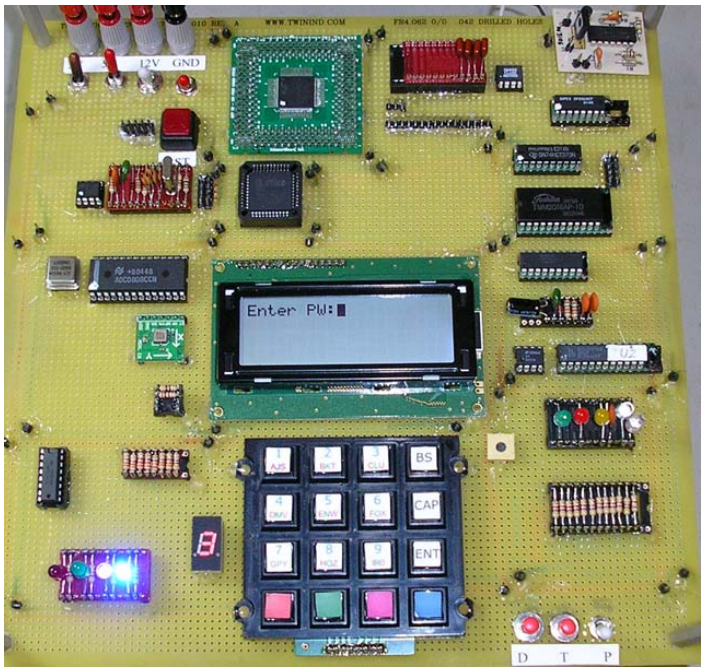
PROTEX-9000



ECET 3220 Summer 2008

Project Team:
Vincent Watkins
Wil Smith
Tyler Long

SOUTHERN POLYTECHNIC STATE UNIVERSITY
University System of Georgia

Table of Contents

List of Tables

List of Figures

Abstract

The PROTEX 9000 project is a demonstration of a stand alone embedded system. The featured components of the project are a C8051F022 micro-controller, 16-button keypad, a 20 x 4 LCD screen, an accelerometer, and visual and audible indicators. Some of the featured processes of the project are implementing analog to digital conversions, accessing external RAM, utilizing a 16 bit address bus with an 8 bit data bus, and employing serial communication to connect to an external complex device. The integration and control of these components and processes are made possible by explicit use of assembly and VHDL programming languages for both software development and hardware implementation. The scope of the project is to propose an original system, execute its hardware and software construction, and document the various methods that result in its creation.

1. Introduction

The PROTEX 9000 auto security system project is a stand alone embedded system. The primary component of the project is a C8051F022 microcontroller. The microcontroller is the central control for the system. The microcontroller processes CMOS and TTL voltage levels on its various port pins. The port pins are configured for explicit types of inputs and outputs. The types of inputs and outputs are discreet, analog, and serial. The microcontroller's Special Function Registers (SFR), are used to configure the microcontroller to process its various inputs and outputs. The developer will use assembly code to program the microcontroller to implement various hardware components for a multitude of functions and processes.

The secondary components of the PROTEX 9000 project are 16-button key pad to 20 x 4 LCD interface, +/- 1.2g accelerometer, and visual and audible indicators. The key pad is used to enter and edit data variables that are used by the microcontroller. The LCD gives visual feedback to show the values that are being entered or edited. The keypad is not directly connected to the microcontroller but is connected to a MM74C922 keypad encoder. The encoder allows the switches to be debounced so that valid input levels are sent to the microcontroller. Another component is the accelerometer. The accelerometer outputs an analog signal that represents a range of g-force values, +/- 1.2g. This device will send that signal to an analog to digital converter, ADC0808; the signal will be sampled and then digitized to discreet values and sent to the microcontroller via the data bus for processing. The final components are the various visual and audible indicators. LEDs are used to indicate system status and alarm conditions. Another visual indicator is the seven-segment display. This indicates the number of tries that the user has to enter the correct password. The there is also an audible indicator that is provided by a siren. The audio and visual indicators are not directly connected to the microcontroller. A GAL22V10 PLD is used to decode values that are sent by the microcontroller based on its inputs' status. The tangible components would be insignificant with out the processes that they employ and that employ them.

The processes of the PROTEX 9000 system are the underlying infrastructure of the main system. The main processes are the previously discussed analog to digital conversions, the transfer of data bits across the address/data bus, accessing the external ram, and employing serial communication to connect to an external complex device. The first process to discuss is the address and data bus. This bus is connected directly from the microcontroller to the CPLD, the address latch and the external RAM. The upper byte is connected to the CPLD which serves as a type of router for the system. The lower byte is both address and data bus. The lower byte is directed to a memory location on the RAM when the latch is enabled. After the addressed is "latched" then it returns to a data bus duty. The RAM is the external memory location that is used to store variables

and preset values. The final process to discuss is the serial communications feature of the PROTEX 9000 system. The serial output of the microcontroller is connected to a SP232-AEP which transforms the TTL level voltages (3.3V-5V) to RS232 standard levels, 3V to 25V. The serial connection is made to a computer via cable. The microcontroller will send preset alarm messages to the terminal. The successful development of this system relies on the marriage of the components and their shared processes. That union is made possible by superior software.

The software that functionalizes these components and processes is the Assembly and VHDL coding. The microcontroller is programmed through the use of assembly coding and the ispM4A5 and the GAL22V10 are programmed through the use of VHDL coding. The challenge of assembly coding is the timing and the flow of the program. The assembly code is processed from beginning to end one line after another. The VHDL code is processed all at once. The challenge of the VHDL coding is that complex coding, which is basically logic arrays, requires numerous gates to accomplish relatively simple outcomes. Thus only a finite number of gates constrain the size and power of the VHDL coding. So why use them? The CPLD and PLDs are very robust as compared to a microcontroller's capability to sink significant power levels and to withstand Electro Static Discharge (ESD).

2. Operating Instructions

2.1 Keypad

The interface for user options of the Pro-Tex 9000 is accomplished via a 16 key keypad. See Appendix for keypad layout diagram. The keys for user functions are the following 16 keys:

Enter (ENT) – The enter key allows for menu entry and exit. It also allows for the following: Password check, password change, and acceleration setpoint check.

Backspace (BS) – During password or new acceleration setpoint entry, it removes the previously entered character.

Caps Lock (CAP) – When selecting letters, the CAP function key will allow the user to select capitol letters.

Non-Function Keys (0-9 & A-Z) – 9 keys that select numbers or letters determined by the user selected function keys.

Red Function Key – Red mode enables user to select letters A-I.

Green Function Key – Green mode enables user to select letters J-R.

Pink Function Key – Pink mode enables user to select letters S-Z and the number 0.

Blue Function Key – Blue mode enables user to select numbers 1-9.

2.2 Entering Text

Entering text in the Pro-Tex 9000 system is an integral part of the password and set point features of the system. When prompted for a password or set point change, the user will be able use the Red, Green, Pink, Blue, Enter, Backspace, and Caps lock keys in conjunction with the alphanumeric keys to select the desired alphanumeric character combination for the password (or number combination for the Acceleration set point).

The Red, Green, Pink, and Blue keys determine what character will be selected when the user presses one of the 9 non-function keys. Each non-function key is labeled with the characters associated with it. For instance, the non-function key 1 can be used to enter the number 1 or the letters A, J, and S. The color of the character on the non-function key corresponds to the color of the mode button responsible for selecting that character. To select a number, first the blue key is

pressed. The user must then simply press the non-function key that corresponds to the desired number (except 0: 0 is selected using the pink key). Letters are broken up into three colors: Red, Green, and Pink. Pressing the Red key will select the red colored letters while pressing the Green or Pink buttons will enable the user to select the green or pink letters. The LCD will display an asterisk on the screen during password entry. The LCD will show the actual character being entered while at the change password screen or at the change acceleration setpoint screen. Once all characters have been entered, the enter key may be pressed to confirm the correct password has been entered, change the password in the "enter new password" screen, and also confirm that a new valid acceleration has been entered.

The backspace key will remove the last entered character and allow for re-entry of another character.

Pressing the Caps Lock key will allow the user to select only capital letters for entry into the system. To disengage caps lock, press key until caps lock indicator de-energizes.

2.3 Menu

2.3.1 Arming/Disarming the System

See Appendix for Screen Flow Diagram. Pressing enter from the Home screen will bring up a list of menu items. Select option 1 for Arm/Disarm. You are now at the Arm/Disarm menu. Select 1 for arming the system and 2 for disarming the system. You may also return to the Main menu by pressing enter key.

Arming the system will not prompt for a password in order to activate the alarm system. After arming, the menu will return to the Home screen.

Disarming the alarm will prompt for a password to be entered in order to disarm the system. After successfully disarming the system, the menu will return to the Home screen.

2.3.2 Changing the Password

The default password for the alarm system will be “1234.” Press enter to select Menu from the Home screen for a list of menu items. Select 3 for the Change Password option. The user will first need to enter the current password correctly. Once the current password has been validated, the user will be asked to establish a new password. The LCD will display “Password Changed” on a successful change of the password. The Home screen will appear following the password changed confirmation.

2.3.3 Changing the Acceleration Set point

The default acceleration set point is +0.75g. To change this set point, press the enter key from the Home screen. At the Menu screen, select option 2. The user will then be prompted to enter a new set point using the alphanumeric keypad. Since the range of the Accelerometer is from 0.0g to ±1.2g, any set point number entered that is outside of this range will result in an invalid set point and prompt the user to enter the set point again. Once a set point within the allowed range is entered, the system will show that the set point has been changed and will then return to the Home screen. The new set point will be displayed on the Home screen.

2.4 Features

2.4.1 Panic Switch

Initializing the panic switch will send the system into an alarmed state. The panic switch is disabled via the system being disarmed.

2.4.2 Accelerometer

The accelerometer gives real time feed back as to the g-force condition of the vehicle when the home screen is displayed. A phone text simulation warning will occur via HyperTerminal if the acceleration reaches the user determined set point.

2.4.3 LCD

The LCD displays system information via 20 x 4 monochrome screen.

2.4.4 Visual & Audible Indicators

Any alarm conditions met as described above will be annunciated via simulated text message from HyperTerminal with current alarm status.

A siren will sound upon a security breach or panic condition. In addition to the siren, a flashing light circuit will be energized as well.

2.4.5 7-Segment Display w/ Password Lockout

To preserve system password integrity, a built in password lockout feature is included. This consists of a seven segment display which will show the number of password entry attempts allowed before the system reverts to a locked out state. Once in the locked out state, a full system reset or power cycle is required to resume operation.

