failed to meet its design specifications (see Appendix E), limiting voltage to only 3.8 volts, which is in excess of the microcontroller's 3.6 volt maximum (see Appendix E).

---

³ Universal Serial Bus
⁴ Low Dropout
**Firmware**

All firmware on both microcontrollers is written in C using the IAR Embedded Workbench, an IDE included with the RF2500 development kit.

Communication between the two microcontrollers is achieved through use of the SimpliciTI libraries. These libraries employ a simple API including these functions:

- `SMPL_Init()` initializes the RF controller
- `SMPL_Link()` and `SMPL_LinkListen()` establish the connection
- `SMPL_Send()` and `SMPL_Receive()` transfer data
- `SMPL_IOCTL()` alters connection settings

The communication code on both microcontrollers is based on a sensor network example available on TI's website (http://www.ti.com/litv/zip/slac139b). This example includes the only binary distribution of the SimpliciTI libraries. Unfortunately, this binary distribution of the SimpliciTI libraries supports only one-way communication; therefore, the Remote Microcontroller can send data to the "Middleman" Microcontroller, but the "Middleman" cannot send to the Remote. Two-way communication would be possible if the SimpliciTI source was recompiled for it. However, even though the SimpliciTI source code is available, it compiles to an object larger than 4 kilobytes, and the version of the IAR Workbench included with the RF2500 development kit will not compile object code larger than 4 kilobytes [6].

Originally, the "Middleman" was to run simple software that sent everything it received from the Remote Microcontroller to the computer and everything from the computer to the Remote. The Remote Microcontroller would store its configuration in two global arrays - one for button actions and one for button labels. It would run in a non-terminating loop, checking for a new incoming configuration and for new button presses on every iteration. If it detected an incoming configuration, it would parse that configuration and store it in the global configuration arrays. If it detected a button press, it would look up the action string corresponding to the pressed button in the appropriate configuration array and send that string to the "Middleman."

However, due to the inadequacy of the available SimpliciTI library, it is impossible to send any configuration data to the Remote Microcontroller. To work around this limitation, the firmware on the "Middleman" was altered to parse incoming configurations itself and extract the action strings, storing them in the global array of actions which was originally stored on the Remote Microcontroller. The Remote Microcontroller's firmware was altered to send only the number of the button pressed rather than looking up the appropriate action string. Thus, when the "Middleman" receives a button number, it sends the corresponding action string from its array. The array of labels remains in the Remote Microcontroller's firmware, but it cannot be updated except by altering the firmware.

**Software**

All software is written in Java, except for the WSH script that is used to emulate keypresses. The Java software is combined in a single program which includes the functionality of the driver and the configuration program. It appears to the user as an icon in the system tray.
Class-Level Description

The computer software for the PCR includes six classes.

Figure 3 shows how the classes interact. The *Main* class is where execution starts for the program. Immediately upon startup, it creates an instance of *SerialConnection* to establish a connection to the virtual serial interface of the "middleman" microcontroller. *SerialConnection*, in turn, creates an instance of *KeyDriver* to listen for commands from the remote and call the WSH script when necessary. If the user instructs the software to upload a new configuration to the remote, *Main* creates an instance of *ConfigFile* to parse the configuration file. This in turn creates * AppConfig* objects, which may create *ButtonConfig* objects to store parts of the configuration data during the parsing process. Once the file has been parsed, *Main* extracts the parsed data from *ConfigFile* and sends it using *SerialConnection*. 
Operation

When launched, the PCR software appears as an icon in the system tray. Right-clicking on this icon will cause a menu to appear, as in Figure 4.

Selecting the "Load Config" option from the menu will launch a file browser dialog. The user then selects a configuration file using this dialog. Once a valid configuration file is selected, the program will parse it and add more options to the menu, as shown in Figure 5.

An "Upload" option is shown for each configuration stored in the selected file. When the user selects one of these options, the corresponding configuration is uploaded to the "Middleman." If any errors occur during this process, they will appear as pop-up bubbles.

