Embedded Control Handbook

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

September 1994 / DS00092C

Table of Contents

Table of Contents

9

10

11

CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL

CROSS REFERENCE GUIDE TO APPLICATION NOTES - ALPHABETICAL (continued)

CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL

CROSS REFERENCE GUIDE TO APPLICATION NOTES - NUMERICAL (continued)

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT

Please note that application software written for one family is easily converted to fit another.

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
24CXX serial EEPROM interface	AN515				
93CX6 serial EEPROM interface	AN530				
A/D conversion	AN513	AN546			
AC power line, interface to	AN521				
Addition, 16+16	AN526		AN544		
Addition, fixed point	AN526	AN526	AN544		
Addition, floating point	AN575	AN575	AN575		
Alarm clock implementation	AN529				
Analog to digital conversion	AN513				
Apple desktop bus interface		AN591			
Asynchronous serial port implementation in software	AN510				
Battery management IC				AN570	
Bank switching macros	AN586				
BCD addition	AN526	AN526	AN544		
BCD conversion routines	AN526	AN526	AN544		
BCD subtraction	AN526	AN526			
BCD to binary	AN526	AN526	AN544		
Binary to BCD	AN526	AN526			
Brown-out circuits	AN522	AN522			
Capacitance measurement	AN512				
Capture routines		AN594	AN545		
CCP module		AN594			
Chip select with VCC					AN535
Clock/calendar implementation	AN529	AN582			
Compare routines		AN594			
Comparison of low-end 8-bit microcontrollers	AN520				
Compatibility - 24C01A					AN517
Compatibility - 93C06					AN516
Data encryption			AN583		
Data Memory Bank Switching Macros	AN586				
D/A using PWM			AN538		
DDE, using PICMASTER's	AN584	AN584	AN584		
DES data encryption			AN583		
Digital voltmeter		AN557			
Digital filters			AN540		
Division, 16/16, unsigned	AN526	AN526	AN544		
Division, 16/16, signed	AN526	AN526	AN544		
Division, fixed point	AN526	AN526	AN544		

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Division, floating point	AN575	AN575	AN575		
DTMF generation			AN543		
EMI Results	AN577				
Endurance					AN537
Endurance predictive model					AN562
FFT			AN575		
Floating point addition	AN575		AN575		
Floating point multiplication	AN575		AN575		
Floating point routines	AN575		AN575		
Floating point subtraction	AN575		AN575		
Frequency Counter	AN592				
Frequency measurement			AN545		
Global interrupts	AN576	AN576	AN576		
I/O expansion			AN547		
I^2C implementation (for serial EE interface only)	AN515			AN571	
I^2C implementaion	AN541	AN554			
I^2C multi-master operation		AN578			
IIR filter			AN540		
Interfacing to 24C32/65					AN558
Interfacing to 24CXX					AN515
Interfacing to 24LCXX					AN567
Interfacing to 93CX6					AN530
Interfacing to 93LC56/66					AN560
Interface to serial A/D	AN531				
Interrupt, disabling global	AN576	AN576	AN576		
Interrupt, PORTB		AN566			
Interrupt, software	AN514				
Keypad interface	AN528/AN529	AN557			
LCD, direct driving with PIC16C5X	AN563				
LCD module interface	AN587	AN587			
LED display interfacing	AN529/590	AN557			
Long calls	AN581				
Low power clock		AN582			
Macros, page and bank switching	AN586				
Math routines	AN526/AN575	AN526/AN575	AN544/AN575		
Measuring capacitance	AN512				
Measuring frequency	AN592		AN545		
Measuring period			AN545		
Measuring pulse-width		AN594	AN545		

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Measuring resistance	AN512				
Measuring temperature	AN512				
Microwire interface	AN531				
Motor control	AN531		AN532		
Mouse, serial	AN519				
MTA11200, calibrating				AN570	
Multiplexing LED and keypad	AN529				
Multiplication, 8 x 8, unsigned	AN526		AN544		
Multiplication, 16 x 16, unsigned	AN526		AN544		
Multiplication, 16 x 16, signed	AN526		AN544		
Multiplication, fixed point	AN526		AN544		
Multiplication, floating point	AN575		AN575		
Oscillator design	AN588	AN588	AN588		
Page switching macros	AN586				
Parallel slave port		AN579			
Parity generation		AN555	AN547		
Period measurement			AN545		
PIC16C84 development programmer			AN589		
PID	AN531		AN532		
PLA implementation	AN511				
PLD replacement using PIC16CXX	AN511				
Pointing device				AN569	
Position control	AN531		AN532		
Positioner	AN531				
Power-on Reset	AN522				
Program Memory Page Switching Macros	AN586				
Pseudo-random number generation			AN544		
Pulse width measurement			AN545		
PWM, using		AN594	AN564		
PWM, choosing frequency		AN594	AN539		
PWM, choosing resolution		AN594	AN539		
PWM generation in software	AN531				
PWM routines		AN594	AN539		
PWM to analog output			AN538		
Quadrature encoder interface			AN532		
Random number generation, Gaussian			AN544		
Random number generation			AN544		
Real time clock (TMR1)		AN580/AN582			