3. Hardware Description

3.1 Device Overview

The Protex-9000 Car security system makes use of an embedded 8051 microcontroller system. It multiplexes several external devices and switches to gather and store information input from the user as well as the status of the vehicle.

3.2 C8051F022

The core of the Protex-9000 system is the 100 pin TQFP Silicon Labs C8051F022 microcontroller. The F022 provides unparalleled power and system flexibility as a result of its numerous user friendly features and assembly language programming. First and foremost of these features is the F022's non-intrusive debug interface. Unlike conventional 8051's, the F022, uses a special USB debug adapter to download and debug code. The adapter allows the chip to remain in system and also allows code to be run real time on the chip. This eliminates the need for debug code and emulation software since the design can be tested immediately on chip and in circuit. The adapter utilizes the IEEE standard 1149.1 JTAG interface and connects to the F022 through the use of the Silicon Labs Integrated Development Environment (IDE) software. After the connection to the chip is made, downloading and debugging are as easy as the click of a mouse.

Another important feature of the F022 is the massive I/O capability of the chip. The F022 features 64 digital I/O pins that can be configured as push-pull or open drain and with or without pull up resistors depending on the operation required by the user. The Protex-9000 system utilizes 4 of the 8 ports (in push-pull mode) to

send and receive signals to other devices in the system. The flexibility of the F022 chip shines in this area since its I/O pin functions are determined by the user in software. For instance, normal 8051 chips have their address/data bus pins already predefined, but this is not the case for the F022. Its address/data bus pins can be selected on its high ports (P7, P6, and P4) or its low ports (P0, P2, and P3). The Protex-9000 system makes use of the high ports so that the bit addressable low ports can be used for priority functions such as serial communications and interrupts. These functions have dedicated pins as determined by the priority of the function. Serial communication is the highest priority on the chip and has the lowest port pins assigned to it. This is followed by the external interrupt pins and then finally the ALE, WR, and RD external interface signals.

Another excellent feature of the F022 is the ease at which it can be clocked and timed with other devices. The F022 can be configured to operate at variable oscillator speeds ranging from 2 MHz to 25 MHz. This is accomplished through use of either the onboard system clock or through the use of an external oscillator (operation is determined in software). The Protex-9000 system clocks the chip utilizing a 14.7456 MHz external crystal oscillator in order to facilitate proper baud rate generation for the serial applications of the system. The F022 is unique in that 1 clock cycle for the device equates to 1 machine cycle. This means that the majority of the device's commands can be accomplished in 1 to 2 CLOCK CYCLES. This feature makes timing using one of the chip's five 16 bit timers much more accurate since the instruction overhead is drastically reduced. The Protex-9000 system makes use of three of the five timers including using timer 4 solely as a baud rate generator. However, the most crucial aspect of the F022's timing is its ability to vary the length of the external interface signals generated from a "movx" command in software. The F022 has the extraordinary ability to lengthen or shorten the address setup and hold time, ALE, WR, RD, and data signals in software. This ensures that multiplexed device timing via address/data bus interface goes smoothly and without issue.

The F022 is not without its problems though. Since the chip is so large and powerful, it is important to connect every source of power to the chip. Any mistake in wiring can lead to confusing signals coming from the chip. In fact, one of the most difficult portions of construction was building the JTAG communication interface from the chip to the IDE. An entire day was consumed in the initial construction period because analog power and ground was not connected to the chip and therefore the debug adapter would not connect the system to the IDE. The analog connections were overlooked due to the fact the chip had three other digital power connections and it was thought that the analog connection was not needed. The important lesson is to check and recheck the datasheets for correct connection instructions and to verify each connection on the chip and on a diagram.

3.3 Ispm4A5 Complex Programmable Logic Device (CPLD)

The Lattice Semiconductor 44 pin, PLCC, ispm4A5 CPLD is a device that is the heart of the signal routing capabilities of the Protex-9000 embedded system. The CPLD's primary function is to decode the addresses output by the C8051F022 and translate them into signals to activate and deactivate external devices such as the LCD and ADC. Unlike the F022, this is accomplished through the use of the VHDL programming language. VHDL programming allows for higher level programming processes to be used therefore making device decoding extremely easy. The CPLD is a very robust device. It has 32 I/O pins allowing for many signals to be sent and received. However, to use such a large gate array to do simple address decoding would be somewhat of a waste. Therefore the CPLD was employed to de-bounce all the alarm input switches as well as send the alarm code word to the F022. This switch de-bouncing was accomplished in software through the use of VHDL programmed D flip flops. D flip flops though, require a clock signal input to function correctly. Since the CPLD does not have an onboard clock (it is largely an asynchronous device), a simple 555 timer was employed to provide a 2.5 KHz clock signal to the device therefore allowing the D flip flop to function.

The interface to the CPLD was just as easy as the interface to the F022. It is accomplished through the use of a JTAG standard connection. Programming is accomplished using the ispLever classic software provided by Lattice Semiconductor. The actual program that connects to the chip is the ispVM program also provided by Lattice. Once started, the ispVM program automatically scans the computer parallel port to look for the CPLD signature. Then all that must be done is load the compiled J-dec file and download it to the chip.

3.4 GAL22V10D

The Lattice Semiconductor GAL22V10D was the device of choice to interface nearly all of the systems power intensive items. Since it was desired to buffer the control chips from all of the high current devices, the GAL was employed to act as a go between for the control chips. The GAL's ability to sink larger currents made it the ideal IC to drive all the system LED's as well as the opto relay to drive the audible alarm circuit. Like the CPLD, the GAL is programmed using VHDL. However, the programming for the GAL is quite simple since it is really only interfacing LED's. In fact, the most complicated algorithm on the GAL alternately flashes two LED's in time with a 1 Hz clock provided by another 555 timer (the GAL is also largely asynchronous). The physical programming of the GAL is accomplished through the use of the Max loader programming software by EE tools and a Unimax programmer.

3.5 ADXL213 Accelerometer

An interesting feature of the Protex-9000 system is the ability to measure vehicle acceleration and tilt. The device responsible for generating these signals is the Analog Devices 0-5 volt ADXL213 Dual Axis Accelerometer. The ADXL has two modes of operation: Normal and Tilt. In normal operation the IC is parallel to the ground and measures acceleration in the X and Y directions (the Protex-9000 system employs the X direction only). At 0.0g of acceleration, the output on the X_A pin will be approximately 2.5 volts. As acceleration increases in the direction the X arrow is pointing, the voltage on the X_A pin will rise until it reaches 5 volts (indicating +1.20g). If the acceleration is in the opposite direction of the arrow, the voltage on the X_A pin will lower until it reaches 0 volts (indicating -1.20g). In tilt mode, the ADXL chip needs only to be tilted to register a change in voltage. The chip's sensitivity to the tilt depends on how close to perpendicular with the plane of gravity it is. The closer to 90° the less sensitive the chip is. The G forces registered depend on the direction of the tilt (positive or negative). The Protex-9000 system does not directly employ the tilt function as part of its normal operation though the tilt can be used to test the user defined acceleration set point limit of the system.

3.6 ADC0808CCN

In order to transform the analog signal coming from the ADXL213 to a usable digital signal, the Protex-9000 uses the National Semiconductor ADC0808CCN analog to digital converter. The 0808 is a multi-channel successive approximation converter that can multiplex multiple analog inputs and send out an 8 bit equivalent of the voltage. The Protex-9000 system only has one analog input. Therefore the 0808 is configured to use its lowest channel at all times (IN0). Since the Protex-9000 system uses a start and wait method to obtain the digital word from the 0808, a fast oscillator (1 MHz) was used to clock the chip. In this way the wait time is reduced somewhat. Since the ADXL213 is a 5 volt device, the logical choice for the positive reference voltage of the 0808 is its Vcc voltage of five volts. The negative reference was chosen to be ground since the ADXL213's lowest voltage is zero volts.

The 0808 communicates to the F022 via the F022's address/data bus with the address decode provided by the CPLD. When a conversion is desired, the software initiates a "movx" with the data pointer loaded with 3800h. The CPLD decodes the 3800h and enables the start conversion on the 0808. The F022 will then wait 116 µs for the conversion to finish. When the time has elapsed, the F022 initiates another "movx" with the data pointer loaded with 3000h. The CPLD decodes the address and enables the output enable pin of the 0808. The F022 will have the digital value in its accumulator after the "movx" is complete.

3.7 TMM2016 RAM & SN74HCT373

The Toshiba TMM2016 RAM chip and Texas Instruments SN74HCT373 latch are used in tandem by the Protex-9000 system for external volatile data storage. The RAM chip is used to store the current acceleration decoded by the F022 as well as the current security password entered by the user. Since the 2016 only has 2048 addressable locations (7FFh), one would normally need to take care as not to fill the chip up inadvertently. However, this is not an issue with the Protex-9000 system since only 13 addresses are used on the chip (2000h-200Ch). However, the information stored on the chip is an important part of the system operation. In order to access this information correctly using the multiplexed address/data bus scheme, the 373 latch must be used. The latch's function is to hold the low byte of the address during a "movx" that writes to or reads from the RAM chip. This function is needed since after the ALE pulse from the F022, the low byte of the address is replaced with the data. If not for the latch, the RAM chip would never have the appropriate address location sent to it. The RAM chip is activated from the CPLD decoded address that determines if a read signal was sent or a write signal was sent (the RAM chip enable was tied low such that the chip itself was always enabled). The interface to the chips seems difficult but in reality it was very simple to build and work with.

3.8 MM74C922

The Fairchild Semiconductor MM74C922 16 key encoder is the chip responsible for determining what keys were pressed on the Jameco keypad and relaying that information to the F022 microcontroller. This tactile interface is the primary way that the user interacts with the system. The key encoder is not a complex device. It senses a key press and develops a 4 bit word that equates to the key that was pressed (0000 to 1111 – 16 keys). When a key is pressed, the 922 will signal the CPLD using its data available pin. The CPLD will invert the signal and send it to an external interrupt pin on the F022. The F022's program will vector to an interrupt service routine that will perform a "movx" command. This "movx" address will be decoded by the CPLD to enable the 922 to put the code for the key on the data bus for interpretation by the F022. From this point on, the function of the key is determined by the F022's programming algorithms.

