

# **Solar Door Panel**

**Team Sodopa**

**University of Utah**

**Computer Engineering Senior Project**

David Hurst

Tyson Hunt

Jeffrey Kelley

[david.hurst5@gmail.com](mailto:david.hurst5@gmail.com)

[tyswitz@hotmail.com](mailto:tyswitz@hotmail.com)

[scientist009@gmail.com](mailto:scientist009@gmail.com)

## Introduction

Door displays are used commonly throughout many business and schools, however, they are static and boring. Think of how nice it would be if someone could tap into their wireless network to update or change their door display that is powered from solar cells. The display uses a wireless radio for all communication, and can be accessed remotely. This allows the user to change their name, set the display to their office hours, work hours, picture logo, or anything else the user would like. Using wireless communication and being powered by solar panels, it is completely standalone, requires no maintenance, and best of all no recharging of batteries. Using solar energy helps reduce energy costs, is friendly to the environment, and it is available in most settings where the device could be located such as an office hallway. To further decrease the environmental impact, super-capacitors were used instead of lithium-ion batteries which contain toxic materials.

This project consists of several major components. A cholesteric LCD (ChLCD), an energy harvester to capture and store solar energy into a storage element, a micro-controller unit (MCU), which controls all data flow and energy usage, and a wireless radio are wall mounted outside of an office, or conference room. This energy harvester monitors energy availability and notifies the MCU of the ability to request an update for the display. Since an update takes the majority of energy that the storage element can hold, there is no point in trying an update unless the operation can finish. All the components chosen use as little energy as possible in order for the solar panel to supply enough energy in a reasonable amount of charging time and lighting conditions.

Another major piece of this project is the base station radio connects the user's computer to communicate with the wall mounted display nearby. The radio on both ends uses the 802.15.4 protocol, which has an indoor range of 80-140 feet, relatively low power requirements, and is readily available. This base station acts as a bridge between the host computer's USB port and the radio.

The software component of the project enables the user to change what is displayed, preview any changes before making an update, and view information general information about the display. The custom-designed web server serves an interface for the user to connect to, either locally or over the Internet. The user also has the option to integrate the display device with a Google calendar and enable automatic updates every day. This is especially useful for conference rooms that have a daily schedule.

## Functional Description

Figure 1 below shows how each component of the system is connected.

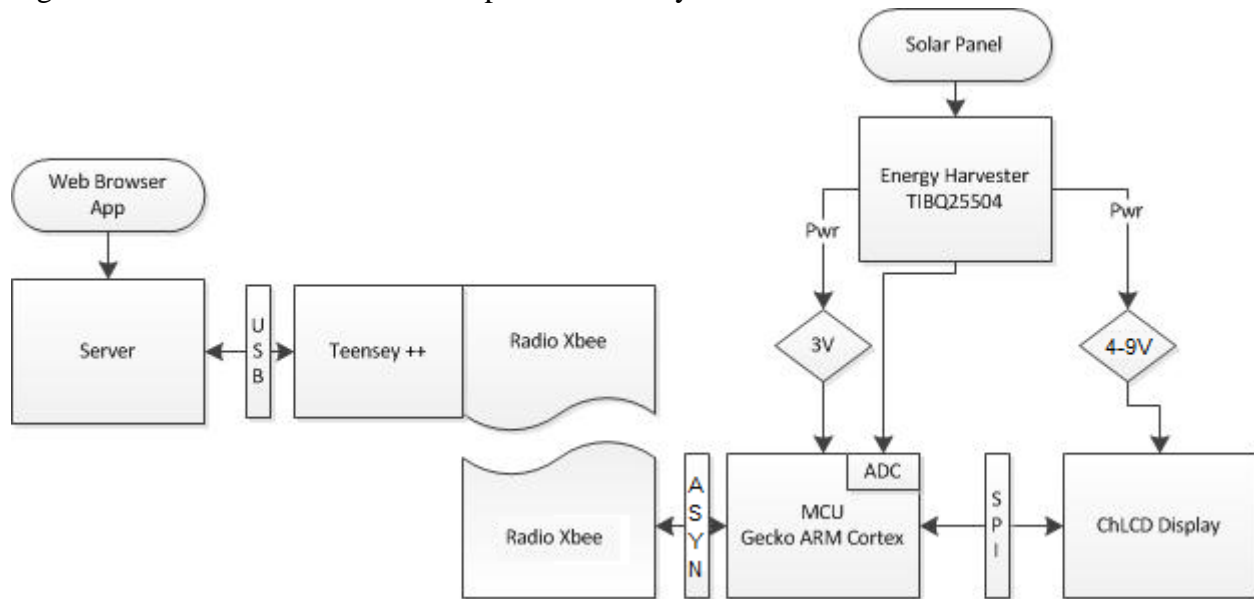


Figure 1 - Functional Design of the Solar Panel Display

A solar panel is connected to a custom-designed energy harvester board which charges the storage element, consisting of a bank of super capacitors. The device uses a bank of four 5.5V 1.5 Farad capacitors which have the capability of storing about 20 Joules of energy per capacitor. The same board supplies both 5V and 3.3V in order to power external peripherals including the MCU, XBEE Radio, and the display. It also monitors the voltage and lets the MCU know when the battery is full or empty, based on voltage levels of the 3.3 volt line. The minimum voltage level for operation is 3.1 volts which is considered empty by the Energy Harvester. The energy harvester board has the ability to turn off its own voltage regulators preventing serious discharge of the capacitor bank.

The MCU manages waking up and sleep settings for all of peripherals, including the radio and display. The radio uses asynchronous communication and the display uses a SPI serial interface. A radio dongle connected to the user's computer via USB acts as a bridge to communicate with the display. The software running on the computer sends and receives data to update the display and gather information about the device.

