Home Automation Communication System I

By

Brett McNerney
Tim Evans

Submitted to:

Dr Gary Dempsey
EE451 Senior Laboratory

October 28, 2003
## Table of Contents

Abstract p. ii

Objective p. 1
Specifications p. 1
Division of Labor p. 2
Software Design p. 2
Software Results p. 4
Hardware Status p. 5
Hardware Results p. 6
Problems p. 7
Conclusion p. 7

Appendix A
High level Software Flow Chart a-1
24 hour clock interrupt handler a-2
Bit detection handler code a-3
Hardware Schematic a-4
Abstract

This article contains the objectives, specifications, results, and conclusions of a laboratory project to design a home automation communication system. An EMAC 80515 microcontroller development board was used to transmit a signal over a 24 AWG copper phone wire. This signal was converted to TTL logic levels and sent to another EMAC 80515 microcontroller development board. The microprocessor must translate these signals to commands to turn on various devices. The entire system must run in real-time and was designed for mass-production (i.e. simulate with Monte Carlo analysis in PSPICE). According to demo grades, the software was 60% functional, and hardware was 94% functional.
Objectives:
The purpose of this project was to develop a Home Automation Communication System. The system received a signal from the transmitter which then turned on various devices connected to the receiver. The signal received from the transmitter was converted to TTL levels in order to be processed by the EMAC 80515 microcontroller development board.

Specifications:
1) The design needs to function properly at a line length of at least 1000 ft.
2) All code must be developed for real-time
3) All hardware needs to be designed for mass production.
4) Power Supply: ±5VDC (± 5%) for hardware interface, provided to students.
5) Product Temperature: 0 to 120 degrees F.
6) Phone line wiring: 24 AWG copper, **maximum length to be determined by students.**
7) Port 4.0: flip pin to show fastest interrupt timing.
8) Port 4.1: Turn on LED for 2 seconds when time bit is received.
9) Port 4.2: Toggle LED on/off when frame bit received.
10) Other Port 4 pins: show other timing generated by interrupts.
11) Keypad: After initialization, key=command for LCD display or override control of the output devices.
12) Output Devices: Turn devices on when corresponding received bit D7-D0 is logic 1:

D7: turn on outdoor lights (a 5V relay will be provided to switch on/off 120VAC light).
D6: turn on sprinkler system water pump (a 5V relay will be provided to switch on/off 120VAC pump).
D5: activate security lock (a 5V relay will be provided to switch on/off 120Vsolenoid).
D4: turn on indoor lights (a 5V relay will be provided to switch on/off 120VAC light).
D3: turn on security alarm (a Sonalert device will be provided).
D2: turn on stereo (a 5V relay will be provided).
D1: turn on coffee maker (a 5V relay will be provided).
D0: turn on TV (a 5V relay will be provided).

10) 2-Line LCD Format (See Table 1):

<table>
<thead>
<tr>
<th>CMD or Key</th>
<th>Line 1</th>
<th>Line 2</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>after startup</td>
<td>ENTER A-D FOR CMD</td>
<td>OR 0-7 FOR OVERRIDE</td>
<td>main menu</td>
</tr>
<tr>
<td>CMD B</td>
<td>D7 D6 D5 D4 D3 D2 D1 D0</td>
<td>actual received bits</td>
<td></td>
</tr>
<tr>
<td>CMD C</td>
<td>800 AM</td>
<td></td>
<td>Show current time in hr/min</td>
</tr>
<tr>
<td>CMD A</td>
<td>SW reset</td>
<td></td>
<td>Hold on display for 1sec</td>
</tr>
<tr>
<td>CMD D</td>
<td>ENTER A-D FOR CMD</td>
<td>OR 0-7 FOR OVERRIDE</td>
<td>Back to main menu</td>
</tr>
<tr>
<td>CMD 0-7</td>
<td>OUTPUT OVERRIDE X</td>
<td>OR 0-7 FOR OVERRIDE</td>
<td>Toggle output device on/off, X=device (0-7)</td>
</tr>
</tbody>
</table>