3.9 20x4 Liquid Crystal Display

One of the most interesting and complex devices employed by the Protex-9000 system is the Optrex 20x4 LCD display. The LCD is an integral part of the Protex system as it is the primary visual interface for the user. The LCD is normally very difficult to interface to because of how slow the device is in its command pulses. This problem is overcome by the timing interface of the F022. Since the interface pulses from the F022 can vary in length (determined by software), the microcontroller has no problem in syncing up with the LCD. The interface with the

LCD is the most complex interface in the system. The LCD is on the address/data bus much like almost every other device. However, the LCD needs separate control signals sent to it in order to operate correctly. These control signals are developed by a combination of the F022 and the CPLD devices. The F022 will send special addresses out on the address bus for the CPLD to decode. Each command for the LCD has specific address related to it. For instance, the enable of the LCD has its own address that the CPLD decodes to activate the enable pin of the LCD. In this way, all functions of the LCD can be used and the LCD can be fully integrated into the system.

3.10 SN74LS47

The Texas Instruments SN74LS47 BCD to 7 segment decoder is a small but noteworthy edition to the Protex-9000 system. The Protex system uses this chip to save valuable pin I/O on the CPLD. Instead of using 7 of the CPLD's pins to directly drive the seven segment, the 7447 is employed to conserve I/O. In this way the F022 can directly send the password count to the Seven Segment without the need for a go-between chip. Since the Protex system only has a password attempt of three, the two upper bits of the BCD word can be tied to ground.

3.11 SP232AET

Since the Protex-9000 system sends serial communications to an operating PC, the CMOS voltage the F022 output must be translated into RS-232 signals that the PC's serial port can understand. This is accomplished using the Sipex SP232AEP TTL/CMOS to RS-232 converter. This particular chip uses a series of capacitors to collect enough voltage to translate a TTL or CMOS signal to a higher voltage RS-232 signal. The higher voltage is required by most PC serial ports for correct serial communication. The Protex system only uses the 232's transmit terminals since incoming transmissions are not needed by the system and are therefore not connected.

3.12 General hardware interface and operation

When the system is first turned on, the F022 begins constant communication with the CPLD and LCD to display the interface screens. The F022 also initiates a RAM write to load the default password. Once on the first enter password screen, the F022 enables its interrupts allowing it to interface with the 922 encoder. On the "Home" screen, the F022 begins an interval timed reading and storing of the acceleration from the 0808. The F022 also writes the acceleration to the 2016 RAM chip for later comparison and analysis. Since the Protex system is interrupt driven, the F022 is constantly looking for the panic alarm input. When the system is armed, the F022 will look for the panic, tamper and door ajar switch inputs. Once the system verifies that an alarm has been activated it will energize the associated LED (through the GAL22V10D) and activate the audible alarm circuit

via an opto relay. Also when an alarm condition is received, the F022 will communicate a string of ascii characters to the SP232 to send to a serially connected PC. Once the alarm is disarmed the F022 will turn off all activated alarms and allow further alarms to occur. When the system acceleration is over set point, the F022 will initiate a serial string that will alert the user as well as show the acceleration that caused the alert. This ability is allowed to happen once until the system is disarmed. Then it is allowed to happen again.

4. Software Description

The microcontroller was initialized in the beginning of the main routine. The various parameters used during initialization will be discussed more in depth herein.

The Man-Machine Interface (MMI) to our microcontroller was reliant entirely upon an interrupt driven system. Those driven by external interrupts were the 4x4 keypad and the alarm switches (Door ajar, Tamper, & Panic). An internal timer interrupt was used in order to periodically receive the current acceleration of the system.

A state machine was developed in order to determine which screens were shown. Each screen or menu had a dedicated state or number assigned to it. This allows for ease of use when a state change is desired within the program.

Variable declarations were used to streamline the programming process prior to assembly time. Changing a variable declaration will change all instances of the variable within the program.

4.1 Main.asm

The “Main.asm” file contains all org statements for the following: reset vector, Keypad vector (/INT0), alarm vector (/INT1), and Timer 1’s vector. On startup or a reset, the program branches to the reset vector location 00h. The program is then sent to the tag “Main:”.

The first item within “Main:” initializes the stack pointer to 30h. A problem with the stack overwriting the current state location in scratch pad ram (21h) was found once the state machine was completed. All of the lcall & push directives used within the state machine allowed the stack to begin reaching the memory locations of scratch pad ram that were integral for the program to function properly.

After initializing the stack pointer, the main routine initializes the following devices: the microcontroller, LCD, and RAM. The individual subroutines will be discussed within their individual modules.

The scratch pad RAM locations 28h through 2Bh are cleared by the main routine. These locations hold the user entered password. If not cleared, then the user would simply be able to press enter on reset as long as a correct password has been entered at least once. The main routine also sets the initial setpoint to 0.75g by loading scratch pad RAM locations 25h through 27h with the ascii values that correspond with the setpoint. Otherwise, the system would startup with an unknown setpoint.

Once the internal RAM locations have been setup properly, the main routine starts the state machine with state 00h. When all three preliminary states have been shown, the main routine then becomes the resting point for all routines with "sjmp $". The program will branch accordingly once an interrupt occurs.

4.2 Init.asm

The "Init.asm" file contains all necessary parameters for setting up the microcontroller. For initializing the device, an "lcall" to the tag "Init_Device" from the main routine takes place. The "Init_Device" tag was responsible for the order in which the different functions of the chip are setup.

4.2.1 Reset_Sources_Init

This routine disables the watch dog timer. This was important since the system relied upon "sjmp $" to function properly. When the program counter holds its value at "sjmp $" for a certain amount of time, the watch dog timer would normally initiate a system reset. Disabling this timer allowed for a completely interrupt driven system.

4.2.2 Timer_Init

In this routine, timer 4 was setup as a baud rate generator for UART 1. Timers 1 & 0 were setup as 16-bit timers with their SYSCLK divided by 12. Timer 4 overflow was setup to be the RX & TX clock source for generating UART 1's baud rate. Timer 4 was also told to run and initialized with the count that correlates to a baud rate of 115.2k baud (RCAP4L=0FCh & RCAP4H=0FFh).

4.2.3 UART_Init

Baud rate divider was disabled in this routine. UART 1 was setup as having an 8-bit variable baud rate.

4.2.4 EMI_Init

P4, P6, & P7 were being used as the address/data bus. The internal xRAM that is on-chip was disabled. The ALE signal was extended by 4 SYSCLK cycles. The address setup time was extended by 3 SYSCLK cycles. The /WR & /RD

signals were extended by 12 SYSCLK cycles. The address hold time was extended by 3 SYSCLK cycles.

4.2.5 Port_IO_Init

TX1 & RX1 were setup on ports P0.0 & P0.1 respectively. The keypad external interrupt was setup on port P0.2. The alarm interrupt was setup on port P0.3. The entire port of P1 was setup as Open-Drain and in digital mode. The entire ports of P2 & P3 were setup as Push-Pull and in digital mode.

4.2.6 Oscillator_Init

The microcontroller was setup in crystal oscillator mode with an operating frequency greater than 6.7MHz. A 14.7456MHz crystal was used to facilitate a lower theoretical percent error when generating common baud rates. Once the new external crystal oscillator becomes stable, the routine disables the internal system clock and uses the external crystal instead.

4.2.7 Interrupts_Init

This routine enabled global interrupts and set up the interrupt priority. Timer 1 and /INT1 were given high priority while the keypad was left as is.

4.3 ADC.asm

On an overflow of Timer 1, the program vectors to “ADC_GetAcc”. The whole purpose of this module was to obtain the current acceleration of the system, write the current acceleration to external RAM, check to see if the current state allows for printing the current acceleration to the LCD, and determine whether or not the current acceleration had exceeded the user defined setpoint.

4.3.1 ADC_GetAcc

This routine is initially called upon during an overflow of Timer 1. It was important to save any pertinent SFRs and/or registers used within this routine prior to the execution of any commands. The ADC converter was commanded to begin conversion of its current analog value. A delay was placed before reading the digital output of the ADC since the maximum time for proper conversion was 116µs. Once the time delay elapsed, the digital value from the ADC was imported.

If the current state was 06h (Home screen), the current acceleration was printed to the LCD. If the current state was anything but 06h, the subroutine finished and returned from interrupt.

4.3.2 ADC_Convert

On entry to this subroutine, the current acceleration had already been translated into a 0 to 255 value. A lookup table was created with seven ascii characters to represent the following: plus or minus symbol, ones digit, decimal, tenth's digit, hundredth's digit, the ascii letter 'g', and a null terminating character. 256 strings existed for every possible value of acceleration that could be reached within an 8-bit register. A null character ended each string to assist in ease of programming. Since a value of 00h in acceleration corresponds to the first string of "-1.20g" nothing needs to be done to the DPTR. For any other acceleration, the value ranging from 0 to 255 was multiplied by 7 (7 characters per string) such that a value was obtained which corresponds to the number of times the DPTR should be incremented (assuming the DPTR is already pointing at the first string in the lookup table) in order to point to the respective first character of the correct acceleration.

4.3.3 ADC_Compare

This routine brings in the current acceleration from external RAM and compares it to the user entered setpoint. The routine compares from the MSB to LSB of the acceleration. The current acceleration must be greater than the setpoint for the routine to consider it a valid alarm state. On a valid alarm condition, the routine then calls the ADC_Serial_Print routine.

4.3.4 ADC_Serial_Print

This routine first checks to see if it is allowed print a serial string of the exceeded acceleration setpoint to HyperTerminal. A limit of 1 print per a disarming of the system was imposed so that an overrun of the HyperTerminal would not occur. When allowed to print, the Acceleration Alarm string was printed first. This was followed by the current acceleration string from external RAM to be printed beneath the Acceleration Alarm string.

4.4 Alarm.asm

The "Alarm.asm" module contains all subroutines related to a potential alarm condition. /INT1 (active low) was triggered on activation of the Door Ajar, Tamper or Panic switch. When /INT1 triggers, the program vectors to "Alarm_Check".

4.4.1 Alarm_Check

The panic switch was the first to be checked. The system does not have to be armed for the panic switch to activate the alarm. If the panic switch was tripped, then the program activates the alarm and then branches to the Alarm_Ser_Panic

routine after disabling /INT1. The system must be disarmed before another alarm (of any kind) is allowed to take place.

If the panic switch was not tripped, then the program checks to see if the tamper or door alarm switches were the cause of the interrupt. If the tamper switch was tripped, then the program activates the alarm and then branches to the Alarm_Ser_Tamper routine after disabling /INT1. If the door ajar switch was tripped, then the program activates the alarm and then branches to the Alarm_Ser_Door routine after disabling /INT1. Otherwise, the program exits the routine.