## Microcontroller (MCU)

The low energy microcontroller which can consume less than 6 $\mu$ A of current per second allows the display to operate without consuming large amounts of energy. The Gecko EFM32890gf128

starter kit was used. The MCU has a large enough memory to store the MCU program code and still be able to store data for the images needed for the Kent display. The MCU has a total of 128kB of flash memory. It has the ability to operate in different energy modes and has the ability to control many external devices through its USART, UART, TIMERS, RTC, and other features in the different energy modes as shown in Figure 3 and 4. Refer to the figure below for a layout of the MCU and energy modes the interfaces can operate in.

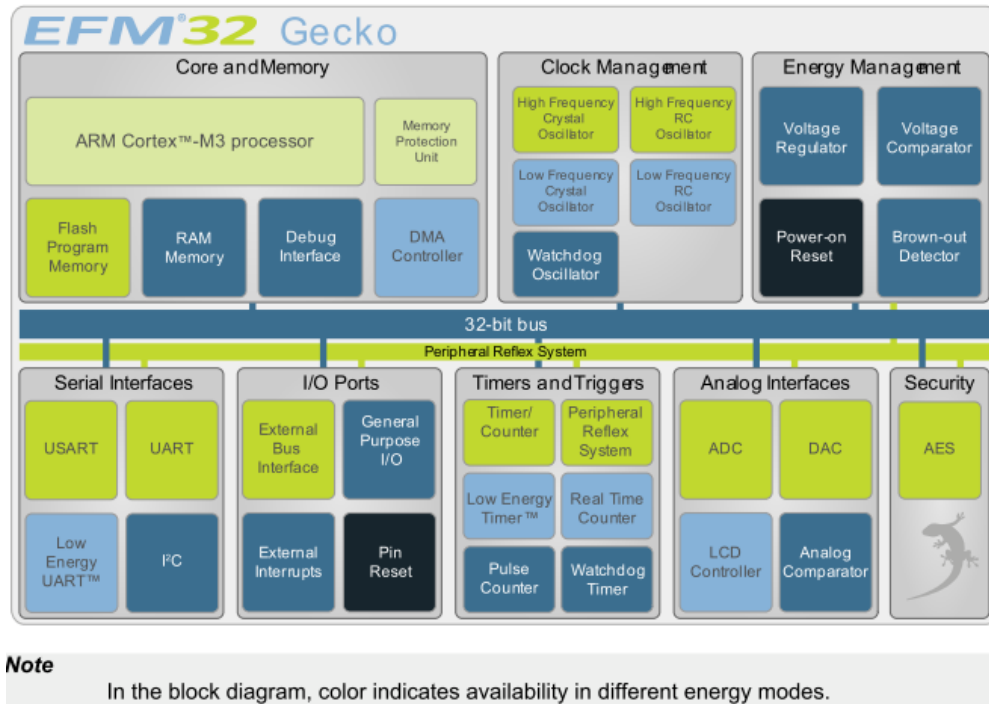


Figure 2 - MCU Board Layout (Energy Micro, 2011)

The energy modes of the MCU are called EM0, EM1, EM2, EM3 and EM4. The colors that represent each energy mode are shown in the figure below. EM0 mode is the normal operation of the MCU with all features enabled. EM1 disables the MCU and allows functionality for most of the device. The device was found to consume about 1.08mA in EM1 mode. EM1 is enabled automatically whenever the `__WFI()` instruction is called. This instruction waits for an interrupt from an internal component.

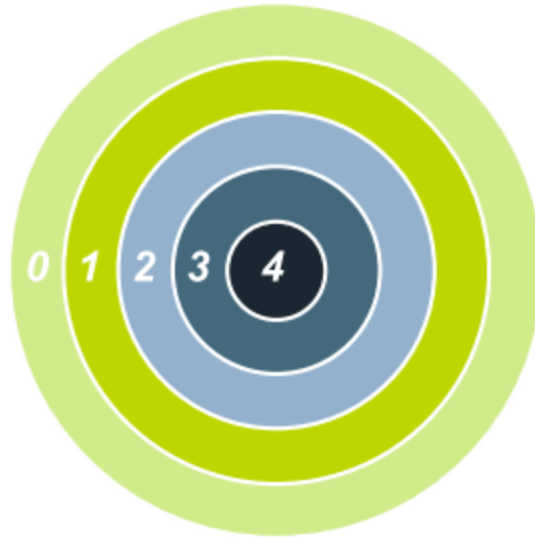


Figure 3 - Energy Modes of MCU (Energy Micro, 2011)

The list below shows the current consumption in each mode as seen on the data sheet (Energy Micro). The four energy saving power modes are shut off mode (EM4), stop mode (EM3), deep sleep (EM2), and sleep mode (EM1).

The Energy Micro EFM32g Reference Manual (2011) stated the following for energy consumption:

- 20 nA @ 3 V Shutoff Mode
- 0.6  $\mu$ A @ 3 V Stop Mode, including Power-on Reset, Brown-out Detector, RAM and CPU retention
- 0.9  $\mu$ A @ 3 V Deep Sleep Mode, including Real Time Clock with 32.768 kHz oscillator, Power-on Reset, Brown-out Detector, RAM and CPU retention
- 45  $\mu$ A/MHz @ 3 V Sleep Mode
- 180  $\mu$ A/MHz @ 3 V Run Mode, with code executed from flash (p. 5)

EM2 is the energy mode that is desired while the display device is asleep to consume the least amount of energy as possible. In this mode the high frequency clocks are disabled along with USART communication required for the Kent display. The problem with this mode is very few components are enabled which means the device must be woken up before resuming operation. Since the RTC timer is enabled in this mode it is used to wake the device up after the sleep period has elapsed. This period can be changed but was set to 4.6 hours. The RTC timer is used in multiple places to help reduce further energy requirements of needing a second timer. EM2 is activated by calling Enter\_EM2() or whenever the \_\_WFI() function is called as long as the CMU->CTRL->DEEP\_SLEEP register in the MCU is set to 1 first.

