Adaptive Elevator Dispatcher (Using an 8-bit AVR Microcontroller)

by

Greg Cloutier
gcloutier@hartford.edu
Dep. of Electrical & Computer Engineering
University of Hartford

Saeid Moslehpour
moslehpou@hartford.edu
Dep. of Electrical & Computer Engineering
University of Hartford

Abstract: *An elevator usually operates in a first come, first served mode. Any individual passenger only selects to go up or down from their current location. It is not until the passenger has entered the elevator that their actual desired location is known. The elevator will run up and stop at floors in ascending order where either a desired location is known or a new pickup request has been made. The elevator then repeats this process in the down direction and continues to cycle in this fashion until it has reacted to all the requests. Also note that the elevator has no idea what the passenger count is per request. A group of 10 passengers only needs to request a destination one time.*

This project will focus on a simulated office building that has 10 floors and four elevators. There are at least three different time periods where it would be nice to optimize the dispatch of the elevators with the intent to increase passenger throughput. These times are in the morning when workers first arrive, lunch time when some workers go to a cafeteria on a particular floor (or leave the building), and in the afternoon when workers leave the building. Initially, this project will focus on the Monday morning rush with the intent to increase passenger throughput over a fixed period of time.

I. Introduction

The easiest way to dispatch the elevators would be to have all passengers enter their desired location into the system as they get in line. Since this input is not possible, this project would implement the next best thing. It will monitor elevator usage over time and use historical patterns to predict how the elevators need to be dispatched for higher passenger throughput. For example, the Monday morning passengers will probably arrive at the same time and go to the same destination every week. The adaptive elevator dispatcher will learn this pattern. If the passenger mix does happen to change, the dispatcher will adapt to this change over time as well. This project will also try to determine the value of knowing how many people get on and off at certain floors. The

assumption would be that a vision package (or some other sensor) could count the passengers after any load/unload of passengers.
This project is intended for use as a proof of concept, with no actual functioning elevator to try it on. It is assumed that elevator functionality would already be taken care of, so this project will focus on the predictive aspect of the control system. The goal is for an elevator to arrive before passengers press the button. This project will handle the learning of passenger arrival and adapting to trends over time. Without an actual elevator, there are instead a few assumptions and built-in simulations related to timing and sequencing based on what an elevator actually does.

This paper describes:

- Development Hardware
- Development Software Environments
- Algorithms
- Conclusion

II. Development Hardware

An 8-bit microcontroller should be powerful enough to perform the predictive algorithms and simulations. An additional lower level goal of the project is to learn how to make an Atmel AVR [1] function. An Atmel AVR Butterfly [2] evaluation board was chosen as it contains all the functionality needed to implement the elevator algorithm (and more). The Butterfly evaluation board contains the following hardware which allows the project to function as required:

- A 32.678 kHz watch crystal to run the real-time clock functions
- An in-circuit programming header
- A JTAG in-circuit debugging header
- An RS-232 level converter for communications with a PC
- 4 megabits of dataflash for storing the elevator usage history. The dataflash communicates with the microcontroller via the spi bus.
- An Atmel ATmega169 [3] 8-bit AVR RISC microcontroller. This microcontroller has many peripherals on board which this project uses. It has 16-Kbyte flash memory, 1-Kbyte SRAM, JTAG for on-chip-debug, UART, SPI, hardware timers, and an on-board 8-mHz oscillator. The micro also has many more peripherals which this project does not use, such as PWMs, ADCs, and a 100 segment LCD controller. If we were to lay out a board for this project, an alternate AVR would have been chosen.

Figure 1: Atmel AVR Butterfly demonstration board

For programming the microcontroller and for performing Real-Time In-Circuit Emulation (debugging), an Atmel AVR JTAG ICE [4] was used, which allowed for monitoring the functionality inside the microcontroller in relation to the code developed for the project. Other hardware used was a laptop PC as part of the simulation, 2 USB to RS-232 converters (one for PC communication, one for the JTAG debugger), and a Butterfly Carrier Board [5] available from Smiley Micros [5], which breaks out the pins in an easy-to-access manner.