4.4.2 Alarm_Ser_Panic

If the panic switch was tripped, then the program loads the DPTR with the respective alarm string and proceeds to print the string to HyperTerminal via serial communication.

4.4.3 Alarm_Ser_Tamper

If the tamper switch was tripped, then the program loads the DPTR with the respective alarm string and proceeds to print the string to HyperTerminal via serial communication.

4.4.4 Alarm_Ser_Door

If the door ajar switch was tripped, then the program loads the DPTR with the respective alarm string and proceeds to print the string to HyperTerminal via serial communication.

4.5 Key.asm

Key.asm is the largest and most complicated module that the system relies on for proper operation. The program will vector to Key_ISR anytime a key is pressed (the key pad is operated via /INT0). From there, the program determines what key was pressed and then checks the system state to determine the rules associated with the key that was pressed. The program will branch accordingly and call the state machine to change the current state if warranted.

4.5.1 Key_ISR

Key_ISR is directly jumped to from the /INT0 vector location. Once in the routine, the key value is read into the system and then bit masked to ensure that the accumulator contains a value from 00h to 0Fh. The program then uses subtraction to determine if a function key was pressed. Otherwise the program assumes that a non-function key was pressed. If the program determines that a function key was pressed, it branches the respective tag within Key_ISR that

corresponds to the key that was pressed. Once the program has completed all the subroutines associated with the key pressed, it polls P0.2 (/INT0). The program will not exit the ISR until the user releases the key thus preventing the ISR from continually looping while the key is held down.

4.5.2 Key_Func_Ent

The program branches to this subroutine if it is determined that the enter key was pressed. Each sub-tag within this routine first checks for a valid state in order to determine proper program branching. If the enter key is pressed in a condition that requires a password check, the program calls a password compare routine to determine further program branching.

Certain states require menu changes that are directly controlled by the enter key. When the enter key is pressed in these states, Key_Fun_Ent will change the state and branch the program accordingly.

One state requires that a valid acceleration setpoint be entered before the program can advance to the next screen. When the enter key is pressed in this state, Key_Func_Ent will call a routine that will determine if a valid setpoint has been entered.

4.5.3 Key_Accel_Valid_Check

This routine is called by Key_Func_Ent on a press of the enter key during state 0Eh. Since the entered setpoint may not exceed 1.20g, the program begins by comparing the MSB of the user entered setpoint. If the program determines that the MSB of user entered setpoint is greater than one, the program branches to the invalid setpoint state. Otherwise, the program will determine if the entered number is a zero or a one. If the ones digit is equal to zero, the program checks the tenth's and hundredth's digit for a values less than or equal to 39h (ascii 9). If the ones digit is equal to one, and the tenth's digit equal to two, the program will check the hundredth's digit to ensure that it is not greater than zero. If the ones digit is equal to one and the tenth's digit is less than two, then the hundredth's digit must be less than or equal to 39h. Otherwise the program will branch to the invalid setpoint state.

4.5.4 Key_PW_Check_XXh

Since there are multiple states that require the same password check, the "XXh" above represents any state that requires this function. This routine determines if the user entered password matches that which is stored in external RAM. The program checks from MSB to LSB and will branch to load either an invalid password state (dependent on number of attempts) or will load the next state for a correct password.

4.5.5 Key_Func_BS

This routine is called upon if Key_ISR determines that the backspace key was pressed. The program simply checks the state to determine if the backspace is allowed to be pressed or not. If the program is in a valid backspace state, it calls Key_BS_Resolve to perform the backspace function.

4.5.6 Key_BS_Resolve

This routine is called by Key_Func_BS. It determines if the backspace key is allowed to delete characters based on the value held by register R2. If R2 holds a value greater than zero, the backspace key commands the LCD to locate its cursor one space to the left and writes the space character 20h. The program the shifts the cursor back to the left once more to counter the effects of the LCD auto increment feature. R2 is then decremented. If the subroutine is entered with R2 holding zero, the program exits this routine. This prevents the program from being able to backspace unless a non-function key is entered. R0 is used as a pointer to either the MSB of user entered password or user entered acceleration setpoint. Each time the backspace is pressed (assuming that it is allowed to be pressed) R0 is incremented such that it points to the previous entered password or acceleration character.

4.5.7 Key_Func_Caps

The program calls the Key_Func_Caps routine if Key_ISR determines that the caps key was pressed. This simple routine compliments bit 07h and P1.3 to internally set/clear the caps flag for the program to examine later. P1.3 controls the output for the caps LED.

4.5.8 Key_Func_(Color)

The “color” in parenthesis refers to the blue, pink, green, and red subroutines. Each of these routines is the same except for the bits that they toggle. This routine is initially called by Key_ISR if the program determines that one of the colored keys has been pressed. Pressing of the colored function keys can happen independently of the system state. Therefore regardless of what screen the program is on (except the system locked screen) any of these buttons can be pressed.

4.5.9 Key_State_Chk

If no function key was pressed, Key_ISR will assume that a non-function key was pressed and will call Key_State_Chk. The program simply checks the state to determine if a non-function key is allowed to be pressed or not. If the program is in a valid non-function key state, one of three possibilities can occur: 1. the state calls for Key_Func_PW, 2. the state is directly changed, or 3. the state calls for

Key_Func_Accel. If the current state does not allow for non-function key entries, the program returns.

4.5.10 Key_StateXXh_Menu

This routine is called from Key_State_Chk. The “XXh” refers to either state 07h or 08h since they both use non-function keys to determine menu choices. The program first checks to see if a non-function key that corresponds to a valid menu choice was pressed. If the key was valid, the program directly branches to the state that corresponds to the menu choice. If the key was not valid, the program exits.

4.5.11 Key_Func_Accel

Key_State_Chk calls this routine if the program is in state 0Eh. First the routine checks if three non-function keys have already been pressed. If so, then the program exits. If not, the program continues to check if either the blue or the pink function keys are active. If the neither blue nor pink are active, the program will exit. If a valid colored function key is active, the program will increment R2 (shows that a valid non-function key was pressed). Since the accumulator enters this routine with the bit masked value of the non-function key pressed, the DPTR will point to the correct character as indicated by the color of the function key. After the correct character has been chosen, the program will move the ascii value for that character into internal RAM as indicated by R0. The program will also print the character to the LCD. R0 is then decremented to point to the next location in RAM in which to write the next setpoint character.

4.5.12 Key_Func_PW

Key_State_Chk calls this routine during a password entry state. The routine checks to see if four characters have already been entered based on the value held in R2. If R2 is equal to or greater than four, the program exits. Otherwise R2 is incremented and the program will continue. Next, the program checks to see what colored function key is active and branches to the respective color tag. It will further choose the correct color look up table based on the status of the caps lock bit (07h). After the correct character has been chosen, the program will move the ascii value for that character into internal RAM as indicated by R0. R0 is then decremented to point to the next location in RAM in which to write the next password character. If the program is not in state 14h, the program will print an asterisk to the LCD instead of the actual decoded character.

4.6 LDC.asm

This module is responsible for initializing, writing to, and controlling the various functions of the LCD. All lookup tables for the range of screens that the LCD displays are located within this module.

4.6.1 LCD_Init

This subroutine is called by “main:” in the beginning of code for the following: Sets the display to 8-bit mode & 5x8 dots, turns the LCD on w/ flashing cursor, and commands the LCD to auto increment the DDRAM address of the LCD after each write command.

4.6.2 LCD_Print

This subroutine must be entered with the cursor in the correct location, the DPTR pointing at the lookup table string to print, and the ACC must be pointing to the desired location within the string. This routine is one of the most called upon throughout the entire program. A loop statement was written such that it will continue printing to the LCD until a null character is reached within the string. A null character causes the routine to exit and restore the values saved at the beginning.

4.6.3 LCD_Wait_3sec

This routine was called during screen transitions which required a delay according to the screen flow diagram located within the appendix. R2 was preloaded with a value calculated such that an increase by 20 added another second to the delay time. R2 was then decrement on each overflow of timer 0. Once R2 became zero, the timer was stopped and the program was allowed to exit.

4.6.4 LCD_Clear

This routine is primarily used within the state machine when a new screen is to be printed. When this routine is called, the LCD writes the space character 20h to all visible DDRAM addresses and returns the cursor to the home location.

4.6.5 LCD_Busy

Even though there are only three lines related to this routine, it is by far the most used out of all LCD routines. Each LCD routine must use this routine after each

command to ensure the display has completed its current instruction before processing another.

4.7 RAM.asm

This module is responsible for initializing, writing to, and reading from the external RAM. It is used during password changes and during acceleration updates. A memory map was used to designate what was to be stored in each location of the RAM. This was done before any routine was written for the RAM. During the testing phase, the RAM chip was extracted before startup to determine how integral the device was to operation of the project. Upon reaching the third screen, the system crashed and began to print garbage to both the LCD and HyperTerminal. Once the RAM chip was restored, the project began operating as designed.

4.7.1 RAM_Read_PW

The purpose of this routine was to extract from the pre-designated location in RAM the stored password. This routine is accessed during a request to compare the user entered password to the current system password. R0 is used as a pointer to take the current password from external RAM and store it within internal scratch pad RAM.

4.7.2 RAM_Write_PW

This routine is accessed during a change of password request and only after the current password has been entered correctly. This routine moves from internal to external RAM the changed password starting with the LSB ascii character first.

4.7.3 RAM_Write_ADC

On each overflow of Timer 1, this routine is accessed from the “ADC.asm” module in order to update the external RAM with the current acceleration of the system. The most difficult part of this routine was in designing a loop that would juggle two different DPTR addresses. One DPTR address belonged to the lookup table corresponding to the current acceleration while the other equated to the space in RAM where the current acceleration was to be written. Creating a loop of this sort saves internal scratch pad RAM such that the lookup table may be read and the value immediately written to RAM for use at a later time. The key to juggling the two DPTR’s lies in the ability to save the low byte of the DPTR before it is overwritten by the address of the lookup table.

4.7.4 RAM_Init

The purpose of this routine was to set the initial password and to assist in desiging a loop making it easier to print the over setpoint acceleration to the

HyperTerminal. After the RAM was initialized with the default password, it was followed by the Carriage Return character “0Dh”, Line Feed character “0Ah”, and the null character “00h”. By having the carriage return, line feed, and null character follow the current acceleration memory location in external RAM, the ADC_Serial_Print routine could be programmed as a loop and wait until a null character was reached to exit. The main purpose of the carriage return and line feed was to reset the HyperTerminal to the home position on the next line awaiting new alarm conditions.

