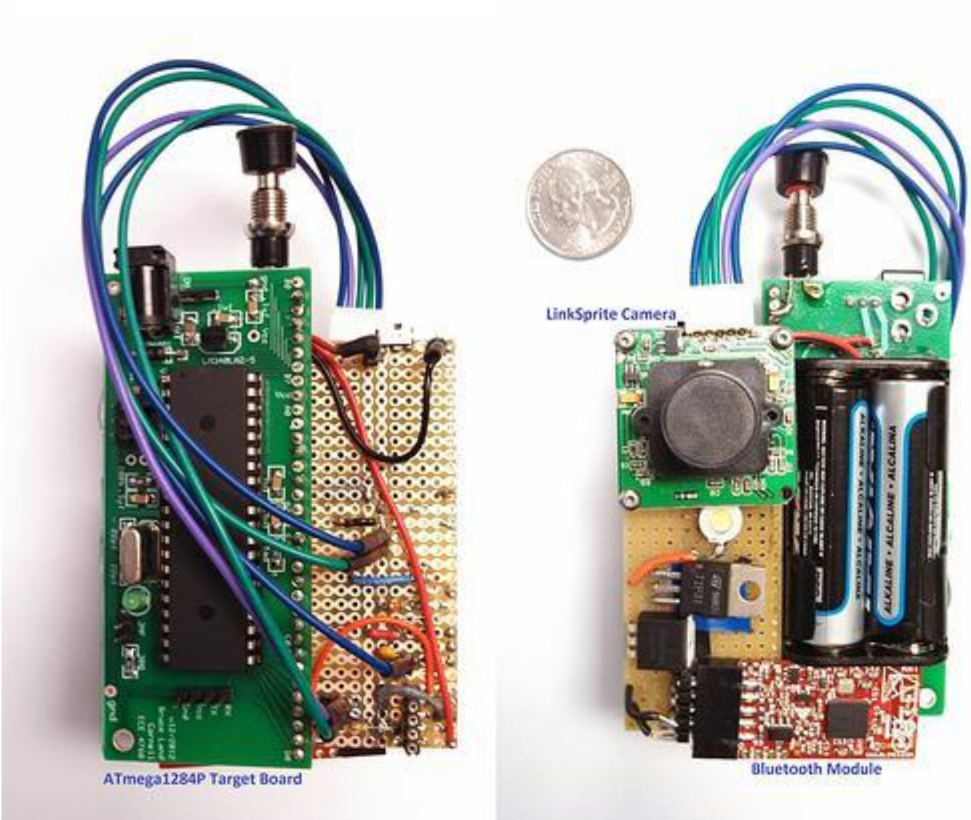
PepperGuard:

An Augmented Pepper-spray with Camera and Emergency Response



John Yin (zy93), Cordelia Lee (crl65), Adrian Tung (lt277)
Final Project for *ECE4760: Design with Microcontrollers*
2012 / 11 / 19

Contents

1 Introduction

In view of the rising crime rate here on campus and the difficulties that many victims faced with identifying crime suspects, we developed *PepGuard*, an augmented pepper-spray. Our device will be paired and connected with a Bluetooth-enabled cell phone at all time. When triggered, it will automatically take a picture of the crime suspect as the deterrent is released, and forwards the picture to the cell phone via Bluetooth. It also directs the cell phone to make an emergency phone call to a pre-configured number, such as the police emergency line at 911.

As a working prototype developed in Cornell ECE4760, our device currently includes an Atmel® ATmega1284P microcontroller from the course, a camera module with a high power LED as flash light, a Bluetooth® transceiver and various supporting components mounted on the solder-board. The total development cost is kept under $100 in compliance with the project requirement.