The other energy modes were not used since the device cannot be woken up unless a system reset is performed. The MCU handles all communication between the radio and the Kent display. It processes any commands received from the radio and processes them accordingly. The RF Radio Communication protocol explains the possible commands. Whenever a byte is received from the radio it appends it to the payload. A payload follows the following protocol. The first byte is the command. The next two bytes are the size of the payload for the command (set to 0 if there is no payload). The first byte (second byte received) is the lower byte of the size of the payload and the second byte (third byte received) is the upper byte of the size payload. The size of the payload is checked to determine how many more bytes are needed to receive the entire payload. The first 3 bytes, the command, lower byte of the size of the payload, and upper byte of the size of the payload, are all required. There may be no payload present (if size is set to 0).

To help reduce errors and determine if valid packets are received the two bytes that make up the size must follow the following convention. For the upper byte of the size, the most significant bit (0x80) must be set to 1 to show that it is not garbage. For the lower byte the least significant byte (0x01) must be set to 1 to show that it is valid. The remaining 14 bits determine the size of the payload. The top byte performs the operation  $TOP\_BYTE \& 0x7F$  and shifted right one bit to ignore the valid bit while the lower byte is shifted right one bit removing the valid bit. The remaining two bytes ( $TOP\_BYTE | BOTTOM\_BYTE$ ) once combined make up the total size of the payload assuming the top byte is already at an offset of one byte to the left.

After the timeout of the RTC timer is fired it begins to process the completed payload. If the packet has not been received in this time it is discarded. If the size value is corrupted it will either not receive enough of the packet or receive too much. This is the reason for using valid bits in the payload to help determine when errors occur. If the packet payload is full it will look at the packet and check that it has received a correct command and a valid size packet.

The MCU allows a timer wakeup from sleep mode or a timeout state if the radio times out (no response or takes too long to send packet). Another way to wake up the radio in case the user wants to update the display sooner than 4.6hours is via a manual switch. The switch will fire an internal GPIO interrupt and the MCU will process the interrupt waking it up disabling the RTC timer since it has already wakened up.

## **Kent Display**

This project utilizes a low power Cholesteric display that retains information when there is no power. This allows the device to be turned off and still show the information needed on the screen saving energy. When the display is in the idle state (sleep) it consumes 10 uA according to the data sheet as seen in Figure 4. The display operates at a logic level of 3.3V and requires a voltage of 4-9V for the power.

Parameter		Minimum	Typical	Maximum	Units
Logic Supply ( $V_{CC}$ ) <sup>1</sup>		3.0	3.3	3.6	VDC
Power Supply ( $V_{PWR}$ )		4.0	-	9.0	VDC
High Level Logic Output Voltage ( $V_{OH}$ )		$V_{CC}-0.6$	-	$V_{CC}$	VDC
Low Level Logic Output Voltage ( $V_{OL}$ )		$V_{SS}$	-	$V_{SS}+0.6$	VDC
High Level Logic Input Voltage ( $V_{IH}$ )		$0.8 \times V_{CC}$	-	$V_{CC}$	VDC
Low Level Logic Input Voltage ( $V_{IL}$ )		$V_{SS}$	-	$V_{SS}+0.6$	VDC
Average Operating Power @25°C (while driving image) <sup>2</sup>	$V_{CC}$	-	7	-	mW
	$V_{PWR}$	-	150	-	mW
Average Operating Power @60°C (while driving image) <sup>2</sup>	$V_{CC}$	-	7	-	mW
	$V_{PWR}$	-	234	-	mW
Standby Current <sup>2</sup>	$V_{CC}$	-	156	-	$\mu A$
	$V_{PWR}$	-	13	-	mA
Sleep Current <sup>2</sup>	$V_{CC}$	-	10	-	$\mu A$
	$V_{PWR}$	-	0	1	$\mu A$

Figure 4- Kent Characteristic Information (Kent Displays Incorporated)

Through experimentation it was found that the device drew about 4.3mA in an idle state (while awake) and about 4.9mA while driving the image. The MCU in idle mode was found to consume 1mA with only the CMU clock enabled in EM0. This shows that the Kent display used about 3.9mA of current while driving an image.