4.8 State.asm

The purpose of this module was to create an easy way to setup rules and display screens for the system. From outside, this module is only accessed through the State_Lookup tag which will be described herein.

4.8.1 State_Table

All “dw” directives within this table hold the 16-bit address of the tag referenced after assembly time is complete. This 16-bit address will be used later as an index by State_Lookup to point to all state machine instruction tags.

4.8.2 State_Lookup

This routine is entered with the ACC equal to the desired state. First, the DPTR is loaded with the first address within the State_Table as described above. Since the “dw” directive defines two bytes within code memory, the initial value of the ACC must be multiplied by two in order to point to the high byte of the desired state. For instance, if State_02 was desired, the ACC would contain 02h on entry into State_Lookup. There are four bytes (State_00 & State_01) above State_02h. These four bytes need to be skipped. When the ACC is multiplied by two (rotate left through carry w/ carry being cleared), the new ACC is now equal to four. According to the instruction “movc A,@A+DPTR”, the new ACC is now pointing to the high byte of State_02. For the state machine to function properly, the low byte should be pushed onto the stack first. To accomplish this, the high byte is saved by temporarily moving the high byte pointer that the ACC holds to R0. The ACC is then incremented to point to the low byte of State_02. Now the low byte can be pushed onto the stack. Note that the low byte is also pushed onto the stack first when an “lcall” instruction is used. Once the low byte has been saved to the stack, the high byte is then restore from R0 and placed back into the ACC. Now that the high byte is in the ACC, it can now be pushed onto the stack. As soon as the program reaches the “ret” at the end of this routine, the program updates the program counter with the 16-bit address that was obtained through using the State_Table as an index. The program then branches to the desired state.

4.8.2 State_XX

This section represents all states being called by State_Lookup. The “XX” corresponds to states 00h through 15h. All states are inherently similar by means of: Calling routines which print strings, placing the cursor in the correct position, enabling/disabling interrupts, updating the seven segment display, update the current state, and setting up the rules for both function and non-function key presses.

5. Conclusion

The development of the PROTEX 9000 system encompassed a gambit of challenges. The main challenges of the project's development were related to facilitating the implementation of unknown IC components and developing the code necessary to enlist the IC's functions, and a positive team dynamic. These challenges were decimated by innovative technical expertise involvement and goal oriented execution.

The successful completion of the PROTEX 9000 project showcased the proven skills, technical prowess, and the balanced efforts of a spectacular development team. The combined skills of the PROTEX 9000 development team are phenomenal. The predominantly high levels of electronic and digital knowledge that surpassed academic levels were absolute. One of the technical challenges for this project was the unfamiliarity of the new C8051F022 micro-controller. It was selected for the project because of it numerous I/O ports and serial availability, its processing speed, and its debugging features. Hours and hours of data sheet research, discussions, and heuristically based methods occurred before the power of these amenities in the PROTEX 9000 system could be enlisted in the project's system. Once the central control component was "mastered" the integration and implementation of the other devices began. Those main components were of course the 20 x 4 LCD screen, the 4 x 4 keypad, the 2016 external ram, the accelerometer, and the ADC0808CCN. The LCD initially presented challenges that related to the enable bit. This was resolved by realizing that the enable pin on the CPLD that enabled the LCD held the key to the timing scheme of the LCD's implementation. The challenge of the key pad involved interrupt control and establishing the true state of the key that was pressed. The next issue concerned the understanding how the address and data bus connected to the latch and to the RAM, thus providing the lower bytes of the address bus and the entire data bus occupied the same eight pins. The remaining component for consideration was the ADC device. Its major requirements lie with the hard wiring of significant pins. After the marriage of the system's components was complete, their successful coding was next.

Programming in assembly or VHDL is just like writing literary prose or poetry. It has been said that "To the experienced programmer it is a work of art". The teams approach to the code writing is a unique method that is finding its way into present industry standards. We were glad to see that by imitation that there is evidence of intelligent life out there. The team's innovative approach was to co-write the code simultaneously. Two persons would sit at the computer and code each line together. One person would type the syntax, but each person would cite the pseudo code, create the actual code and comment its entry. This proved to be a very efficient method for writing the project's code because it instantaneous gave valuable feed back and discussion that magnificently fine-tuned each line's logical placement and realization. Valuable experience was gained and skills were enhanced during to this process. After its completion, the project's perfect performance was an affirmation of masterful proportions. The level of success is directly related to the superior combination of each team member's contribution.

The team's professional attitude was a definitive resource that inspired personal and professional growth. The end-game scenario for this project was to realize a system that precisely functioned as described in the initial proposal. The pseudo client's needs were often considered when operational decisions were necessary to be made. The end-game scenario was the motivation to execute the tasks that led to the projects successful completion. Also the persistence to do the highest quality of work and pay the closest attention to the smallest detail proved to be the integral guide line for the team's success. The measure of any success is always based on the challenges and circumstances that were overcome. Don’t be discouraged when proved wrong. Another lesson learned is that of humility and team integrity. Don’t be discouraged when a team mate has proven your approach to be wrong. Be humble and maintain focus on the team's agenda and not individual ones. The accomplishments of this project are not just a functional stand alone embedded system. The value of this project's success is the technical skill development and the professional and personal growth that it inspired.