2 High Level Considerations

2.1 Logical Structure

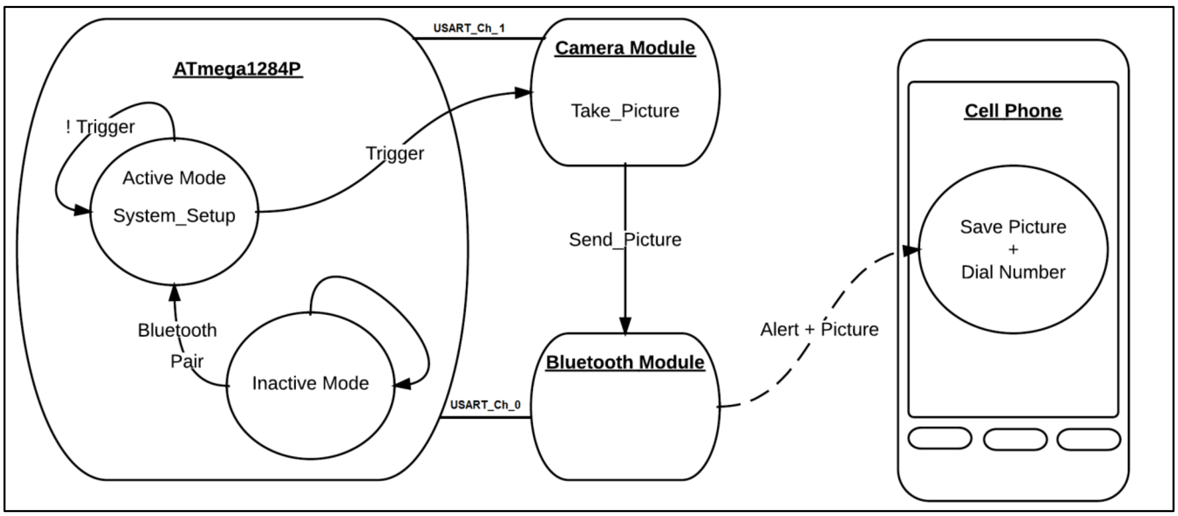


Fig. 1: Conceptual overview of *PepperAlert*.

PepGuard communicates with a Bluetooth-enabled cell phone using client-server architecture. The system initializes in **Inactive Mode** with the Bluetooth transceiver on our device acting as the server. After a cell phone connects with *PepGuard* as client, the system enters **Active Mode** and waits for the button-press trigger. In case of an emergency, the user would press the push-button to release the pepper deterrent, thereby triggering the **Camera Module** to take a photo of the crime suspect (LED flashes at the same time). Immediately thereafter, the microcontroller retrieves the photograph from the camera and routes it to the **Bluetooth Module** to be forwarded to the cell phone. All communications between the ATmega1284P microcontroller and the Bluetooth and camera modules are based on the universal serial asynchronous receiver / transmitter (USART) channels. At the receiving end, the cell phone application receives an alert message, saves the incoming picture data as a JPEG file and initiates an emergency phone call.

2.2 Hardware / Software Tradeoff

There exists commercially available hardware that allows the camera to save photographs directly to a MicroSD card. While this off-the-shelf solution would simplify our software design significantly, it complicates the user interface by requiring more steps to set up the system and thus contradicts our design philosophy. The choice to implement a routing mechanism in this trade-off between hardware and software allows our prototype to be more realistically evaluated.

Push-button de-bouncing can also be accomplished in either hardware or software. Following a similar train of thought, a simple software de-bouncing mechanism is implemented because the former would introduce extra components into the circuit board, which goes against our goal of keeping a small device footprint.

2.3 Standards and Patents Compliance

- **Communication**: The Bluetooth module that we used has been marketed and sold with approval from the FCC.
- **Android Application**: Our prototype currently employs a demo version of the app that we developed and has not been submitted for approval to the Google Play store. It is used as a proof-of-concept user terminal.
- **Patents**: Existing patents cover fixed security fixtures that involves less-than-lethal deterrent (US 2010/0128123 A1), or hand-held security devices that require multiple steps of human intervention (US 2002/0057365 A1). We assume that our prototype developed in class is not in violation of existing patents.