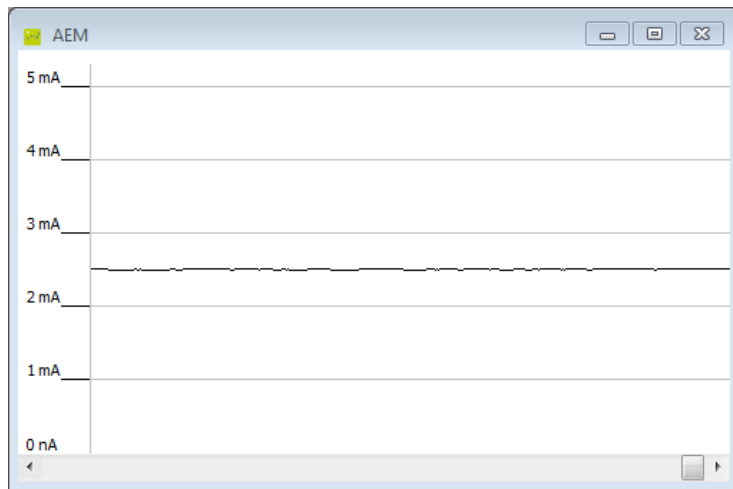


Figure 5 - Kent Display Sleep mode Current Usage

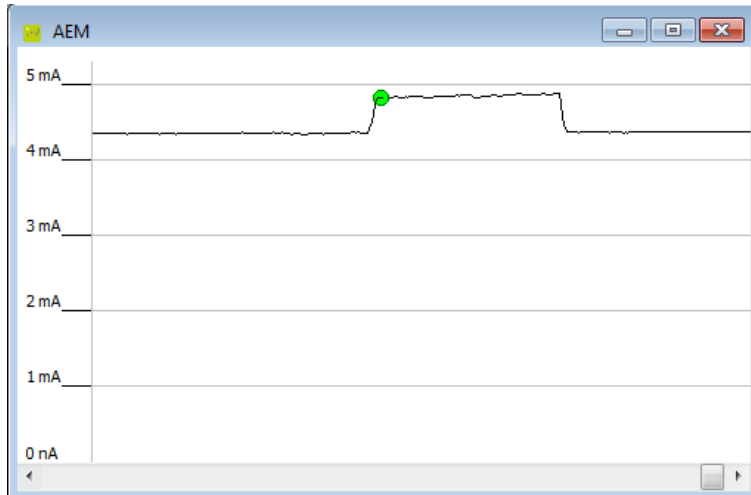


Figure 6 - Kent Display Screen Update Current Usage

These tests were performed for the RAM\_WRITE (writes a byte at a time to the kent's memory) and DISP\_FULLSCRN (performs a full screen update from the memory) commands. The test wrote a 240x320 image array into the memory one byte at a time. For a complete list of commands see the data sheet for the Kent display or the KentDisplay.c file for the MCU.

The Kent display uses a 100m pitch ribbon cable connector. A PCB board was purchased from SparkFun that would connect to the ribbon cable and create a PCB board breakout connector. The connector layout for the Kent display pins can be seen in the figure below. The reset line must be tied to 3.3V. It is tied internally high but it was found that a lot of the time it did not function correctly. It was fixed by wiring the reset wire straight to the logic 3.3V line to avoid constant resets.

Name	Number	Type	Description
RESET	1	Input	Reset (Optional Use).
BUSY	2	Output	Display Busy Status Indicator (Optional Use).
V <sub>SS</sub>	3,8,12,14	Supply	Ground.
SCK	4	Input	Serial Data Clock.
SO	5	Output	Serial Data Output.
SI	6	Input	Serial Data Input.
$\overline{CS}$	7	Input	Chip Select.
V <sub>CC</sub>	9	Supply	Logic Supply (3.3V).
V <sub>PWR</sub>	10,11,13	Supply	Power Supply (4.0V – 9.0V).
NC	15,16	--	No Connection.

Figure 7 - Kent Internal Pin Layout (Kent Displays Incorporated)

Each image sent to the display had to be aligned horizontally (320pixels x 240pixels) with each byte in Big Endian format. The top left byte of the image would map to Byte[0, 0] in the displays memory bank. Refer to Figure 8 for a more detailed look at the memory bank. The boundaries of



the memory show a large pixelated area. This area is called the active layer of the device. The active layer is an additional pixel to the left, top, bottom, or right, of the boundaries (the boundary pixels are actually 2 pixels wide and/or high). So Byte[0, 0] has an active pixel to its left and above it. The active layer cannot be written to. They mimic the pixel's value adjacent to it. So if Byte[0,0] was a white pixel the active layer pixel next to it would become white. There are, however, commands that allow the user to ignore the active layer (avoid the active layer from changing values). The images sent to the display from the microcontroller do not care what values the active layer are or become. It writes whatever image is received from the microcontroller.

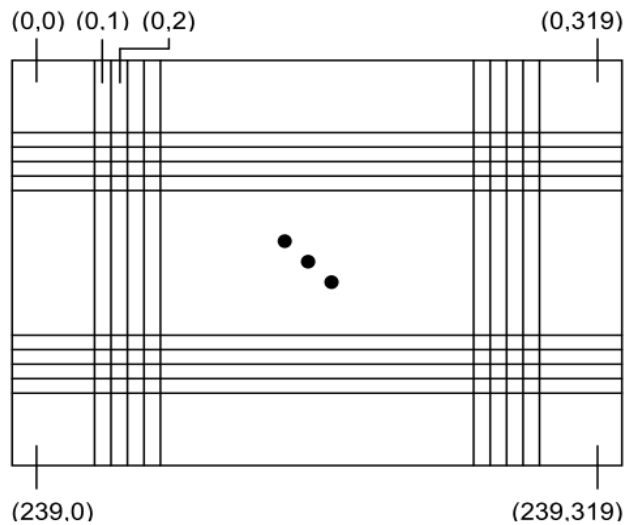


Figure 8 - Active Area of the Kent Display (Kent Displays Incorporated)

The Kent display is controlled via USART (SPI) interface. The SPI settings are slightly non-standard. While it requires CLPOL = 0 (standard), MSBF sent first (standard), it requires CLPHA = 1 (data valid on leading edge). The baud rate was set to 9600 which seemed to work well for the display. The display is very slow and takes 2-3 seconds for a full screen update. The USART route was tied to LOCATION 0 (USART\_USED->ROUTE) in the MCU meaning it uses Port D PINS 2 (TX | SI), 3 (RX | SO), 4 (SCK), 5 (CS) for the USART2 interface. The internal Chip Select line pin in the USART was disabled so that it could be manually set by the drivers file since it needed special chip select capability.

## RF Radio Communication

This section explains the commands that the radio uses, the interface of the radio, and the protocol that the radio uses.

## Commands

**READY:** Value 0xDC. Sent by the MCU to the host computer to state that is waiting for an update. The MCU waits for a response from the computer until a packet is sent from the computer that tells the MCU what to do. The payload is set to 0 since no information is needed by the host.