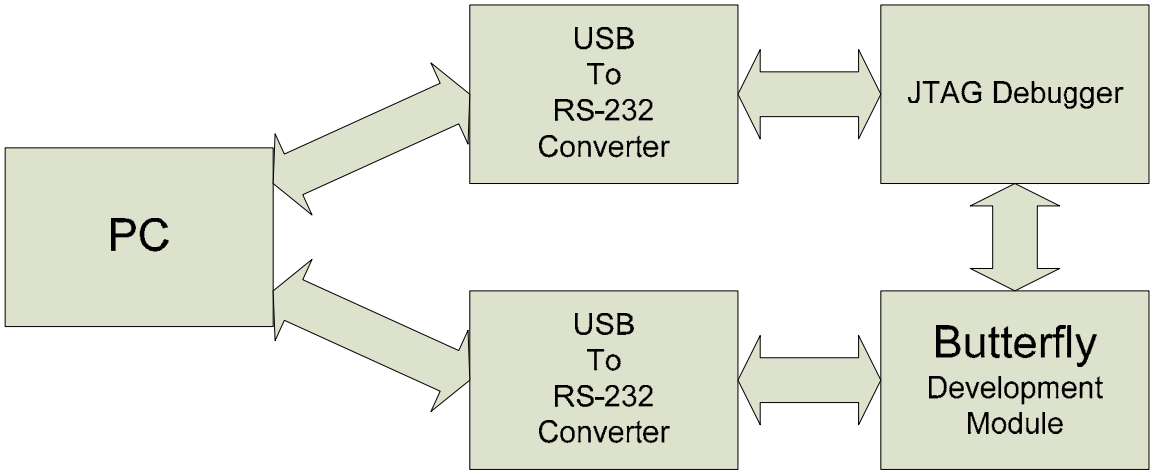

Figure 2: Hardware Block Diagram

III. Development Software Environments

For the microcontroller code development, the program was created in standard "C" language. Although there are many commercial C compilers on the market, the GNU GCC compiler was chosen. This compiler has been imported to the Atmel AVR and is available with an installer for the Windows platform. The package is appropriately named WINAVR [6], is available as a free download, and supports most Atmel AVRs.

For programming and debugging purposes, Atmel's own AVR Studio 4[7] was chosen. If only assembly language programming was used, this is all the software that would be required. In the latest edition, Studio 4 can also act as an Integrated Development Environment (IDE) for the WINAVR compiler. It handles the project organization, provides syntax highlighting, and builds the project from within one interface. Beyond

that, it is Studio 4 which interacts with the JTAG debugger. From Studio 4 you can program the processor, run the code, stop the code, reset the processor, monitor variables, monitor system registers, interactively step through your C code, and set breakpoints. If needed, the contents of variables can be changed on the fly.

If the microcontroller was to act like an elevator and respond as an elevator should, there needed to be a way to simulate the usage of an elevator. For example, the simulation needs to pretend to be somebody pressing a button and waiting for the elevator car to arrive. Remember that the goal is for the elevator car to be available and where it needs to be before the button is pressed; this way the door will just open for the passenger rather than the passenger having to wait for the car to arrive first. For the simulation program, C# (C-Sharp) was chosen to create a user interface and provide communications between the PC and the Butterfly module. In the first generation of C#, it was very difficult to gain access to the serial port. The Microsoft Visual C# 2005 Express Edition [8], which is a free download, has the ability to use the serial port with new functionality. Essentially, the user interface will send a command to the Butterfly module and wait for the response to come back.

IV. Functionality

A Real-time Clock (RTC) is used to pace the elevator system. Pacing is accomplished by attaching a 32.678 kHz crystal to an 8-bit timer. With a pre-scaler set so that the timer increments every 128 cycles, microcontroller will roll over once every second and set an interrupt. In the interrupt routine, the seconds, minutes, hours, days, and weeks are incremented. A flag is also set so that the main loop knows that one second has passed.