3 Hardware

3.1 Overview

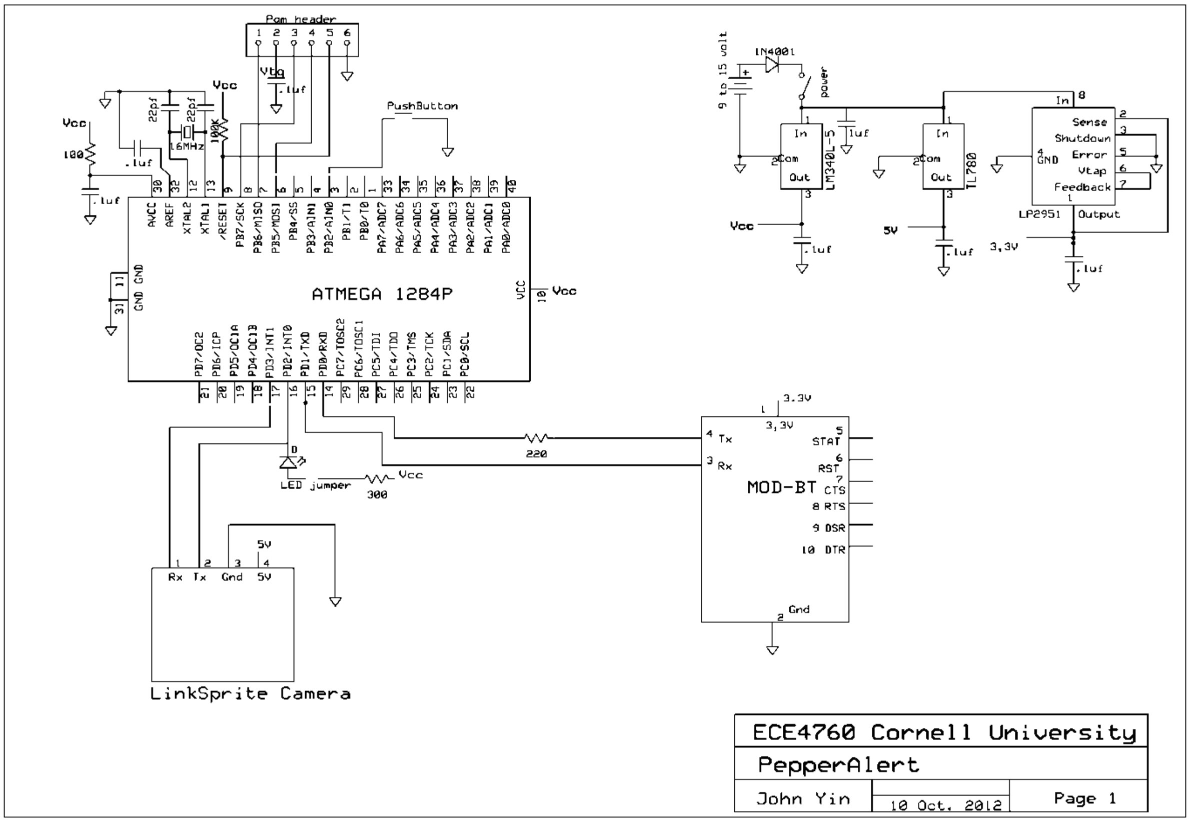


Fig. 2: Schematic capture.

3.2 ATmega1284P Target Board

The ATmega1284P microcontroller (MCU) has been used since the first few laboratory exercises of this course. We populated a target board for the MCU to integrate and communicate with all other peripherals. The target board extends pin-outs of the serial communication ports for debugging. It connects with the peripherals through a single row of SIP connectors.