**UPDATE:** Value 0x50. This command tells the MCU that an image update is being sent. The payload represents an image for the Kent display. This image is written to the onboard memory of the Kent display after the packet has been determined to be valid (valid bits of the size packets are set and the size matches 9600Bytes).

**NULL:** Value 0x00 Used to define an empty command.

## Interface

The XBEE RF Series I radio was used due to its ease of integration. It requires 4 pins. The XBEE interface employs 4 pins: power pin which provides 3.3 volts, ground, DIN (data to transmit), and DOUT (data received). An optional sleep line is being used to put the radio to sleep whenever it is idle. The radio uses around 1mA when idle. While trying to transmit it uses about 7mA and receiving it uses 50mA. In sleep mode drops the radio down to 6uA. The sleep line must be pulled high (3.3 volts) to enter sleep mode. It will only sleep after the TX and RX buffers have been emptied and the SM setting internally must be changed to 1. To force the radio to sleep, the TX line (DIN) had to be pulled low since the GPIO is active high by the USART.

The radio is operated via asynchronous serial communication interface. The radio is operates at 9600 baud rate and only uses the TX and RX pins of the USART routed to Location 1 by setting the USART1->ROUTE register. Location 1 employs pins PD0 as the TX line and PD1 as the RX line.

## Protocol

### Device:

- 1) Wake up radio every 3 hours.
- 2) Send ready signal
- 3) Turn receiver on for 10 seconds and wait.
- 4) If response
  - I) Parse and complete command
  - II) Once Received Sleep Radio
  - III) Wakeup Kent IF UPDATE CMD
  - II) Put Kent to Sleep
- 5) Sleep

### Server:

- 1) Wait for ready signal
- 2) Send acknowledgement once received
- 3) Send image data
- 4) Receive power/energy data
- 5) Send done signal
- 6) Repeat

## 6) Repeat

Step three timeout for the device must NOT timeout sooner than it can take to receive the maximum sized payload. The maximum payload size that is received is 9603kBytes for the Kent display image. The time to receive this size of packet tends to be around 2-3 seconds. The delay was set high enough to allow sufficient time for the data to be received and adequate time for processing the byte before the next byte is read from the radio's buffer. It was determined that a value under 5 seconds could sometimes cause packet errors as the MCU would wake up before the entire packet was received.

## Energy Harvester Board

### Design

The energy harvester was designed around the recently released ultra-efficient energy harvester chip from TI, the BQ25504. This choice was made in order to maximize the energy harvesting from an unstable source, indoor solar, and make decisions based on how much energy was available in the storage element. A big advantage of using this chip is the Voltage Battery OK (VBAT\_OK) logic line put high when the battery has charged to a set voltage, and goes low if the energy available droops below another point. This board also contains two voltage regulators for the peripherals, 5 and 3.3 volts. The 5 volt regulator is capable of boosting and is powered directly off the capacitors, and the 3.3 volt regulator has very low quiescent current at about 1.6 microamps. All the components on the board were chosen to reduce quiescent current as much as possible. To help understand how much energy is available from indoor usage of solar panels, a development board with the TI BQ25504 and test solar panel (capable of 3W @ 6V maximum) were used. The test environment included 1 fluorescent bulb (35W) at 10 feet, and 1 incandescent bulb (60W) at 15 feet, with incidence angles of ~15 degrees. Figure 9 shows energy harvesting rates vs. voltage of the storage element. The y-axis shows how fast the capacitors charged (in micro joules per second), and the x-axis shows the voltage of the capacitor.

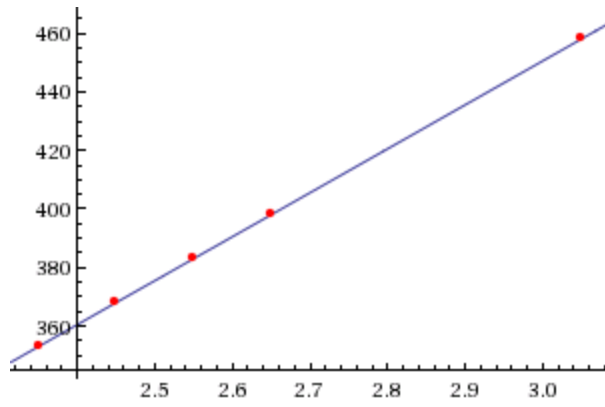


Figure 9 – Capacitor Charging Rate (uJoules/s) versus voltage (V)