A Universal Asynchronous Receiver Transmitter (UART) is used for communicating with the PC. The UART is used to receive commands from the PC and send status messages to the PC. To send the status message, a transmit buffer is filled and an interrupt driven routine sends the message out from the UART without interfering with any other functionality. It essentially happens in parallel with the rest of the program. To receive commands from the PC, an interrupt routine is called for each incoming character. The interrupt routine places each character into a buffer, echos the character (so you can see it in a program like HyperTerminal), and checks to see if the character was a carriage return. If the character was a carriage return, a flag is set that a complete command has been accepted. The main loop will look for this flag and act accordingly.

Elevators are set up dynamically and can contain any number of floors and any number of elevators. The entire functionality of the code will adapt to the number of floors and elevators, so these parameters only need to be set in one place. For the sake of this simulation, elevators will only act on the passage of time without looking for the most efficient route between floors (if there is one). When an elevator is loaded, a countdown timer is started and the elevator will not be available until the timer has expired. The total time is based on where the elevator currently is, where it is going, how many people

load and unload at each floor, etc. The timing used for the functional test was 5 seconds per floor, 2 seconds per door open/shut, and 2 seconds per person load/unload.

For interaction with the PC, a set of commands was created. The PC issues a command and the microcontroller reacts to the commands. The commands include a way to clear the dataflash which stores all of the history information, a way to reset the current time, a way to request an elevator (to simulate a button push), a way to monitor whether the elevator has arrived, and a way to load the passengers. It is when the button is first pushed that the data is stored for historical reference; remember that the elevator is to arrive before the button is pushed.

The predictive nature of this project relies on the ability to store historical data. The resolution of the history is 5 minutes. Every 5 minutes is considered a time block. The data in each time block contains four weeks of history and a running four-week total for each elevator. Data is stored for each floor that the elevator services. It does not matter which elevator had serviced the floors; all elevators are summed together for the history. When a new week passes, the oldest data is dumped; this process is how it adapts over time. Data is only added when an elevator button is pressed. As history is being collected, the routine is also looking at the next time block. If an elevator is available, it will be sent to what historically was the highest priority floor. If more than one elevator is available, they will all be sent to floors based on historical priority. They will sit and wait within one time block. Once a threshold is passed and a new time block entered, the historical data will be saved, a new predictive block is retrieved, and the elevators act accordingly. There is more than enough dataflash available to record an entire month's worth of 5-minute time blocks.