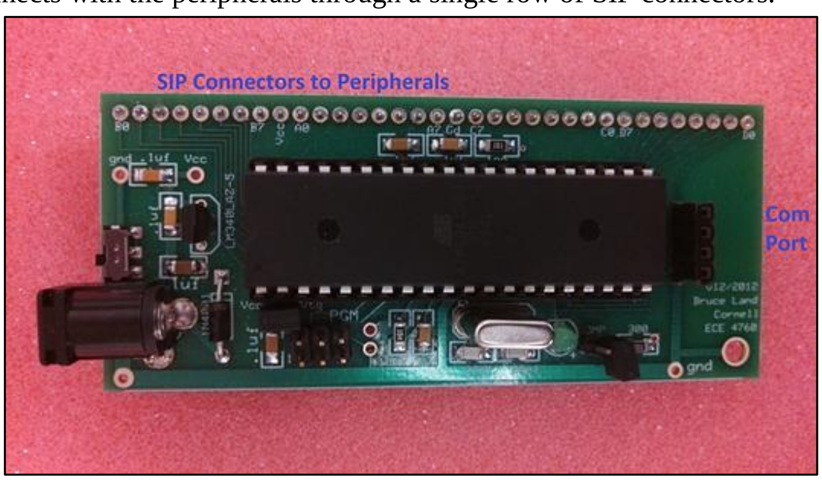


Fig. 3: ATmega1284P target board

3.3 Camera & Bluetooth Peripherals

As shown in **Fig.2**, the LinkSprite® camera module works off of 5V nominal power supply. The rationale for having a separate voltage regulator (TL780) for this module is its excessive current draw (~100mA) that exceeds the maximum rated source current of the on-board regulator (LM340L). The module comes with limited random-access memory (RAM) to temporarily store the previous photograph taken. It communicates with the microcontroller on USART channel 1, configured as 8N1 (8 data bits, no parity, 1 stop bit) at a pre-defined baud rate (default = 38400 bps).

The Bluetooth module is powered by a 3.3V regulator (LP2951) and internally further regulates the power for its various parts. It communicates with the microcontroller on USART channel 0 with 8N1 at 115200 bps.

3.4 External Interrupt

Two of the three dedicated external interrupt pins on the microcontroller coincides with the Rx/Tx of USART channel 1, therefore we used INT2 on Port B to accommodate push-button triggering into the Active Mode. Input to INT2 is not de-bounced in hardware because this will introduce 2 or more extra components into our design, which goes against the small-footprint design principle of our project. As is the norm for servicing interrupts with the ATmega family MCUs, INT2 is driven by the falling edge of an active-low pulse.

Note that it is possible to use almost any general purpose I/O pin to serve as external interrupt. However, such interrupts would trigger an interrupt service routine (ISR) for the entire port and we would then have to poll each pin to find out where it occurred. Although there should really only be one pin (connected to the push button) that can actively trigger an interrupt in our application, all other 7 pins on the same port are exposed as bare connectors in this prototype - thereby significantly increasing the possibility of false alarms. Overall, a dedicated external interrupt pin (INT2) is still a more preferable choice here.

3.5 High Power LED Flash Light

LED flashlight has been widely used in digital photography. It serves as a low cost and robust alternative to the traditional xenon flash when ultra-high intensity illumination is not necessary (see comparison). We chose power white LED, a solid-state lamp, from LUXEON® as the flash light that facilitates our camera operations. The graphs below illustrate the nominal operating conditions of this LED in our device (3V forward voltage across the LED @ ~300mA, producing approximately 0.85 X (typical luminous flux = 105 lumens). See reference for more information on power LEDs.

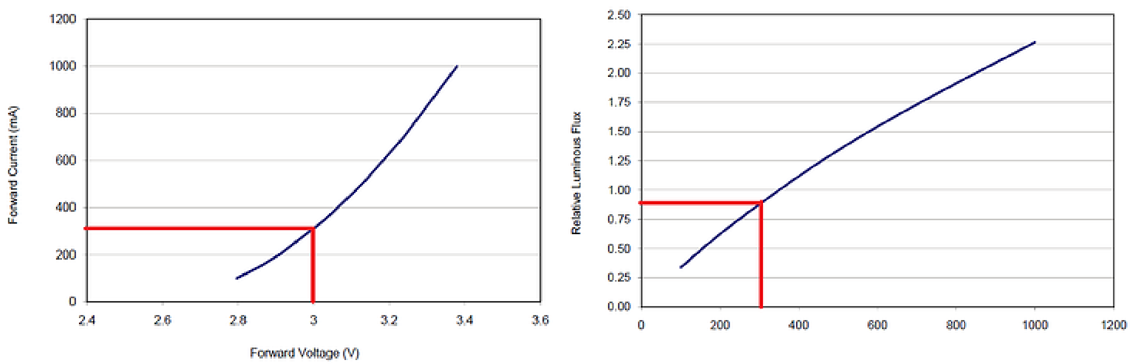
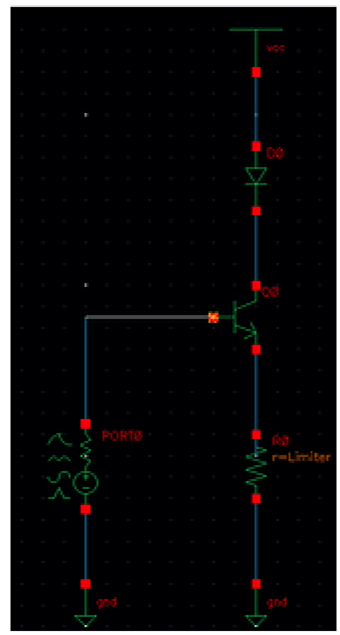