## Schematics

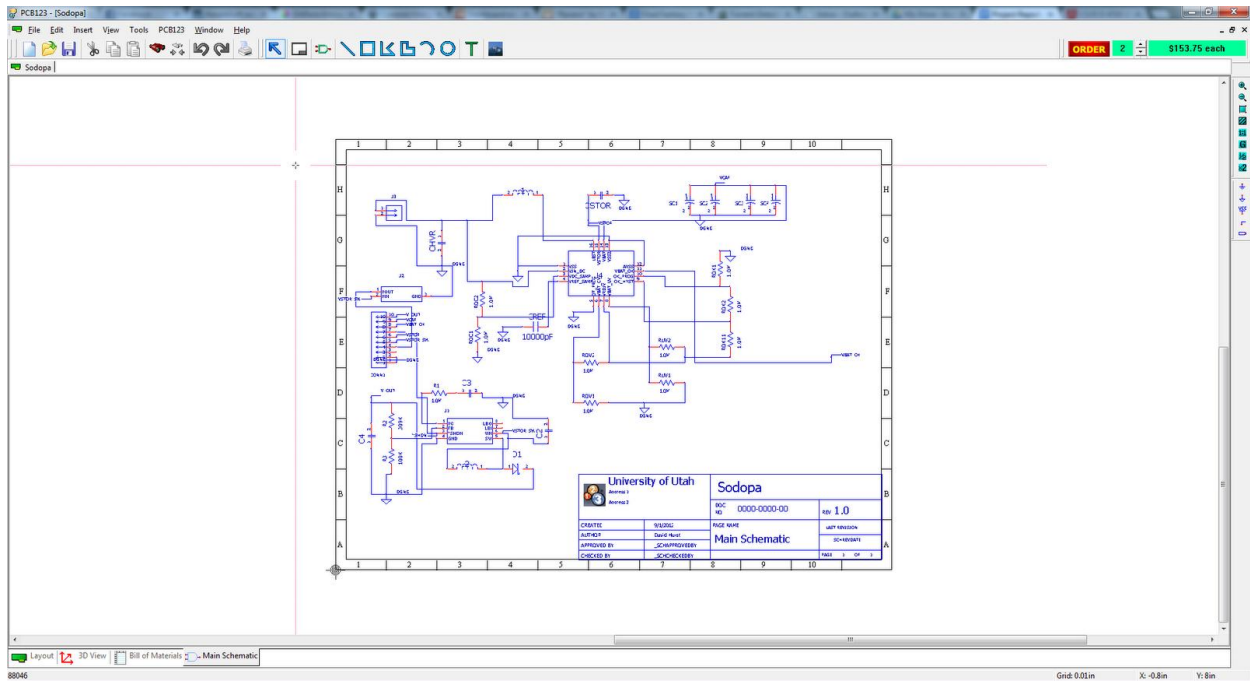


Figure 10 - Energy Harvester Board Schematic

# Layout

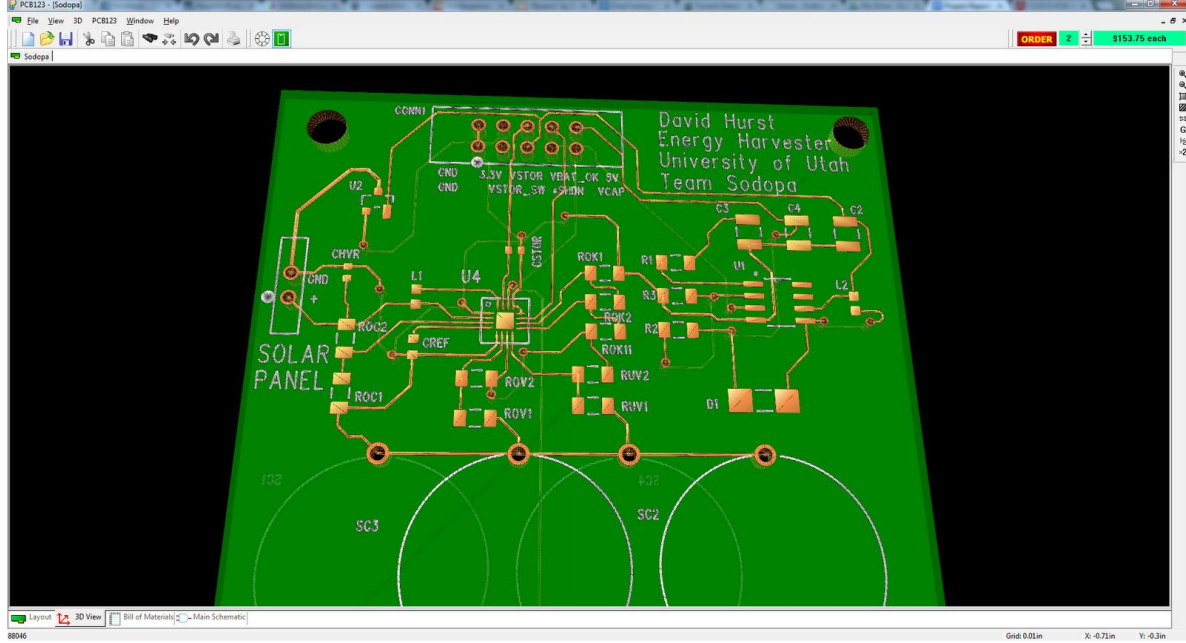


Figure 11 - Energy Harvester Printed 3D Layout

# Parts List

Row	Part	Description	Manufacturer	Package	Ref Des	Price	Quantity	Cost
1	06034D475KAT2A	CAP CERM 4.7UF 10% 4V X5R 0603	AVX Corporation	CC0603L	CSTOR, CHVR		2	
2	LT1308ACSS	IC DC/DC CONV SINGLE CELL 8-SOIC	Linear Technology	SO8-P50-A157	U1		1	
3	12106D476KAT2A	CAP CERM 47UF 10% 6.3V X5R 1210	AVX Corporation	CC1210N	C2, C3, C4		3	
4	10BQ015	DIODE SCHOTTKY 15V 1A SMB	Vishay/Semiconductors	DSO-C2X2.3	D1		1	
5	10073456-001LF	CONN HEADER 10POS DUAL VERT PCB	FCI	CONN10_2X5_US_FCI	CONN1		1	
6	BRC1608T1R0M	INDUCTOR 1.0UH 850MA 20% SMD	Taiyo Yuden	IND0603_39N_TAI	L1, L2		2	
7	66226-002LF	CONN PLUG 2POS 2.54MM VERT TIN	FCI	CON2_1X2_U_FCI	U3		1	
8	BQ25504		TI	SOT758-1	U4		1	
9	CRCW12061.0MJNEA	Resistor Shape	Vishay	RC1206N	ROV2, RUV1, ROK11, ROV1, RUV2, ROK2, ROK1, ROC2, ROC1, R2, R3, R1		12	
10	LM3480IM3-3.3	IC 3.3V 100MA LDO VREG SOT23	National	SOT-23	U2		1	
11	ECJ-1VB1C103K		Panasonic	CC0603N	CREF		1	
Total							26	