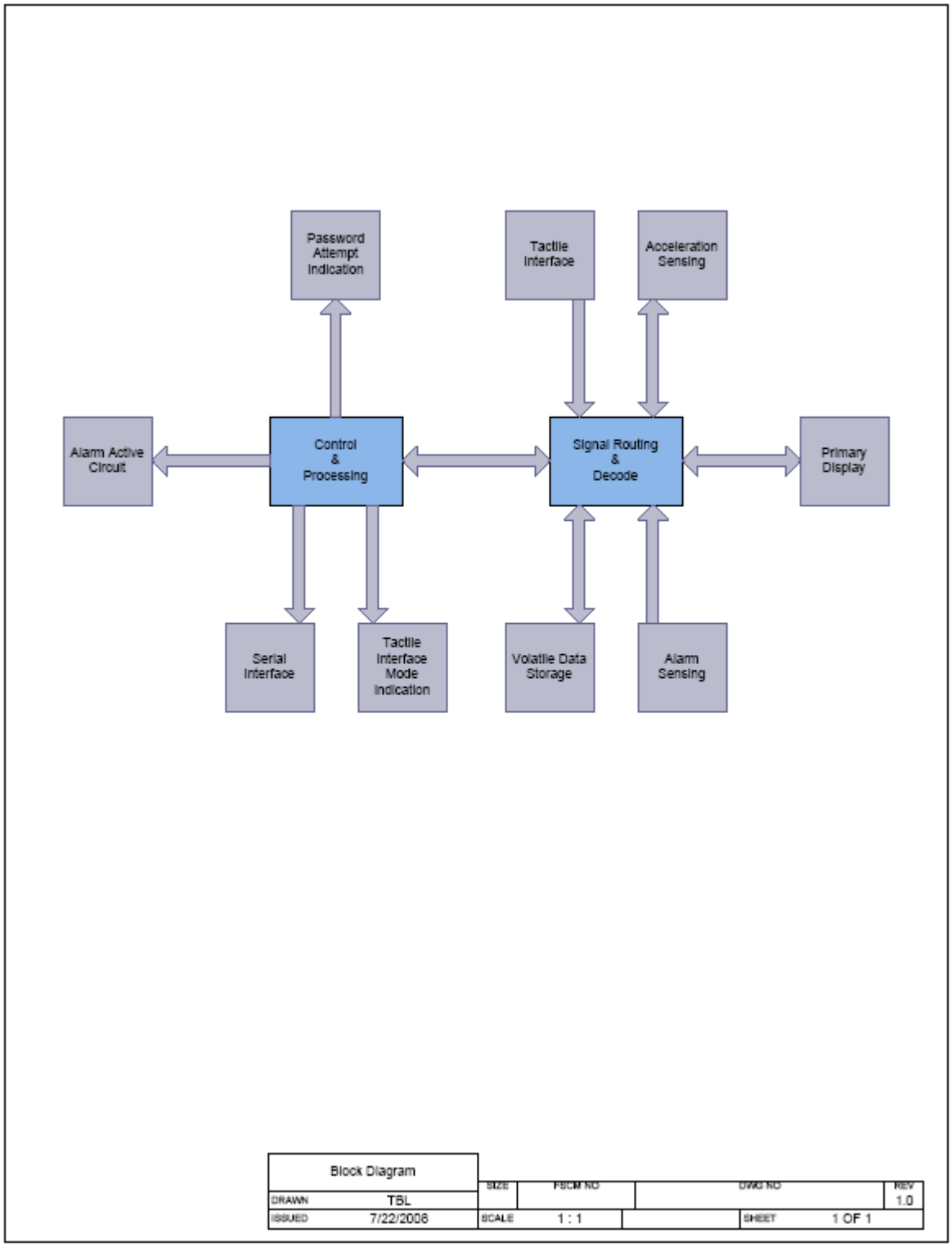


Figure 1: Block Diagram

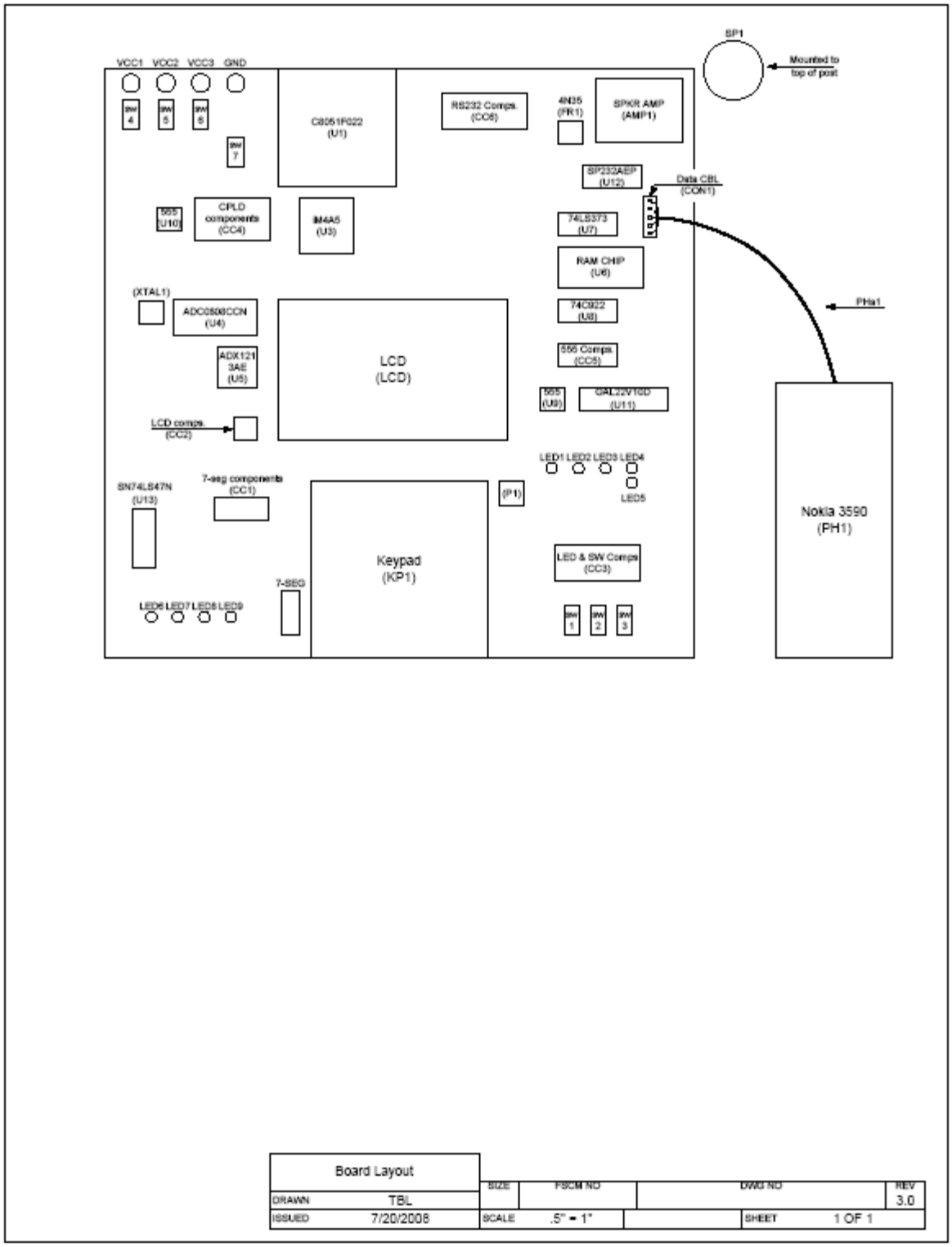


Figure 2: Board Layout

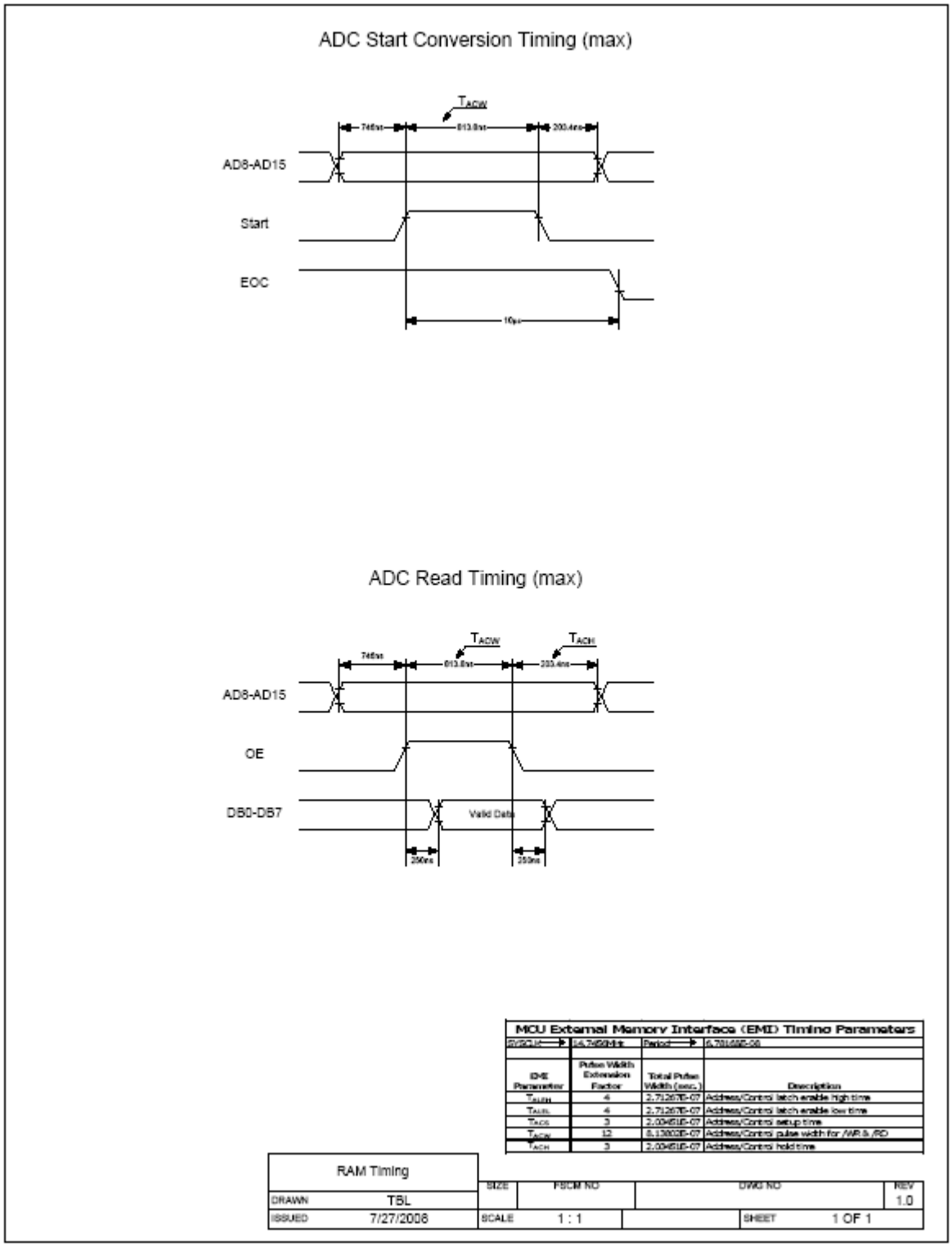
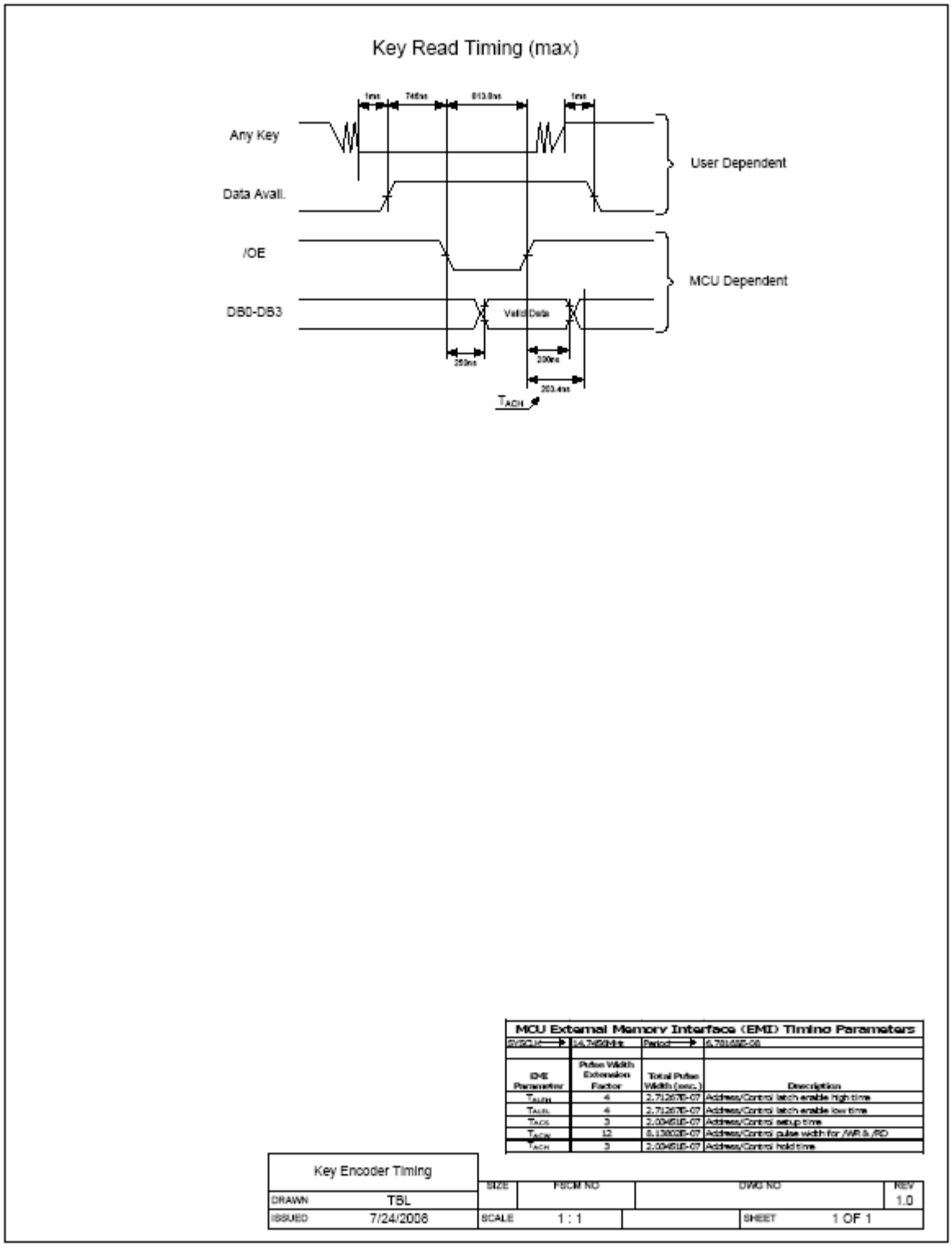


Figure 3: ADC Timing



MCU External Memory Interface (EMI) Timing Parameters

SYSCLK →	64.7450MHz	Period →	4.7814E-08
EMI Parameter	Pulse Width Extension Factor	Total Pulse Width (sec.)	Description
TALEH	4	2.7126E-07	Address/Control latch enable high time
TALEL	4	2.7126E-07	Address/Control latch enable low time
TACS	3	2.0345E-07	Address/Control setup time
TACW	12	8.1380E-07	Address/Control pulse width for /WR & /RD
TACH	3	2.0345E-07	Address/Control hold time