Fig. 4: (Left) Forward Current (mA) vs Forward Voltage (V). (Right) Relative Luminous Flux (X 105 lumens) vs Forward Current (mA)



Recall that the total current budget of an I/O pin on the MCU is 40mA, which is much lower than the 300mA required to drive the LED directly. Therefore, an LED driver or current limiter is necessary. We experimented with an LED driver from TI (LM3405XMK) before realizing that the flash LED in our case does not need to perform any prolonged or complicated blinking pattern. A simple power transistor switching circuit would suffice. In order to trigger the LED with logic level signals (0V / 5V), we used an NPN bipolar junction transistor (BJT, Vth = 0.7V) to serve as a power switch for the power LED.

The pull down current is limited by the resistor on the emitter. Its resistance is calculated by:

$$(V\_B-0.7) / R = 300mA$$

Here, a nominally high resistance (330 ohm) is used in order to avoid thermal runaway. Since the LED has the largest instantaneous current consumption in our system, it is completely decoupled from the MCU to maintain system stability and powered directly from the 9V main supply. In the simulation on the left, we monitored the current through the LED and use an external function generator as the triggering signal. Current draw from the function generator falls within <30mA, which is an appropriate level for the MCU I/O pins to drive.

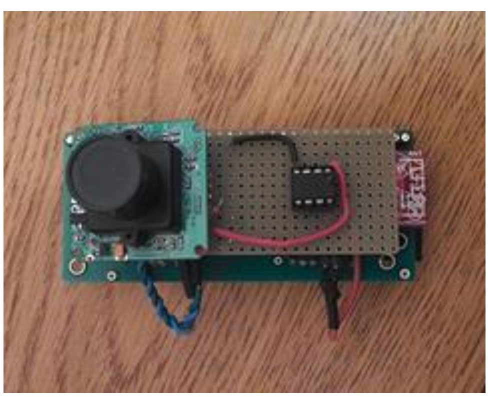
3.6 System Integration and Hardware Debugging

Our first attempt at system-level integration occurred after each sub-system (except the LED) is in working condition by itself (11/12/2012). As much as we would hope to see the entire system miraculously spring into action, it never happened.

The first prototype (shown on the left below) had some serious power-related problems, namely: the Bluetooth was powered from a 3.3V regulator rated at 100mA DC output, and the camera shares Vcc with the MCU. The camera module worked surprisingly well at the time. In retrospect, this is probably because we have not integrated the LED circuitry into our system, and therefore the MCU was not drawing much current from the on-board voltage regulator.

The Bluetooth's behavior, however, was bewildering - it worked at some times but not others. We initially attributed this to a range of hypothetical factors: weak antenna, buggy code, poorly soldered connections. It turns out that at 100mA supply current from the regulator, the Bluetooth is able to successfully connect with the cell phone if the distance and orientation between the two devices were just right - but it would fail most other times. Because the datasheet lacked any information about the electrical characteristics of this module, and our original measurement of 85mA current consumption was much smaller than the typical ~200mA that it actually draws in active mode, it took us a long time to fix this *benign* bug.

After Thanksgiving, the camera module mysteriously failed, most likely due to an unfortunate ESD accident. Now that we had to re-order a new part, it seemed reasonable to completely revamp our design, correcting the mistakes that we made earlier on in the meantime. Thus we have version 2 (on the right).



(Left) Prototype Ver 1.0: BT sandwiched between MCU and camera to minimize device footprint (when batteries were not introduced). (Right) Prototype Ver 2.0

Despite a larger package, this prototype has been working really well for us to test and debug our android application. Some suggestions for further improvement will be discussed in §5.

4 Software

4.1 Camera Module

Commands to the camera module typically involves 2-5 header bytes followed by the payload (if exists). The module returns a response packet if the action is completed successfully. For example, a successful system-reset command appears in **Fig. 5**.