Figure 12 - Energy Harvester Part List

## Resistor/Capacitor Values

Table 1 - Energy Harvester Component Values

<b>ROV1</b>	3.6M $\Omega$	<b>CSTOR</b>	47uF
<b>ROV2</b>	6.2M $\Omega$	<b>CHVR</b>	220uF
<b>RUV1</b>	5.6M $\Omega$	<b>C2</b>	4.7uF
<b>RUV2</b>	4.3M $\Omega$	<b>C3</b>	4.7uF
<b>ROK1</b>	3.3M $\Omega$	<b>C4</b>	10nF
<b>ROK2</b>	3.3M $\Omega$	<b>CREF</b>	4.7uF
<b>ROK3</b>	3.3M $\Omega$		
<b>ROC1</b>	15.62M $\Omega$		
<b>ROC2</b>	4.42M $\Omega$		
<b>R1</b>	1M $\Omega$		
<b>R2</b>	309K $\Omega$		
<b>R3</b>	100K $\Omega$		

## Board Assembly



Figure 13 - Energy Harvester Board Assembled

Board assembly consisted of soldering all the parts to the board received from Sunstone Circuits. Most of the parts were either 0603 or 1206 in size, and were hand soldered. The smallest part on the board was the energy harvester chip, the TI BQQ2504, which is a 16 pin QFN. This was soldered using a hot air gun to first solder the ground pad, and then an iron was used to attach each lead to the pad on the board. Testing of the board revealed 2 problems; the 3.3V regulator's input was attached to the wrong trace. To fix this, the trace was cut, and a wire

was soldered on the input pin to the correct location. The other issue was the L2 inductor used on the 5V boost regulator had too small of a pad. To fix this, a through-hole inductor was used, and each lead was soldered directly to the pad.

## Software

The software is responsible for getting what the user wants on the door tag and getting that into something that the tag can understand. To meet the low energy requirements for the display, all image processing is done on a web accessible server before it is sent over the radio to the door tag for display.

For optimal mobility the front end is a website that can be viewed from anywhere that has access to the modified web server back end. This site allows users to upload an image, pull down a webpage, write simple HTML, pull data from Google calendar or other calendar resources in the ICalendar format at intervals, or input their own calendar data for use. The server takes this data and turns it into an image which it then makes suitable for the door display by, first converting it into 1-bit black or white color, then crops it to the correct 4:3 aspect ratio, and finally shrinking or expanding it to fit the 320 by 240 resolution screen. The server then returns the user to a page where they preview what will be displayed on the tag. When the tag requests an update this image is converted into bytes then sent out through a USB port to the radio communication dongle.

## A Use Case

When a user goes to the server's webpage their browser sends an HTTP get request to the web server which the Java HTTP server library catches, parses into an `HttpExchange` object which it defines and allows access to the requests method, path, headers, and body sections separately as well as the headers and body of the HTTP response. The server first checks the method for what type of request it has received; in this case a "get" request which the server responds to by returning the requested file with a 200 response header or an error page with a 404 header if the page could not be found. For the user accessing the server this is the index page written in HTML5.

This web page has several HTML forms to upload different types of data that the user can put onto the display, because it includes all of the other forms' behaviors. The user may choose to upload a Google calendar's iCal link. When the user pushes the submit button on the form for this their browser sends a "post" request to the server, Java's HTTP server catches it like before, but this time the handler searches the body of the request for the unique ID of the form. For the iCal link form it extracts the URL and requests the iCal data from that. A timer is also setup to retrieve the data again every two hours until there is a manual change to what is to be displayed.

The server then parses the iCal data. First it looks for a "BEGIN:VTIMEZONE" containing a "TZID:" tag and adjusts the time zone to match or defaults to the time zone of the server. It then

splits the data into events based on where “BEGIN:VEVENT” occurs. It then takes the start and end times from the “DSTART;” and “DEND;” lines and the summary of the event to display from the “SUMMARY:” line. Each event’s data is contained in an event data structure which allows sorting by start and end times. These events are stored in a sorted list and culled to ensure that the events to be displayed are on today and end after now (the current time and date). The next four events are then put into an HTML table for display.

This HTML table or the HTML data from other forms is then given to the cssbox library which takes in HTML and CSS data and outputs a buffered image of that data for a given viewer size. In all cases the software passes in the displays 320 x 240 resolution, but this library expands both directions a little.

The image from cssbox or uploaded through the image form goes through a few processing steps. First Java’s built-in ColorConverOp.filter is used to take the image and convert it into 1-bit color with the RenderingHint to enable dithering. Then the getSubimage method of the image is called with the size passed in being the closest to the 4:3 aspect ratio of the desired final image that can be obtained while keeping as much of the image as possible. Finally the image is drawn on another image of size 320x240 which has the effect of resizing the image to the resolution of the display.

This image is now ready to be outputted to the display. To show the user what all this processing has done the image is also saved over to the preview image that is on the website and the html for the site is sent back as the response to the post request through the same HttpExchange. This forces the browser to refresh the page and change the image to the processed data the user just put in.

When the server receives the a packet from the radio dongle the rxtx implementation of the Java communications runs a handler which determines which command has been sent. In response to the READY command the image is converted into a byte array by iterating over the pixels of the image and putting their on or off value into the bits of the array in sequence. This byte array is then sent out through the rxtx library’s outputStream prefixed by the UPDATE command and the size of the whole packet.

## **Radio Communication Dongle**

A Teensy microprocessor and X-Bee radio were combined to allow communications via a computers USB port. The Teensy is used as a simple USB to serial converter using the example code provided with the Teensyduino IDE. An adaptor takes the serial signal from the teensy normal sized pin outs, down converts it to 3.3V and outputs it to the correct pins on the X-Bee’s smaller headers. When the X-Bee receives a radio transmission the inverse path is taken by its serial output.