**Table 1:**

LCD Display Content Based on Command Entered
Division of Labor:

Tim Evans – Hardware Design
- Convert the signal to TTL logic
- Buffer the signal for the microcontroller
- Design for Mass production (PSPICE MONTE CARLO simulations)

Brett McNerney – Software Design
- Signal Detection
- Keyboard Support
- LCD Display
- 24 hour clock
- Real-time design
- LED circuitry

Software Design:
The software portion of the lab was designed to run as a real time system. The final flow chart of the software design can be seen in Appendix a-1. A modular design approach was used when writing the code. This not only makes troubleshooting easier, but also decreases compiling time.

The software was designed to use two timers to handle the interrupts in the system. This included a keypad interrupt, timer0 interrupt, and a timer1 interrupt. Timer0 was used to keep track of the time for the twenty four hour clock. A flow chart for the clock can be seen in Appendix a-2. Timer1 was used to sample the incoming data.

Next the LCD display code was created based on the test code supplied by Dr. Dempsey. The code displayed information on the screen based on which key was pressed. The keypad handler received the data from the keypad, and then converted it to hexadecimal. The flow of the keypad handler can be seen in Figure 1.

![Figure 1](Image)

**Figure 1**
Flow of Keypad Handler Code

The majority of the time was spent on the software detection code. This code detected the presence of the frame bit, time bit, and data bits. The software detection was designed to use timer1 to measure the level of the incoming data. Figure 2 shows the flow of the software handler for the frame bit.
Once the frame bit was detected, the system would then begin to measure the incoming data to distinguish which bits where active. See Appendix a-3 for the flow of the handler of incoming bits.

The last interrupt handler is the time bit detection. The purpose of this is to synchronize the 24 hour clock at exactly 8:00 AM every morning and limits error. See Figure 3 for the flow of time bit interrupt.

The last module to be completed was the manual bit override. This allowed for individual bits to be toggled on or off by the pressing of the corresponding number on the keypad. Since the output does not depend on the input signal in this state, the system could be tested without a functioning input system.
In addition to the manual override, LED’s were attached to port 1.0 through 1.7 to allow the user to see which bits were active and which were not. The LED’s were sinked to the EMAC 80515 microcontroller development board and the circuit is shown in Figure 4.

![LED hook up schematic in sinking mode](image)

**Figure 4**  
LED hook up schematic in sinking mode

**Software Equations:**  
The only equation used in the software design was used to calculate the timer reload values. The equation is as follows:

\[
\text{Reload Value} = \frac{\text{Desired Time}}{(12 \text{ MHz} / 11.0592 \text{ MHz})}
\]

Once the reload value is obtained it is converted into hexadecimal and stored in the reload register.

**Software Results:**  
In the end, all of the code was written into a total of twelve modules. However, not all modules were fully tested. The LCD, manual override, and keypad were all completed and tested. The bit detection code was written but was never tested due to time constraints. The time table can be seen in Table 2.

<table>
<thead>
<tr>
<th>Design</th>
<th>Coding</th>
<th>Debugging</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 hours</td>
<td>20 hours</td>
<td>10 hours</td>
<td>7 hours</td>
<td>13 hours</td>
</tr>
</tbody>
</table>

**Table 2**  
Amount of hours spent on software by development category
**Hardware Design:**
A Schematic Diagram of the final circuit is included in Appendix a-4.

![Hardware Block Diagram](image)

**Figure 5: Hardware Block Diagram**

Figure 5 shows a block diagram of the hardware design. The first step in the design process was to transform the sine wave input signal into a square wave. This was done with a full wave rectifier with a filter. However, the waveform’s amplitude was less than 2.7V, which is the minimum TTL high. To fix this, a comparator circuit was added.

The next step was to design for the voice input signal. Examining the FFT shown in figure 6, it was obvious that a majority of the noise was coming from low frequencies (less than 5 kHz).