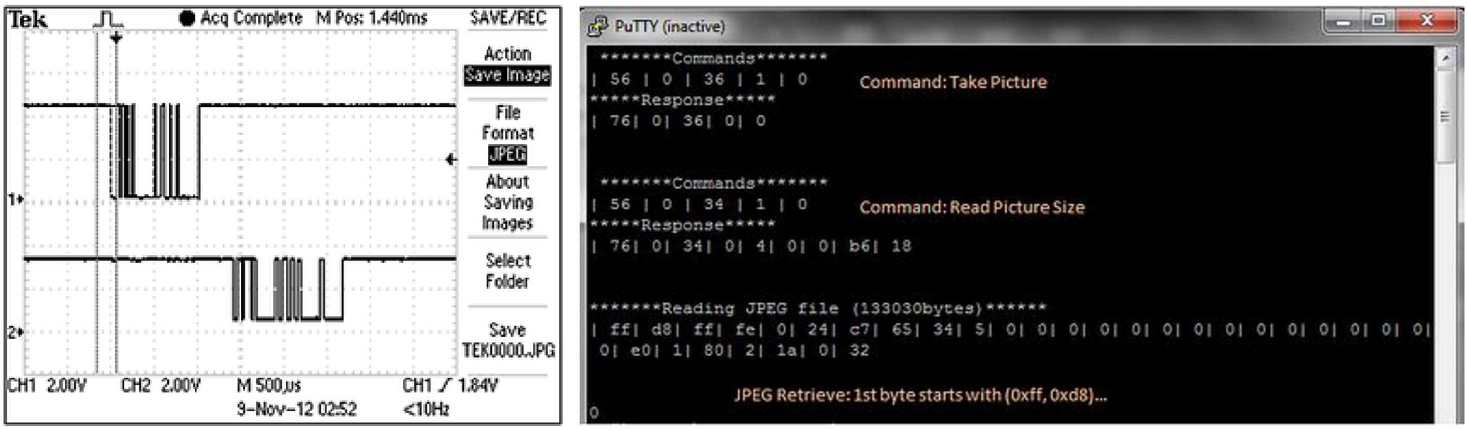


Fig. 5: (Left) Camera reset and response. (Right) Typical camera operation.

To simplify our code and improve on readability, we packaged common command packets into macros, to which the payload is passed in as a function parameter.

After a photograph is taken, it has to be retrieved by accessing memory-mapped addresses of the RAM directly, beginning at location 0x0000. In our implementation, the photograph is first divided into chunks of 32-byte based on its size. Data is then incrementally queried by the microcontroller and routed to the Bluetooth module until the terminator string (0xFFD9) is observed.

4.2 Bluetooth Module

Fig. 6: Typical Bluetooth operation.

The Bluetooth module automatically does a system-reset upon powering on. In Inactive Mode, the microcontroller sets up the module as a server and waits for the cell phone to be connected as a client. During an operation, the module receives data packets routed from the microcontroller and forwards it to the cellphone. **Fig. 6** illustrates this process.

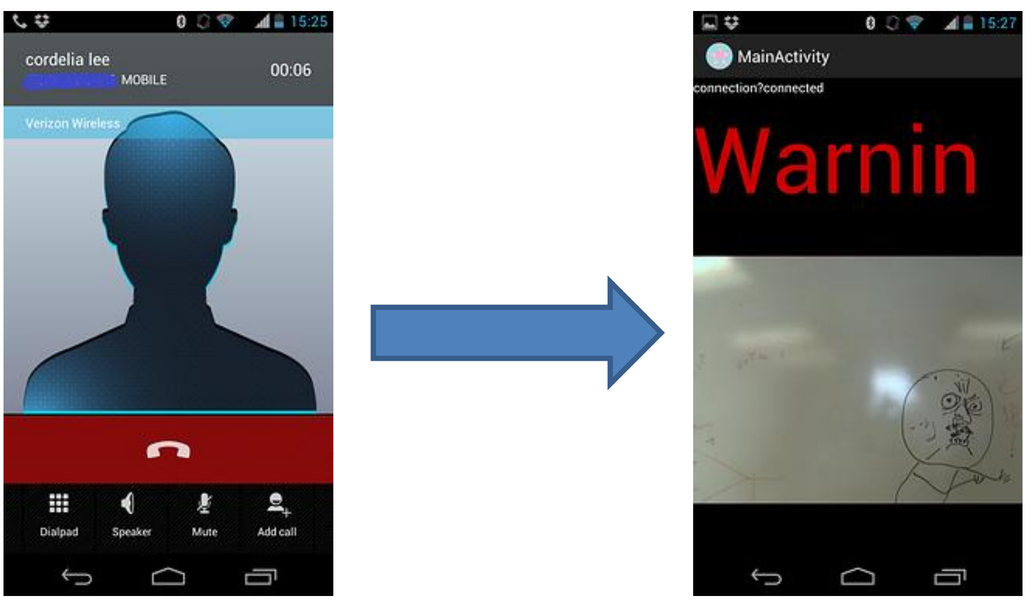
Note that no form of handshaking or error detection algorithm is implemented in our prototype. Communication may become more reliable if these techniques were employed, and they probably should be in a more mature product as the JPEG file format is intolerant to error.