V. Flow Charts

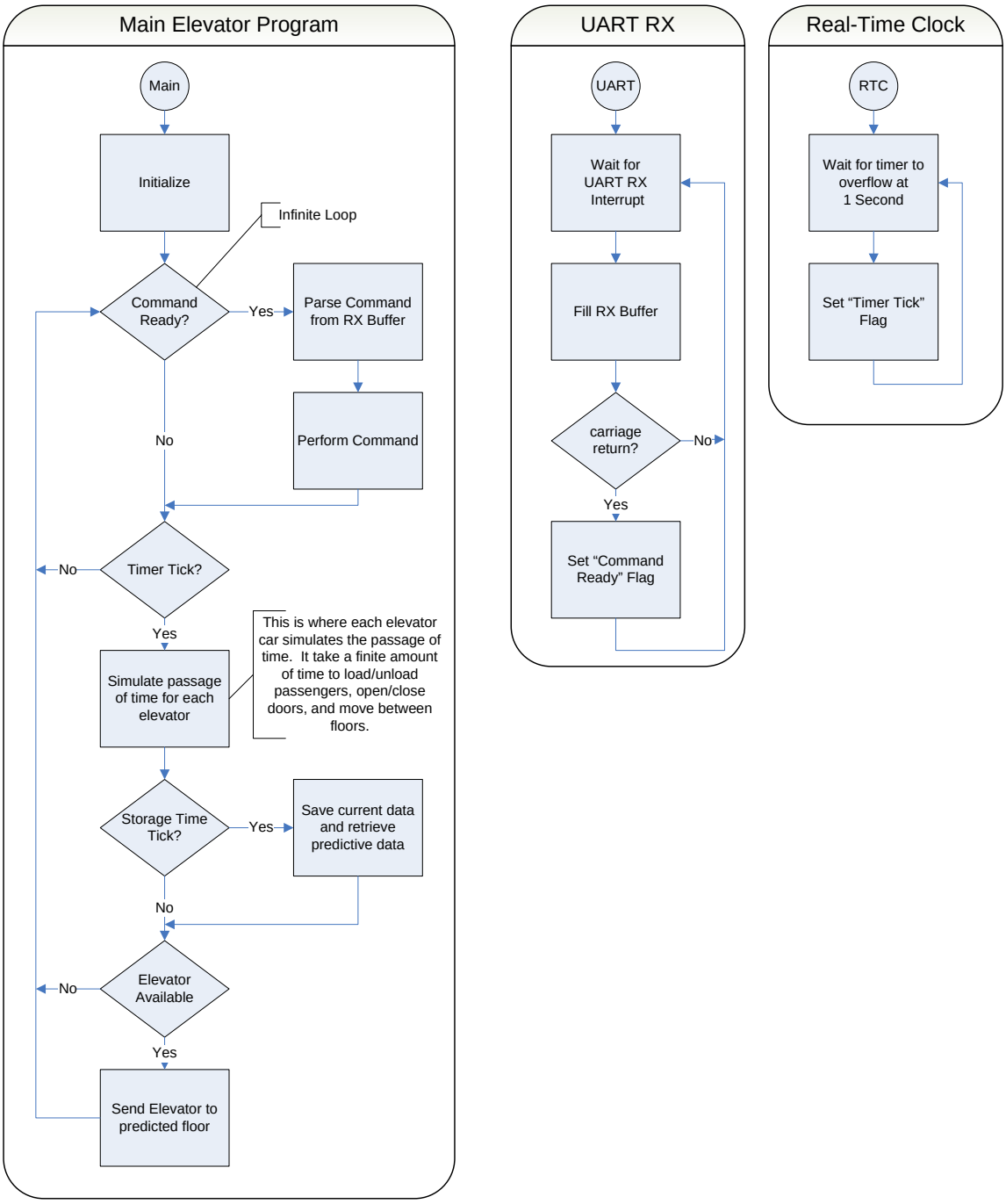

Figure 3: Flow Charts

VI. Sample Code

The following section is a sample of code used to increment the Real-time Counter from within an interrupt service routine:

```
ISR(TIMER2_OVF_vect)

{
    rtcSecond++;               // increment second
      rtcCumulative++;              //increment cumulative time

    if (rtcSecond > 59)
    {
        rtcSecond = 0;
        rtcMinute++;

        if (rtcMinute > 59)
        {
            rtcMinute = 0;
            rtcHour++;

            if (rtcHour > 23)
            {
                rtcHour = 0;
                rtcDay++;

                        if(rtcDay > 6)
                        {
                              rtcDay = 0;
                              rtcCumulative = 0; //time stamp is only
needed for a week
                        }
                    }
            }
      }
      rtcSecTick = TRUE;
}
```

The following section is a sample of code used to open and prepare the serial port used on the PC to provide interaction with the project module:

```
        private void openCom_Click(object sender, EventArgs e)
        {
            if (listCom.SelectedIndex == -1)
            {
                MessageBox.Show("You need to select a COM port from the
            list");
            }

            else
            {
                try
                {
```

```
                openCom.Enabled = false;
                listCom.Enabled = false;
                serialPort1.PortName = listCom.Text;
                serialPort1.Open();
                serialPort1.DiscardInBuffer();
                serialPort1.DiscardOutBuffer();
                string response = commandResponse("B 0");
                textBox1.Text += response;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                openCom.Enabled = true;
                listCom.Enabled = true;
            }
        }
    }
```