CROSS REFERENCE CHART TO APPLICATION NOTES - BY SUBJECT (continued)

Subject	PIC16C5X	PIC16CXX	PIC17CXX	ASSP	Serial EEPROM
Real time clock implementation (from AC power line)	AN521				
Real time operating system (RTOS)	AN585	AN585	AN585		
Resistance measurement	AN512				
RS232 interface			AN547		
SEEPROM operation					AN536
Sequencer implementation	AN511				
Serial EEPROM interfacing	AN515				
Serial port (USART) simple routines		AN555	AN547		
Serial A/D interface	AN531				
Servo control			AN532		
Slave port		AN579			
Sine wave generation			AN543		
Software stack	AN527		AN534		
Square root	AN526	AN526	AN544		
Stack management in software	AN527		AN534		
State machine implementation	AN511				
Status save and restore	AN534				
Subtraction, 16 + 16	AN526/AN575	AN575	AN544/AN575		
Subtraction, fixed point					
Subtraction, floating point	AN526		AN544		
Table Read	AN556	AN556	AN548/AN556		
Temperature measurement	AN512				
Thermostat implementation	AN512				
Timer1 in asynchronous mode		AN580			
Tone generation			AN543		
Trajectory generation			AN532		
TrueGauge calibration				AN570	
UART, software implementation	AN510/AN593	AN555			
USART routines			AN547		
Velocity control			AN532		
Voltmeter implementation		AN157			
Wake up on key-stroke	AN528	AN552			
Write cycle optimization					AN559
Zero crossing detect	AN521				

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

Motivated by customer requirements....

"Microchip Technology Incorporated draws its impetus from the technology expectations of a large base of long-standing customers. Microchip responds quickly with technology to serve our customers' needs. Moreover, as a fully integrated IC manufacturer, Microchip deploys its panoply of resources to act timely and efficiently, and on a worldwide scale in providing: Technology Development, Design, Wafer Fabrication, Assembly and Test, Quality, Reliability and Customer Support.

...and powered by continuous improvement...

"Worldwide competition leaves no room for divergence or mediocrity. Microchip Technology, committed to focus on and continuously improve all the aspects of its business, takes pride in its unique corporate culture. To improve performance, our employees are encouraged to analyze their methods continually. Personal empowerment expands the capability of personal responsibility to continually serve our customers better.

...riding, leading and pushing the wave of technological change.

"Our industry's life-line is innovation. The fast pace of technological change is inherent in our industry. Microchip Technology has accelerated the rate of change of its technology and products to leadership in providing field-programmable space-sensitive embedded control solutions.

"Change is our ally. Driving and managing customer-focused change is our winning strategy."

*****insert signature here*****

Steve Sanghi
President & Chief Executive Officer

Microchip Technology Incorporated

MICROCHIP TECHNOLOGY INCORPORATED

Company Profile

HIGHLIGHTS

- Focused on providing high-performance, field-programmable embedded control solutions
- An experienced executive team focussed on innovation
- Offers RISC 8-bit field-programmable microcontrollers and supporting logic products
- Offers Serial and Parallel EEPROMs and EPROMs
- Offers complementary Application Specific Standard Products
- Fully integrated manufacturing
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement

BUSINESS SCOPE

Microchip Technology Incorporated manufactures and markets a variety of VLSI CMOS semiconductor components to support the embedded control market. In particular, the company specializes in highly integrated, RISC microcontrollers, application specific standard products and related non-volatile memory products to meet growing market requirements for high performance, yet economical embedded control capability in an increasing number of price-sensitive products. Microchip's products feature the industry's most economical OTP (One-Time-Programmable), ROM and EPROM capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

MARKET FOCUS

Microchip Technology targets selected markets where our advanced designs, progressive process technology and industry-leading operating speeds enable us to deliver decidedly superior performance. The company has positioned itself to maintain a dominant role as a supplier of high performance field-programmable microcontrollers and associated memory and logic products for embedded control applications.

(Chandler facility photo)

Chandler, Arizona:
Company headquarters near Phoenix, Arizona; executive offices, R & D and wafer fabrication occupy this 142,000-square-foot facility.