Configuration files for the PCR are structured as follows: A line beginning in "PCR.configs" lists the names of all configurations in the file, separated by commas. Each configuration consists of 16 lines, 2 for each button. The first line defines the button's name, and the other defines the action associated with it. For example, the following line sets the label of button 0 in the "PP" configuration to "PLY":

```
PCR.PP.btn[0].Name = "PLY"
```

Whereas this line sets button 5 in the "IT" configuration to correspond to the Enter key:

```
PCR.IT.btn[5].Action = "{Enter}" 
```
All communication strings sent between the program and remote begin with an 8-bit number from 0 to 5 (known as the message type) and end with a tab character ('\t'). Two message types are currently supported: Send Config (2) and Receive Button Action (5). The format for Send Config messages supports multiple configurations, separated by carriage return characters ('\r'), but the device currently only reads the first configuration sent. Each individual configuration is composed primarily of eight "button configs", which include the label and action string for each button, separated by a line feed character ('\n'). Receive Button Action messages contain the action string only. For more information, see Appendix B.

The format of action strings matches that used in WSH, so that the very same string used throughout the system to represent a particular key sequence can be sent directly to the WSH script for it to emulate the desired key sequence. In this format, most keypresses representable by an ASCII character (e.g. letters and numbers) are represented by that character, other keys such as tab and enter are represented by name ({TAB}, {ENTER}), and keys that are used in combination with others are represented by a symbol (e.g. %=alt). If such a key is used in a combination, the key it is used with is enclosed in parentheses (e.g. shift-tab = +(%{TAB})) unless the key sequence is enclosed in quotes (e.g. shift-tab = "+%{TAB}"). For more information, see Appendix C.

**Alternative Designs**

Numerous alternative designs were considered but rejected, some because the team had insufficient resources to realize them and others because they were impractical or carried too great a risk.

**Original Alternative Design**

At the outset of the development process, a choice was made between two designs for the system. The rejected design featured an EZ430-F2013 development board and a Bluetooth keyboard. The keyboard would have been disassembled and the microcontroller attached directly to its circuitry to emulate keypresses. This design would have made the PCR fully Bluetooth enabled and would have eliminated the need for third-party drivers on the host computer. However, the PCR would not have been configurable unless physically connected to the host computer and reprogrammed. Furthermore, the number of microcontroller outputs necessary to connect to the keyboard was unknown, and project success depended too strongly on the microcontroller's ability to drive the keyboard.

**Production Design**

The version of the PCR developed during this process is a prototype, and the team recommends certain design changes for a production version of the PCR.

First, the device should use Bluetooth for communication rather than the CC2500 transceiver used in the EZ430-RF2500 development board. This would allow the device to be used with any Bluetooth-enabled computer without extra hardware (such as the "middleman" microcontroller). It would also avoid the limitations in the SimpliciTI libraries that made two-way communication impossible.

Second, the WSH script used to emulate keypresses should be eliminated. This could be accomplished by using the Java Robots class library in the existing driver, but ideally the driver component of the PCR software should be eliminated entirely. This could be accomplished by designing the device to be recognized as a standard Bluetooth keyboard. However, a reduced-functionality version of the driver may still be necessary to enable wireless configuration of the device. With the "middleman" and WSH script eliminated, the PCR would work on multiple operating system platforms.
**Enhancements**

Potential enhancements to the PCR were considered by the team but rejected due to resource constraints.

- The Java Robots class could be employed in the driver component of the software to completely replace the WSH script. This would bring the software closer to cross-platform operability.

- The remote cannot recall its configuration from the last time it was used, so it will have to be reconfigured each time it is powered on. This could be changed if by altering the firmware to store the configuration in the microcontroller's Flash memory.

- If the remote could recall its previous configuration, it could also copy its configuration to the host computer. This would simplify the task of moving a configuration among different computers.

- The remote is only capable of storing one configuration at a time. If a user needed to use two different sets of commands (for example, controlling PowerPoint and iTunes during the same lecture), he or she would need to go to the computer and use the configuration program for each switch. A keyboard shortcut could be defined to signal the configuration program to upload a new configuration to the remote, and the user could assign the corresponding command string to a button and use that button to change configurations.