In order to compensate for low frequency noise in the input signal, a high pass filter (HPF) was added. This particular design uses a first order filter. A higher order filter would have increased the protection against noise, but would be more expensive. A band pass filter (BPF) would have protected against high frequency noise as well, but would be even more expensive. The following two calculations determine the break frequency and gain of the HPF.

\[ 2 \pi f_0 = \frac{1}{RC} \] - Break frequency calculation
\[ G = \frac{R_2}{R_1} \] - Gain Calculation
Figure 6: FFT of the Voice Signal

Note that there is considerable noise in the low Frequency Range

Hardware Results:
The core design of the hardware system was completed and tested. However, the additional hardware to be used to test the software was not added (except the LED’s), since the software was not implemented with the hardware.

The goal of this particular design was low cost (see the bill of material in Table 1). However, this makes the system slightly less functional. These problems are seen when the line length is increased to approximately 3000 ft. (see PSPICE outputs in Tim’s notebook, pages 96-103). The system ran correctly at 1000 ft in lab, and up to 4000 ft in PSPICE. PSPICE modeled the noise as a 500mV, 4kHz signal.

The full wave rectifier circuit needed to use ‘412 op-amps, since ‘741’s could not handle the high frequency due to slew-rate limitations – for evidence of the problem, see Tim’s notebook, pages 57-61. The time table can be seen in Table 3.

<table>
<thead>
<tr>
<th>Early Reports</th>
<th>Design</th>
<th>Debugging</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 hours</td>
<td>15 hours</td>
<td>10 hours</td>
<td>15 hours</td>
<td>5 hours</td>
</tr>
</tbody>
</table>

Table 3
Amount of hours spent on hardware by development category
Table 4  
Bill of Material

Table 4 shows a bill of material for the project. It should be noted that the design also requires a 5V power supply, as well as an EMAC 80515 microcontroller development board. Prices were based on a run of 10,000, and were obtained from the following two websites:
http://www.elexp.com/sem_n914.htm
http://www.digikey.com

Problems:
- Insufficient time to test the bit-detection code.
- High line length was not accurately converted to TTL due to noise.

Conclusion:
The hardware was almost completely finished, and the system earned a 94% demonstration grade. By increasing the cost and complexity of the design, a higher degree of functionality could have been reached. First, adding hysteresis to the comparator may or may not be helpful. It should help against noise, however it would theoretically add to the delays, making it hard for the software end.

Second, the HPF used was a first order filter. This did not filter out the noise as much as higher order filters, but it was much cheaper. However, if a higher line length is needed, or if the signal is going to have considerable noise, then a fourth order Sallen Key implementation would be recommended. This way, a 747 (dual 741 op-amp) chip could be used. This would allow for much more filtering without a great deal of added cost. If that still is not enough, the degree of the HPF can be increased fairly easily.

The software was mostly completed, and the design earned a 60% demonstration grade. By statistical observation of the class’s progress, it appeared that the software was considerably more difficult than the hardware. This is possibly because a good portion of the software was new material, or concepts that the class was weak in. On the other hand, the hardware was all mostly familiar material.
Appendix a-1
High level Software Flow Chart
(Dashed lines show interrupt connections)
Appendix a-2
Flow Chart for 24 hour clock interrupt handler code

- **Over Flow Counter**
  - Increments minutes every time overflow happens
  - Loops every one second

- **Seconds**
  - Loops 94 times
  - Increments minutes every time seconds loops 60 times
  - Loops every one minute

- **Minutes**
  - Loop 60 times
  - Increments Hours every time minutes loops 60 times
  - Loops every hour
  - Sets AM/PM flag when reaches 12:00

- **Hours**
  - Loop 12 times

- **All counts reset when they are decremented to 0 and are reset to their original count value**
- **If AM/PM flag is set then PM is displayed**
Appendix a-3
Flow Chart of bit detection handler code