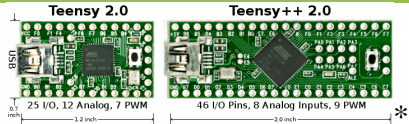

## Bill of Materials

Table 2 - Bills of Materials

Component	Vendor	Contact info	Part Number	Price	Qty
Teensy ++	PJRC	<a href="http://www.pjrc.com">www.pjrc.com</a>	TEENSY	\$16	1
<b>Teensy to XBee Adaptor Kit</b>	PJRC	<a href="http://www.pjrc.com">www.pjrc.com</a>	KIT_XBEE_ADAPTOR	\$6	1
XBee	DigiKey	<a href="http://www.digikey.com">www.digikey.com</a>	XB24-AWI-001-ND	\$19	2
1E-Ink (backup)	Pervasive Displays, Inc.	<a href="#">Contact Link</a>	EW074AS184 / EW074AS183	\$50	1
ChLCD	Kent Displays	TEmanuele@kentdisplays.com	015574130050167	\$109	1
MCU	DigiKey	+47 23 00 98 00 or website	EFM32-G8XX-STK	\$72.88	1
Assorted Caps/Resistors/Connectors	Mouser	~	~	N/A	N/A
Energy Harvester	TI	<a href="#">Contact Link</a>	BQ25504	\$49	1
PCB	Sunstone, Inc.	www.sunstone.com	~	N/A	1
Super Capacitor	Panasonic (via mouser)	<a href="#">Contact Link</a>	S5R5H155	\$4.14	2
Solar Panel	Solmaxx		6V 3Watt Thin Film	\$28	1

## Material Descriptions

Table 3 - Material Descriptions

Component	Picture	Basic Info
Teensy 2.0 (PJRC)		<a href="#">Visit Website</a>
ChLCD (Kent Displays Incorporated)		<a href="#">Visit Website</a>

<p>MCU (Energy Micro)</p>	 <p style="text-align: right;">*</p>	<p><a href="#">Visit Website</a> 128kb Flash, 3 SPI, RTC, power and onboard management unit.</p>
<p>Super Capacitor (Mouser)</p>	 <p style="text-align: right;">*</p>	<p><a href="#">Visit Website</a> 1.5Farad @ 5.5Volts</p>
<p>TI BQ25504 (Texas Instruments)</p>	 <p style="text-align: right;">*</p>	<p><a href="#">Visit Website</a></p>
<p>XBee Radio (Digi)</p>	 <p style="text-align: right;">*</p>	<p><a href="#">Visit Website</a></p>

## Conclusion & Lessons Learned

There were many issues that were encountered. The first is finding an efficient way to deal with the packets from the radio. Having a timeout feature on the radio caused it to often wakeup and stop receiving packets before it had been fully completed. The timeout feature was used to help prevent energy consumption. This value had to be increased higher than anticipated to help receive full payloads (9600Bytes). Also the RX radio interrupts seemed to sometimes stall which made the interrupt finish even though it had more information on the buffer. To fix this a local variable was set to determine when the RX radio had woke up. If it had done timeout yet, it would continue to check the interrupt line (RX line) for data until a full payload is received or the timeout expires.

The web server at one point used separate handlers depending on the type of file that had been requested, but this was removed for a simple handler that could access all files in the server's resources. Dithering and monochrome conversion went from a custom Floyd–Steinberg implementation to the built-in halftone because the built in method worked better for the displayed images. In all these cases the simpler solution took less time to make and worked better than the more complex methods that were tried before them. By simplifying the code the results were cleaner though the complex solutions, which did work, gave lousy results.

The design and assembly of a custom energy harvester board also presented many lessons learned. In addition to the items listed as part of assembling a PCB, the operation of the chip was not entirely clear from reading the datasheet. It wasn't late in the process until it was discovered that the chip could enter into a mode where its boost chargers were turned off, despite having energy available. Placing most of the I/O lines on a header was extremely beneficial in debugging and modify the design as needed.

During demonstration of the project, we encountered issues with our radio communication, which was unexpected. We were unable to show off updating the display wirelessly, but we did manage to show the energy harvesting portion, an update from the device's memory, and the software portion.

## References

Digi (n.d.). XBee® ZB ZigBee® Modules. *Making Wireless M2M Easy*. Retrieved April 24, 2012, from

<http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-zb-module#overview>

Energy Micro (2011, November). EFM32g Reference Manual. *Eink*. Retrieved April 24, 2012, from

<http://www.energymicro.com>

Kent Displays Incorporated (2010, April 5). 1/4 VGA Cholesteric Display Module with SPI - Compatible Interface. *Kent Displays*. Retrieved from

[http://www.kentdisplays.com/services/resources/datasheets/25085c\\_320x240\\_SPI\\_datasheet.pdf](http://www.kentdisplays.com/services/resources/datasheets/25085c_320x240_SPI_datasheet.pdf)

Mouser (n.d.). EEC-S5R5H155. *Mouser Electronics*. Retrieved April 2012, from

<http://www.mouser.com/ProductDetail/Panasonic-Electronic-Components/EEC-S5R5H155/?qs=sGAEpiMZZMsCu9HefNWqphhDW1I6X6cazxL9wSMAdug%3D>

PJRC (n.d.). Teensy USB Development Board. *Electronic Projects Components Available Worldwide*.

Retrieved April 24, 2012, from <http://www.pjrc.com/teensy/index.html>

Texas Instruments (n.d.). Ultra Low Power Boost Convert with Battery Management for Energy

Harvester Applications. *bq25504*. Retrieved from <http://www.ti.com/lit/ds/symlink/bq25504.pdf>