- In the PCR software, the serial port on which to find the "middleman" microcontroller is specified either in a configuration file or as a command line argument. Therefore, the user must be aware of the port number the "middleman" is currently using and specify it for the software. With more research into the abilities of the serial communication libraries used by the driver, the software could be redesigned to automatically detect which port the "middleman" is using. The communications protocol already includes query/response messages to make a search for the device possible (see Appendix B), but both the driver and "middleman" would need to be reprogrammed to execute the query and response.

**Conclusion**

The PCR has great potential to create new methods for even nontechnical users to control computers. The simple concept of keyboard shortcuts on which it is based makes it a versatile device, capable of countless uses beyond those for which it was designed.

The prototype of the PCR developed by the team provides proof that this concept of a versatile, programmable remote control device is feasible. With configurable labels, a more attractive outer case, and more mainstream technologies such as Bluetooth, this device could compete with many advanced computer remote control devices on the market - at a competitive price.
Appendix A: Parts List and Cost Analysis

Parts List
The following components were acquired and used for this project:

- 1 ez430-RF2500 Development Kit
- 1 16x2 Character Serial LCD: from www.sparkfun.com
- 1 5V LDO Voltage Regulator: from Fairchild Semiconductor (www.fairchildsemi.com)
- 8 Pushbutton Switches: from All Electronics Corp. (www.allelectronics.com)
- 10 Connector Pins: from the UNL Electrical Engineering Shop

The following components were acquired but not used:

- 1 3.3V LDO Voltage Regulator: from Fairchild Semiconductor (www.fairchildsemi.com)
- 1 3.3V LDO Voltage Regulator (SMT\textsuperscript{5}): from Fairchild Semiconductor (www.fairchildsemi.com)
- 1 24x2 Character LCD from All Electronics Corp. (www.allelectronics.com)
- 1 8-Bit Shift Register from www.sparkfun.com

Datasheets for all components acquired are available in Appendix E. Trivial components - the pushbutton switches and connector pins - are excluded.

\textsuperscript{5} Surface-Mount Technology
**Cost Analysis - Prototype**

The cost to develop the PCR prototype was well within the $500 budget, as detailed in Table 1.

<table>
<thead>
<tr>
<th>Part</th>
<th>Cost to Team</th>
<th>Retail Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ez430 Development Kit</td>
<td>(on loan)</td>
<td>$49[4]</td>
</tr>
<tr>
<td>Serial LCD</td>
<td>$26.60</td>
<td>$24.95</td>
</tr>
<tr>
<td>5V Regulator</td>
<td>$8.44</td>
<td>$1.15</td>
</tr>
<tr>
<td>Switches</td>
<td>$5.50</td>
<td>$2</td>
</tr>
<tr>
<td>Connector Pins</td>
<td>$1</td>
<td>$1</td>
</tr>
<tr>
<td><strong>Prototype Cost</strong></td>
<td><strong>$78.10</strong></td>
<td></td>
</tr>
<tr>
<td>3.3V Regulator</td>
<td>$15.32</td>
<td>$0.74</td>
</tr>
<tr>
<td>3.3V SMT Regulator</td>
<td>$8.31</td>
<td>$1.02</td>
</tr>
<tr>
<td>24x2 LCD</td>
<td>$8.50</td>
<td>$5</td>
</tr>
<tr>
<td>Shift Register</td>
<td>$3.15</td>
<td>$1.50</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td><strong>$76.82</strong></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: Cost Analysis - Prototype*

The Cost to Team column includes shipping and handling charges and represents the total amount of money spent by the team. The Retail Price column represents the value of the parts used in the prototype.
Cost Analysis - Production Model

In a mass-production scenario, an individual microcontroller would be used rather than a pre-packaged development kit, and Bluetooth communications would replace the CC2500 RF transceiver and SimpliciTI libraries (as well as the "Middleman"). The cost to produce such a device is estimated in Table 2.