4.3 Pushbutton De-bouncing

As mentioned in §3.4, de-bouncing of the pushbutton trigger is realized in software. A typical approach is to use a state-machine[1] to monitor the changes in an input pin (INT2), and thus confirm a valid button press. The advantage of this approach on a microcontroller is that it allows the processor to cater to other tasks while de-bouncing a switch, which is an efficient way to utilize limited computational resources. However, this would require slightly longer code than a simple busy-wait in an interrupt service routine (ISR). In our device, chip memory, rather than CPU time, is the truly scarce resource. The microcontroller has no other tasks to cater to before a valid button press, but requires every savings in memory-space possible in order to temporarily store the picture data. Therefore, pushbutton de-bouncing is implemented as a busy-wait in an ISR.

4.4 Android Application

The android application receives emergency alerts and a picture of the crime suspect over Bluetooth when the pepper spray is triggered. By using the built-in Bluetooth API in Android SDK, we configured the cell phone's Bluetooth transceiver as a client to PepGuard. The received picture is saved on the cell phone while an emergency call is initiated.



[1] http://people.ece.cornell.edu/land/courses/ece4760/labs/f2012/lab2.html

The cell phone needs to be paired with the Bluetooth module at all time. If PepGuard is switched on, it automatically enters server mode and waits for an active connection with the correct passcode. The android app is programmed to find the PepGuard device from all paired Bluetooth devices that the phone kept track of. At any given time, if the app is activated, it will attempt to establish a connection to PepGuard as a client. After the connection is established, the application waits for a triggered alarm.

In case of an emergency and the push-button is pressed, the application immediately examines the incoming data stream to look for an alert message. Once this is identified, the application blinks a red "Warning" message on the cell phone screen and immediately starts to convert the incoming data stream into hexadecimal numbers and append to a long hexadecimal string. For each of the 32-byte chunk of received data, the application looks for the JPEG terminating strings "0xFF, 0xD9" to terminate its operation. If this is found, the long hexadecimal string will be converted back to byte array and saved as a "photo.jpeg" file in the device storage. Once the image is successfully saved, it is converted into bitmap to be displayed on the home screen of the application. In the meantime, it initiates an emergency call to a pre-configured number to call for help.

Not all cell phones support concurrent programming, and therefore we did not allow the phone call and image transfer to occur simultaneously. Since the JPEG file is a relatively error-intolerant format, we felt that preserving this information is a more critical and time-sensitive task than initiating the phone call. Note that our application does support background operation. The entire chain of events will remain operational even if the cell phone is in sleep mode (with screen powered off). Furthermore, in case of the cell phone running into memory exhaustion, the emergency call will still preempt the home screen and the picture will be saved.