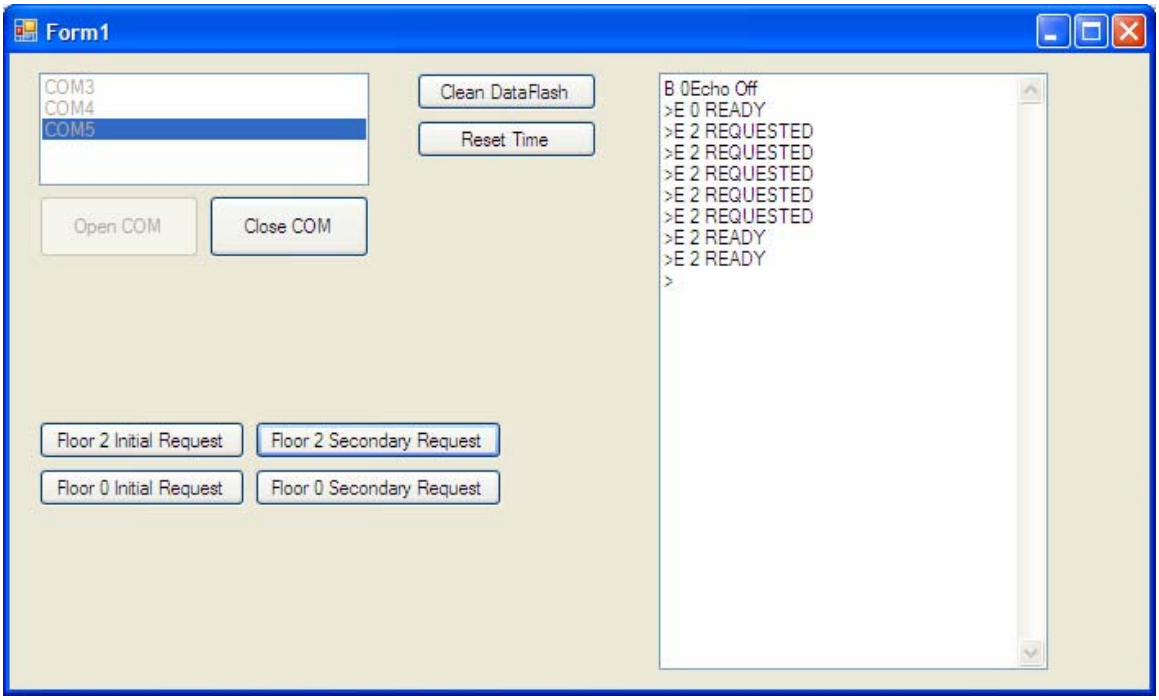

Figure 4: C# User Interface used to supply commands to the project module

VII. Conclusion

The C# language is very powerful and fairly easy to use. Atmel's Studio 4 was a tightly integrated environment, which took care of project management, took care of the WINAVR compiler, and provided a programming/debugging interface for the microcontroller. All of the software is free and unrestricted. The AVR Butterfly is a great development platform for trying out the functionality of Atmel's 8-bit AVR

microcontrollers. The Butterfly can be obtained for about $20 and can be programmed through the serial port right out of the box.

In the end, it would be difficult to verify the functionality of this project due to the lack of an elevator to try it on. The concept was tested and verified for functionality through simulation within the microcontroller and from the user interface on a PC. The concept is modular, so the testing only needed to be performed on single time blocks. All other time blocks should function the same way. Once some data was stored in a time block, the elevators were available much quicker the next time through.

References

[1] AVR 8-Bit RISC, http://www.atmel.com/products/avr/. Last Accessed February 10, 2007

[2] AVR Butterfly, http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3146. Last Accessed February 10, 2007

[3] ATmega169, http://www.atmel.com/dyn/products/product_card.asp?part_id=3012. Last Accessed February 10, 2007

[4] Mature AVR JTAG ICE, http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2737. Last Accessed February 10, 2007

[5] http://www.smileymicros.com/. Last Accessed February 10, 2007

[6] WinAVR, http://sourceforge.net/projects/winavr/. Last Accessed February 10, 2007

[7] AVR Studio 4, http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725. Last Accessed February 10, 2007

[8] Visual C# Express - Easy to Use, http://msdn.microsoft.com/vstudio/express/visualcsharp/default.aspx. Last Accessed February 10, 2007