<table>
<thead>
<tr>
<th>Component</th>
<th>Projected Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRF6150 Bluetooth Transceiver</td>
<td>$4 [7]</td>
</tr>
<tr>
<td>Character LCD</td>
<td>$5</td>
</tr>
<tr>
<td>Voltage Regulator</td>
<td>$1</td>
</tr>
<tr>
<td>Buttons</td>
<td>$2</td>
</tr>
<tr>
<td>Device Housing and PCB</td>
<td>$4[9]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$19.49</strong></td>
</tr>
</tbody>
</table>

Table 2: Cost Analysis - Production Model

At a production cost of $19.49, the device could retail at a price of $30 to $40. This compares favorably with universal remote control devices for entertainment centers, which can cost as much as $300. [8]
Appendix B: Communications Protocol Specification

Last Revised: 03/02/08—John Vogel

MsgType is a 1 byte id for distinguishing packet payloads
The ‘\t’ character is the separator that signifies the end of the packet.

**Button Action Format**

MsgType +Actiont1 + ’\t’ + MsgType +Actiont2 + ’\t’ + MsgType + Actiontn + ’\t’ + …

Actiontn is the key command (i.e. “% ({TAB} {TAB}) ”).

**Config String Format**

MsgType+#n+#a+AF1+AF2+AF3+AF4+…+AF n+’\t’

where #a (1 unsigned byte) and n is the number of Application Configs being sent*
    #n (1 unsigned byte) is an index from 0 to n-1 of the starting App Config*
    AF is the Application Config Format defined as follows:

ACName +’\r’ + #b+BF1+BF2+BF3+BF4+…+BF m+’\r’

where #b (1 unsigned byte) and m is the number of Button Configs being sent*
    ACName is the Application Config Name
    BF is the Button Config Format defined as follows:

Namei+’\n’+Actioni+’\n’

where i is button it refers to the current Button Config
    Name is limited to 3 characters due to LCD size restrictions
    Action is limited to 14 characters due to microcontroller memory restrictions

For the above sequences:
    Application Configs will appear in the same order as in the config file
    Button Configs will appear in identifier order (i.e. btn0,btn1,btn2,btn3,etc)

**Message Query Format**

MsgType+’\t’
**MsgTypes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>MQF Query Device: Is it the PCR?</td>
<td>(Proposed)</td>
</tr>
<tr>
<td>0x01</td>
<td>MQF Query Device: Yes, it is the PCR</td>
<td>(Proposed)</td>
</tr>
<tr>
<td>0x02</td>
<td>CSF Send Config</td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td>CSF Receive Config</td>
<td>(Proposed)</td>
</tr>
<tr>
<td>0x04</td>
<td>MQF Request: Send Config</td>
<td>(Proposed)</td>
</tr>
<tr>
<td>0x05</td>
<td>BAF Receive Button Action</td>
<td></td>
</tr>
</tbody>
</table>

Notes: The entire config will be uploaded all the time.

Reasons:
1. It is the simplest way to handle uploading the config.
2. We may want to support a recursive configuration.
3. The user may want to change the config on multiple different computers. Currently this would require a separate means (i.e. USB key) to transfer the config. We could transfer the config from the remote to the server on startup to facilitate this.

This protocol is meant to be expandable.
The remote should read the entire config and store it on the remote, but not necessarily use all of it.

* All numerical values may not be larger than 8 because a value of 9 or higher may be recognized as a delimiter (\t, \n, or \r) and would interfere with parsing of the message.
Appendix C: Keypress Sequence Codes
03/26/08 – John Vogel

All keypress codes should be surrounded by quotes to ensure that the command is interpreted correctly.

Any key that can be represented as a character (that isn’t reserved) is represented as a character sequence. For example: asdf@assdf*345435 is represented as “asdf@assdf*345435”.