Key Encoder Timing					
DRAWN	TBL	SIZE	FSCM NO	DWG NO	REV 1.0
ISSUED	7/24/2008	SCALE 1 : 1		SHEET 1 OF 1	

Figure 4: Key Encoder Timing

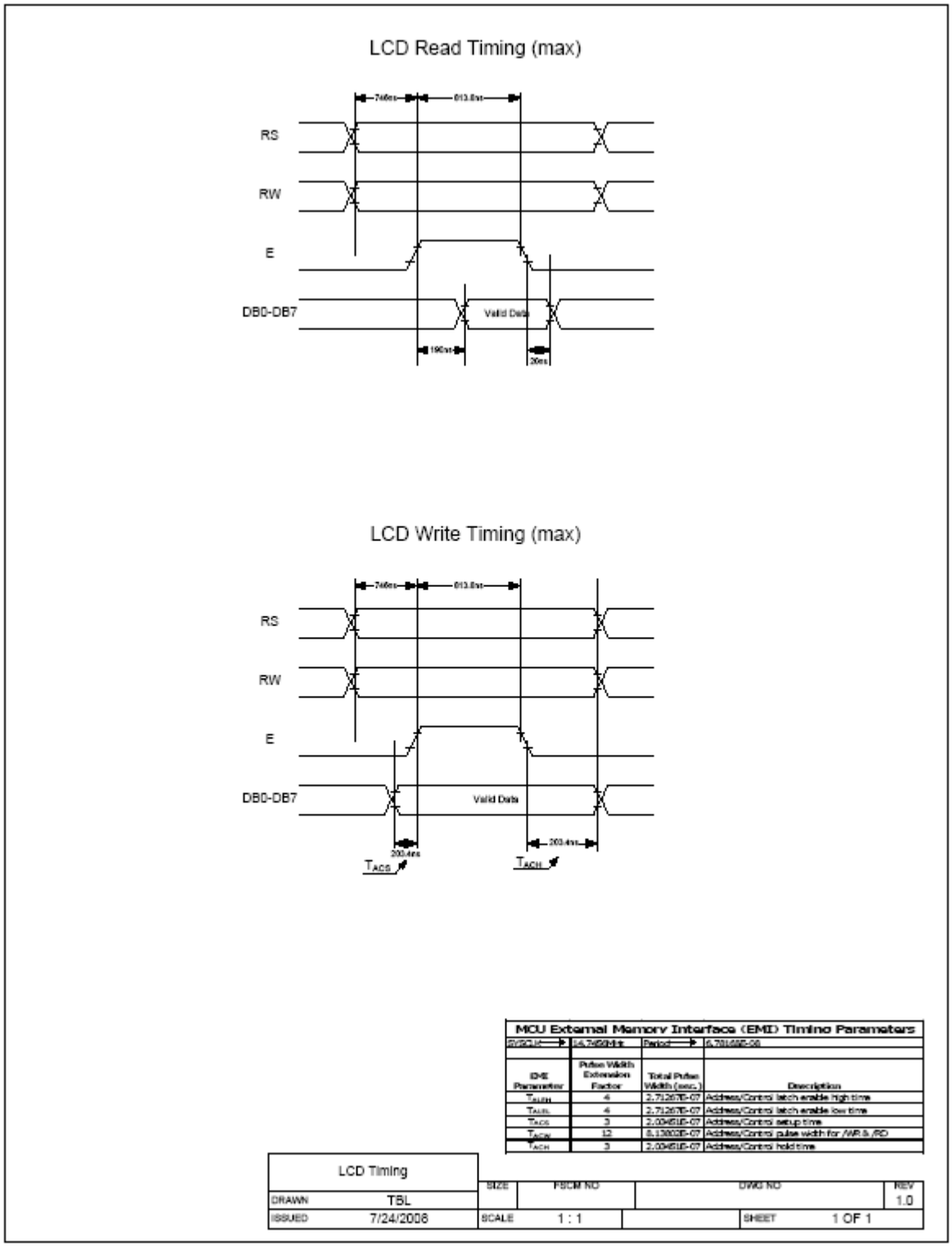


Figure 5: LCD Timing

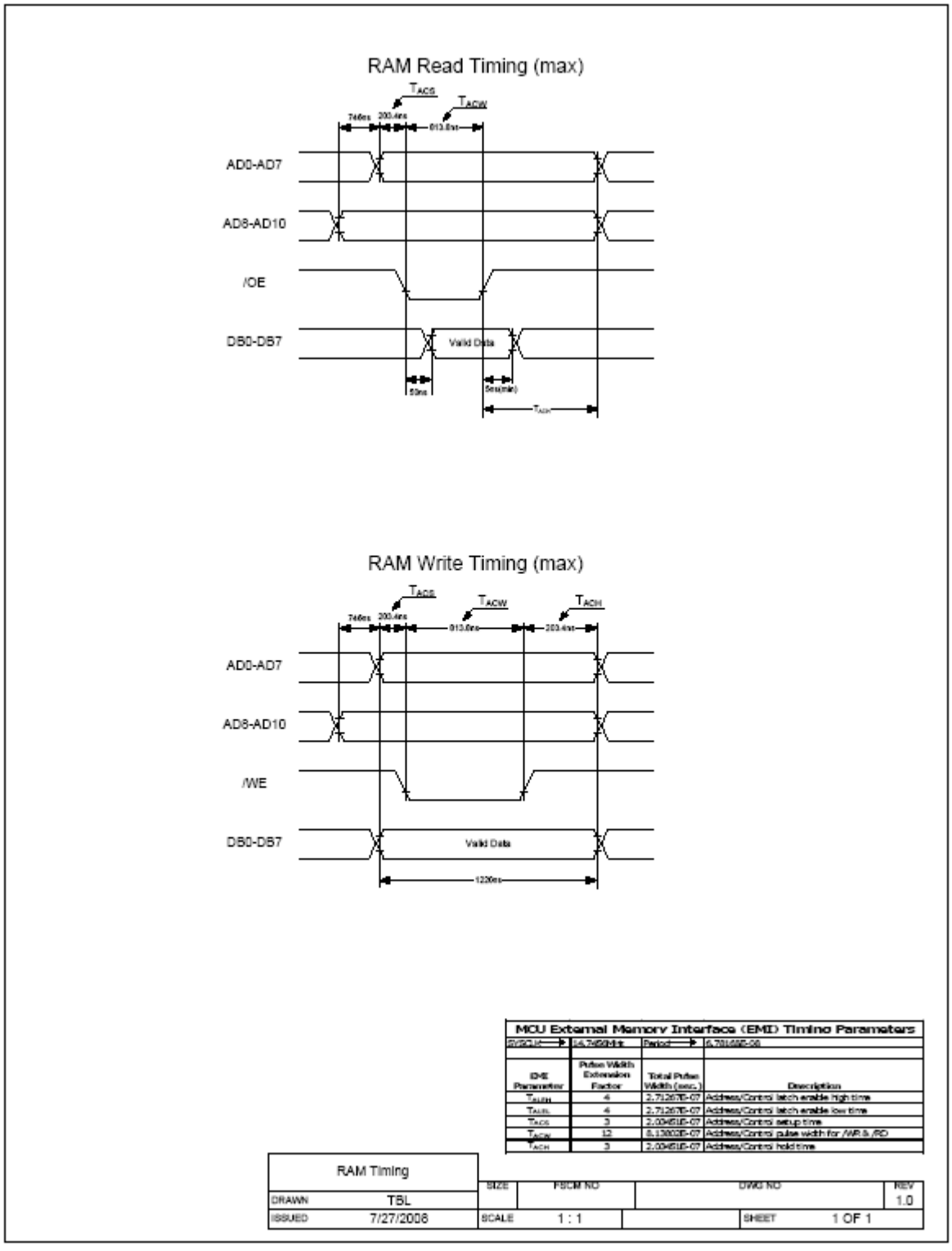


Figure 6: RAM Timing

Byte Address	Contents	Description
7E		
↑	Stack Pointer	The stack is init. At 30h
30		
2F	MSB	
2E	↑	PW storage from ext. RAM
2D		
2C	LSB	
2B	MSB	
2A	↑	User entered PW
29		
28	LSB	
27	MSB	
26	↑	Acceleration Setpoint
25	LSB	
24	0x##	Saves DPL during loops
23.1 (19h)	0 or 1	Limits a serial TX of Accel Alm to 1
23.0 (18h)	0 or 1	Alarm system armed
22	0 → 255	Current Acceleration from ADC
21	0 → 255	Current state for state machine
20.7 (07h)	0 or 1	Caps lock toggle bit
20.3 (03h)	0 or 1	Red function key
20.2 (02h)	0 or 1	Green function key
20.1 (01h)	0 or 1	Pink function key
20.0 (00h)	0 or 1	Blue function key
1F	Not Used	Not Used
1E		
1D		
1C		
1B		
1A		
19		
18		
17		
16		
15		
14		
13		
12		
11		
10		
F		
E		
D		
C		
B		
A		
9		
8		
7		
6		
5		
4	Not Used	Not Used
3	B register	Used in conjunction w/ACC to inc DPTR for lookup table
2	00h	Limits BS/Non-funct. key presses
1	21h	Pointer for current state
0	2Bh	Pointer for MSB of user ent. PW

Table 1: 8051 Memory Map

DPTR Address	Contents	Description
200C	00h	Null character
200B	LF	Line Feed
200A	CR	Carriage return
2009	"g"	ascii for letter 'g'
2008	0x##	Hundredth's place for Acceleration
2007	0x##	Tenth's place for Acceleration
2006	"."	ascii for character '.'
2005	0x##	One's place for Acceleration
2004	"+" or "-"	ascii for characters '+' or '-'
2003	MSB	PW storage to ext. RAM
2002	↑	
2001		
2000	LSB	