5 Result and Future Directions

5.1 Summary

Each sub-system is tested by monitoring microcontroller-peripheral communications through an emulated terminal (PuTTY) on a PC. Thereafter, the integrated device is run in a quasi-real-life setting in the lab (without the actual pepper spray). The system reliably alerts the cell phone and saves the photograph taken, although picture quality varies with the lighting condition. Some areas for future improvement are:

- Android app should respond to multiple pictures sent
- Further reduce device package size
- More reliable communication between cell phone and *PepGuard* (e.g error detection / correction mechanism for JPEG file transfer)

5.2 Design Outcome vs. Expectation

In our original proposal, we suggested the basic use case and architecture of our system, including some possible hardware / software solutions that may be combined to produce a working prototype. In particular, we proposed that:

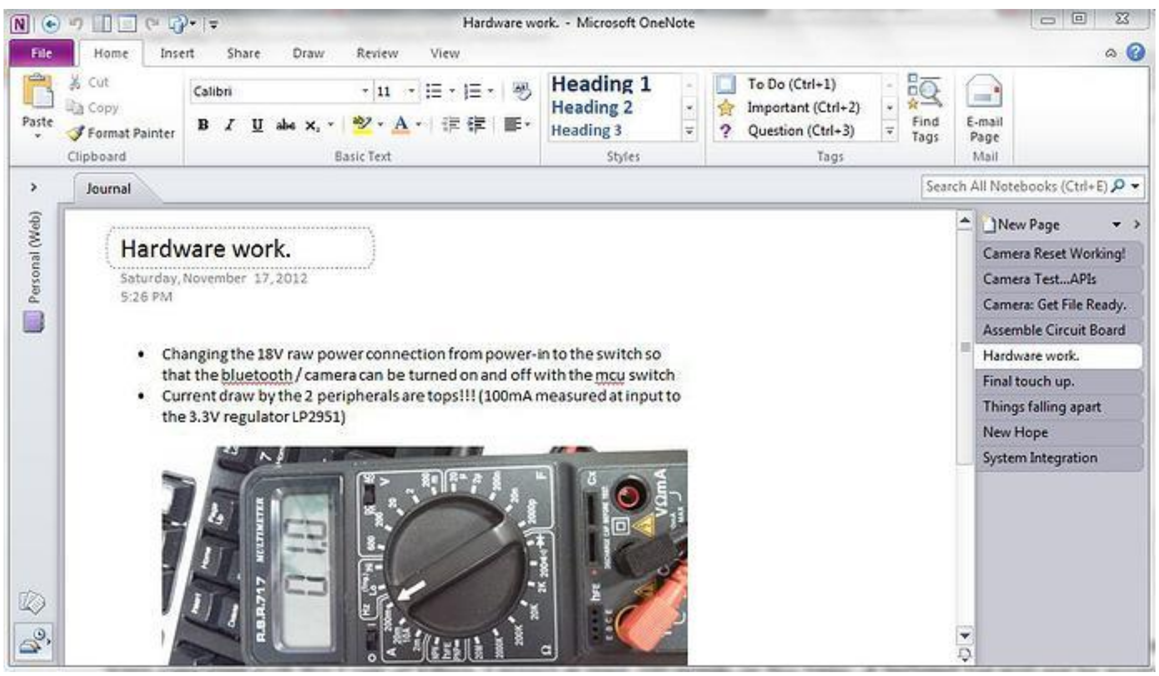
- ✓ Camera and bluetooth modules will communicate with the MCU using USART
- ✓ Camera flash (perhaps a high power LED) is driven by a de-coupling circuit
- ✓ Photographs of the crime suspect should be saved to a microSD card

- ✓ A phone call should be initialized

We are glad that most of our objectives are met in this prototype. The idea of interfacing with the microSD card had to be abandoned halfway through our project because our teammate (Adrian) who should have been working on this part of the project was stuck on the west coast due to Hurricane Sandy. As a remedy, we decided to transfer the photos directly to the cell phone via Bluetooth. One potential drawback of this approach is the extra level of uncertainty that an unreliable wireless communication adds to the overall success of our system. Routing camera data directly to flash (microSD card) through physical wires is almost certainly going to be more reliable.

One thing that we learned from this project is the paramount importance of making progress in interlocked steps. A lot of time could have been saved if we had measured the operating current consumption of each device accurately at the beginning, and performed relevant power calculations.

In addition, we also had ample opportunities to record our weekly progress (see below), and when problems occur, use these records to eliminate improbable causes. Such information has also proven to be really useful in preparing this final report.



Weekly Project Progress Report

6 Acknowledgement

We would like to thank Professor Bruce Land for his prompt guidance throughout, and the Teaching Assistants from ECE 4760 for making this class such an enjoyable experience for all of us.