<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{{}</td>
</tr>
<tr>
<td>}</td>
<td>{}}</td>
</tr>
<tr>
<td>[</td>
<td>{[}</td>
</tr>
<tr>
<td>]</td>
<td>{]}</td>
</tr>
<tr>
<td>(</td>
<td>{()}</td>
</tr>
<tr>
<td>)</td>
<td>{)}</td>
</tr>
<tr>
<td>~</td>
<td>{~}</td>
</tr>
<tr>
<td>+</td>
<td>{+}</td>
</tr>
<tr>
<td>^</td>
<td>{^}</td>
</tr>
<tr>
<td>%</td>
<td>{%}</td>
</tr>
<tr>
<td>Alt</td>
<td>%</td>
</tr>
<tr>
<td>Crl</td>
<td>^</td>
</tr>
<tr>
<td>Shift</td>
<td>+</td>
</tr>
<tr>
<td>Enter</td>
<td>{ENTER}</td>
</tr>
<tr>
<td>Backspace</td>
<td>{BACKSPACE}, {BS}, or {BKSP}</td>
</tr>
<tr>
<td>Break</td>
<td>{BREAK}</td>
</tr>
<tr>
<td>Caps Lock</td>
<td>{CAPSLOCK}</td>
</tr>
<tr>
<td>Delete</td>
<td>{DELETE} or {DEL}</td>
</tr>
<tr>
<td>Up Arrow</td>
<td>{UP}</td>
</tr>
<tr>
<td>Down Arrow</td>
<td>{DOWN}</td>
</tr>
<tr>
<td>Left Arrow</td>
<td>{LEFT}</td>
</tr>
<tr>
<td>Right Arrow</td>
<td>{RIGHT}</td>
</tr>
<tr>
<td>End</td>
<td>{END}</td>
</tr>
<tr>
<td>Enter</td>
<td>{ENTER} or ~</td>
</tr>
<tr>
<td>Esc</td>
<td>{ESC}</td>
</tr>
<tr>
<td>Help</td>
<td>{HELP}</td>
</tr>
<tr>
<td>Home</td>
<td>{HOME}</td>
</tr>
<tr>
<td>Insert</td>
<td>{INSERT} or {INS}</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>Num Lock</td>
<td>{NUMLOCK}</td>
</tr>
<tr>
<td>Page Up</td>
<td>{PGUP}</td>
</tr>
<tr>
<td>Page Down</td>
<td>{PGDN}</td>
</tr>
<tr>
<td>Print Screen</td>
<td>{PRTSC}</td>
</tr>
<tr>
<td>Scroll Lock</td>
<td>{SCROLLLOCK}</td>
</tr>
<tr>
<td>Tab</td>
<td>{TAB}</td>
</tr>
<tr>
<td>F1</td>
<td>{F1}</td>
</tr>
<tr>
<td>F2</td>
<td>{F2}</td>
</tr>
<tr>
<td>F3</td>
<td>{F3}</td>
</tr>
<tr>
<td>F4</td>
<td>{F4}</td>
</tr>
<tr>
<td>F5</td>
<td>{F5}</td>
</tr>
<tr>
<td>F6</td>
<td>{F6}</td>
</tr>
<tr>
<td>F7</td>
<td>{F7}</td>
</tr>
<tr>
<td>F8</td>
<td>{F8}</td>
</tr>
<tr>
<td>F9</td>
<td>{F9}</td>
</tr>
<tr>
<td>F10</td>
<td>{F10}</td>
</tr>
<tr>
<td>F11</td>
<td>{F11}</td>
</tr>
<tr>
<td>F12</td>
<td>{F12}</td>
</tr>
<tr>
<td>F13</td>
<td>{F13}</td>
</tr>
<tr>
<td>F14</td>
<td>{F14}</td>
</tr>
<tr>
<td>F15</td>
<td>{F15}</td>
</tr>
<tr>
<td>F16</td>
<td>{F16}</td>
</tr>
<tr>
<td>Windows</td>
<td>{WINDOWS}*</td>
</tr>
<tr>
<td>Menu</td>
<td>{MENU}*</td>
</tr>
</tbody>
</table>

* Not yet supported.
Appendix D: References


Appendix E: Datasheets

Development Board (ez430-RF2500): See attached file "RF2500 users guide.pdf"

Microcontroller (MSP430F2274): See attached file "Microcontroller datasheet.pdf"

RF Transceiver (CC2500): See attached file "RF Transceiver datasheet.pdf"

5V LDO: See attached file "5V LDO datasheet.pdf"

3.3V LDO: See attached file "3.3V LDO datasheet.pdf"

3V LDO: See attached file "3.3V SMT LDO datasheet.pdf"

Serial LCD: See attached file "SerLCD Datasheet.pdf"

24x2 LCD: See attached file "LCD-113.pdf"

Shift Register: See attached file "Shift Register datasheet.pdf"