Table 2: RAM Memory Map

PROTEX-9000 PARTS LIST						
ITEM #	QUANITY	PART	MANUFACTURER	MANUFACTURER PN	SUPPLIER	COST
1	1	materials, wire wrap board, etc	NA	NA	MISC	$ 92.94
2	1	dev kit	SI Labs	C8051F020DK	SI Labs	$ 79.00
3	1	Micro Controller w/ break out	SI Labs	C8051F022	SI Labs	$ 60.00
4	1	CPLD w/socket	Lattice	lspm4A5	SPSU ARC	$ 30.00
5	1	Accelerometer w/ brk out	Analog Devices	ADXL213	SPARKSFUN	$ 27.00
6	1	LCD	Optrex	C-51847NFJ-SLW	Mouser	$ 26.00
7	5	SPDT toggle switches	Radio Shack	SWTS-2	Radio Shack	$ 17.45
8	3	SWPR-5 push button (mom.)	Radio Shack	SWPR-5	Radio Shack	$ 10.47
9	3	component carrier 24pin	Arles	24-600-10	Mouser	$ 10.38
10	1	key pad	Jameco-value pro	AK-1607-N-BBW-R	Jameco	$ 9.95
11	1	Siren board	Ramsey	MB1	Frys	$ 9.95
12	3	20 pin wire wrap socket	3M	20PINWWS	DII kit	$ 9.27
13	2	28 pin wire wrap socket (wide)	3M	28PINWWS	DII kit	$ 7.80
14	1	key pad encoder	Fairchild	MM74C922	PI	$ 6.15
15	1	opto isolator	Claire	OAA160	PI	$ 5.82
16	1	RS-232 line driver	Sipex	SP232AET	PI	$ 4.92
17	4	4.7KΩ+/-5% resistor	NA	NA	DII kit	$ 4.44
18	2	pin headers	3M	929834-07-36	Mouser	$ 4.26
19	1	1 MHz crystal oscill	ECS	ECS-2100AX-1.0MHZ	Mouser	$ 4.22
20	1	28 pin cap socket	Augat	DS2-328-1AR	PI	$ 4.19
21	1	24 pin cap socket	Augat	DS2-324-1AR	PI	$ 3.98
22	4	binding posts	Pomona	685-0254	Allied Electron.	$ 3.84
23	2	20 pin cap socket	Augat	DS2-320-1AR	PI	$ 3.83
24	1	ADC	National Semiconductor	ADC0808	PI	$ 3.60
25	1	18 pin cap socket	Augat	DS2-316-1AR	PI	$ 3.53
26	1	24 pin wire wrap socket	3M	24PINWWS	DII kit	$ 3.49
27	2	Red LED	King bright	T-1R	DII kit	$ 2.98
28	2	Green LED	King bright	T-1G	DII kit	$ 2.98
29	2	Bright yellow	King bright	604-WP7113SYC	Mouser	$ 2.92
30	1	SPLD	Lattice	GAL22V10	DII kit	$ 2.87
31	1	Transparent Latch	Fairchild	DM74LS373	PI	$ 2.83
32	1	Pink LED	Radio Shack	276-0019	Radio Shack	$ 2.83
33	1	18 pin wire wrap socket	3M	18PINWWS	DII kit	$ 2.48
34	2	8 pin wire wrap socket	3M	8PINWWS	DII kit	$ 2.28
35	9	470Ω +/- 5% resistor	NA	NA	DII kit	$ 2.22
36	1	Seven seg display	King bright	MAN72A	DII kit	$ 1.99
37	1	BCD to 7-SEG	Texas Instruments	74LS47	DII kit	$ 1.68
38	1	Blue LED	King bright	604-WP7113QBC/D	Mouser	$ 1.62
39	1	Yellow LED	King bright	T-1Y	DII kit	$ 1.49
40	1	14.7456 MHz crystal oscill	Fox	FOXLF147-20	Mouser	$ 1.41
41	1	SRAM 2016-12	Toshiba	TMM 2016AP-10	Jameco	$ 1.25
42	2	Timer	National Semiconductor	LM555	DII kit	$ 0.84
TOTAL						$ 481.15

Table 3: Parts List

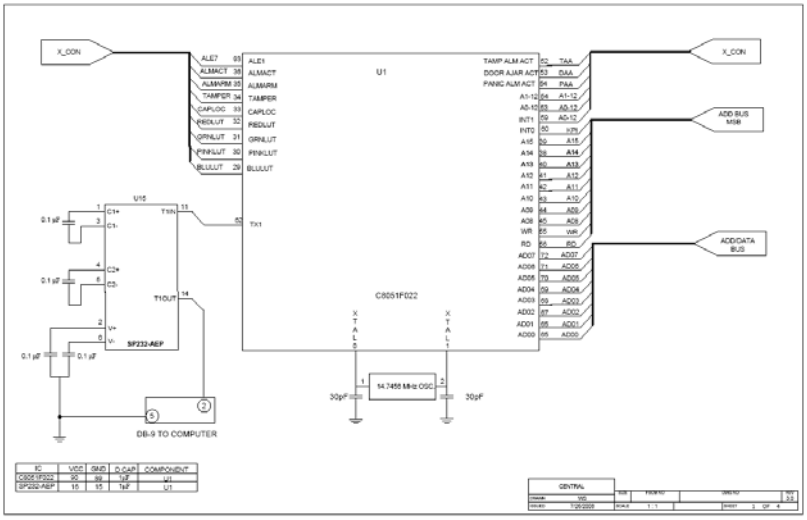


Figure 7: Schematic A

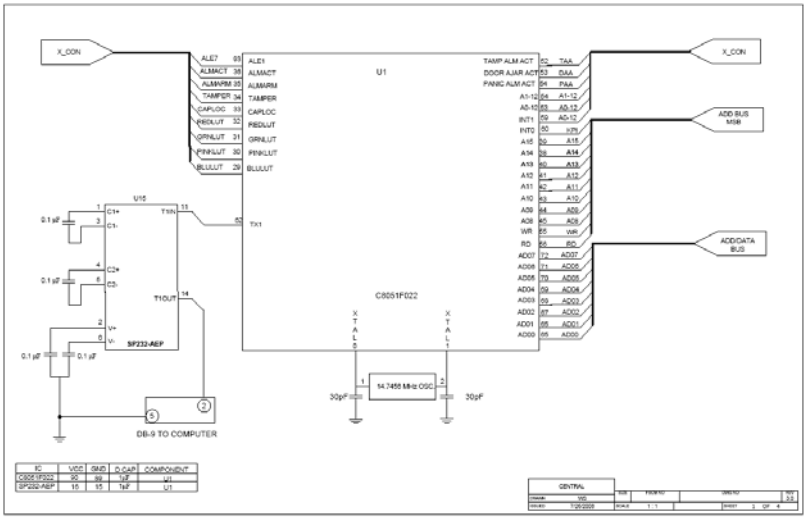


Figure 8: Schematic B

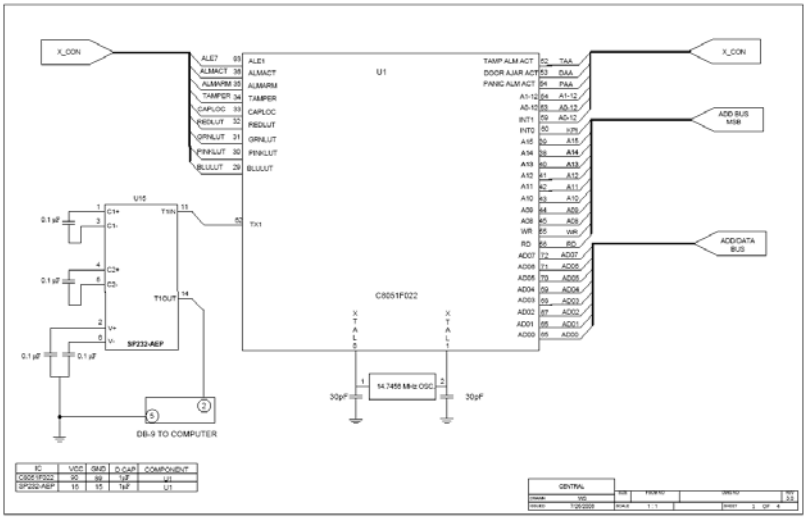


Figure 9: Schematic C

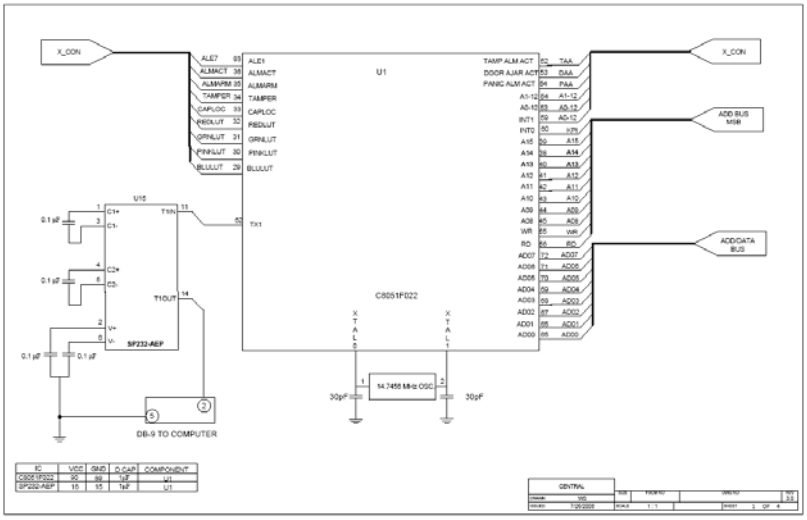


Figure 10: Schematic D

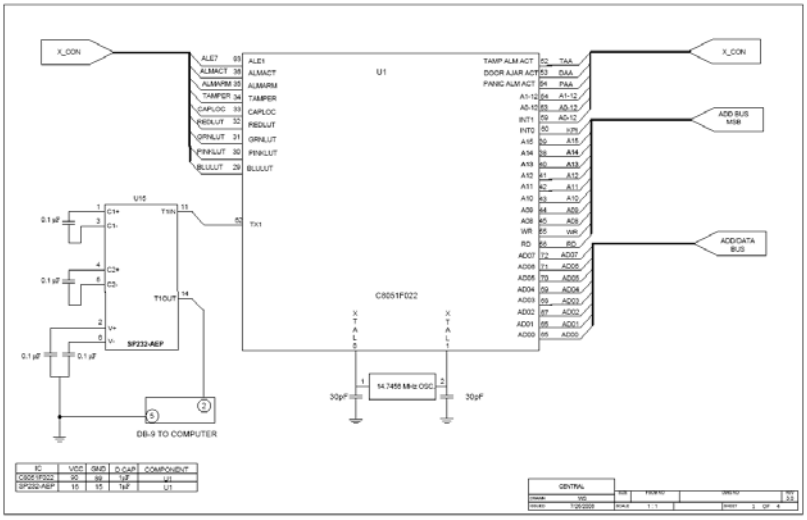


Figure 11: Screen Flow Diagram & Keyboard Layout

Appendix - Code