(Tempe facility photo)

Tempe, Arizona:
Wafer fabrication capacity is expanded dramatically with the addition of our new 170,000-square-foot facility.

Microchip Technology Incorporated

MICROCHIP

- Mission Statement -

Microchip Technology Incorporated is a leading supplier of field-programmable embedded control solutions by providing RISC microcontrollers and related non-volatile memory products. In order to contribute to the ongoing success of customers, shareholders and employees, our mission is to focus resources on high value, high quality products and to continuously improve all aspects of our business, providing a competitive return on investment.

- Guiding Values -

Customers Are Our Focus: We establish successful customer partnerships by exceeding customer expectations for products, services and attitude. We start by listening to our customers, earning our credibility by producing quality products, delivering comprehensive services and meeting commitments. We believe each employee must effectively serve their internal customers in order for Microchip's external customers to be properly served.

Quality Comes First: We will perform correctly the first time, maintain customer satisfaction and measure our quality against requirements. We practice effective and standardized improvement methods, such as statistical process control to anticipate problems and implement root cause solutions. We believe that when quality comes first, reduced costs follow.

Continuous Improvement Is Essential: We utilize the concept of "Vital Few" to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Employees Are Our Greatest Strength: We design jobs and provide opportunities promoting employee teamwork, productivity, creativity, pride in work, trust, integrity, fairness, involvement, development and empowerment. We base recognition, advancement and compensation on an employee's achievement of excellence in team and individual performance. We provide for employee health and welfare by offering competitive and comprehensive employee benefits.

Products And Technology Are Our Foundation: We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system and software technologies to provide timely, innovative, reliable and cost effective products to support current and future market opportunities.

Total Cycle Times Are Optimized: We focus resources to optimize cycle times to our internal and external customers by empowering employees to achieve efficient cycle times in their area of responsibility. We believe that cycle time reduction is achieved by streamlining processes through the systematic removal of barriers to productivity.

Safety Is Never Compromised: We place our concern for safety of our employees and community at the forefront of our decisions, policies and actions. Each employee is responsible for safety.

Profits And Growth Provide For Everything We Do: We strive to generate and maintain competitive rates of company profits and growth as they allow continued investment for the future, enhanced employee opportunity and represent the overall success of Microchip.

Communication Is Vital: We encourage appropriate, honest, constructive, and ongoing communication in company, customer and community relationships to resolve issues, exchange information and share knowledge.

Suppliers, Representatives, And Distributors Are Our Partners: We strive to maintain professional and mutually beneficial partnerships with suppliers, representatives, and distributors who are an integral link in the achievement of our mission and guiding values.

Professional Ethics Are Practiced: We manage our business and treat customers, employees, shareholders, investors, suppliers, distributors, representatives, community and government in a manner that exemplifies our honesty, ethics and integrity. We recognize our responsibility to the community and are proud to serve as an equal opportunity employer.

FULLY INTEGRATED MANUFACTURING

Microchip delivers fast turnaround through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced statistical process control (SPC) and a continuous improvement culture has brought forth tight product consistency levels and high yields which enable Microchip to compete successfully in world markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.

A GLOBAL NETWORK OF PLANTS AND FACILITIES

Microchip is a global competitor providing local service to the world's technology centers. The Company's focal point is the design and technology advancement facility in Chandler, Arizona. Product and technology development is here, along with front-end wafer fabrication and electrical probing.

In late 1993, Microchip purchased a second wafer fabrication facility in Tempe, Arizona–thirteen miles from its existing Chandler, Arizona, operation. The additional 170,000 square foot facility will be equipped with process equipment for use in meeting future production volumes beyond those which could be efficiently produced in Microchip's single existing wafer facility. Initial production from the new Tempe facility is anticipated to begin by late 1994.

Microchip's assembly and test facility in Kaohsiung, Taiwan houses the technology, modern assembly methods and test equipment necessary for plastic and ceramic packaging. Microchip also assembles and tests products in facilities owned and operated by Alphatek in Bangkok, Thailand.

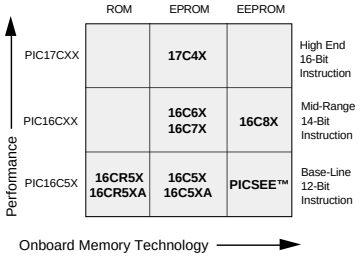
Sales and application offices are located in key cities throughout the Americas, Pacific Rim and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical support, purchasing information and failure analysis.

A PRODUCT FAMILY OF SHARED STRENGTHS

Microchip's product focus is CMOS field-programmable microcontrollers, non-volatile memories and peripherals, and application specific standard products (ASSP). These product lines include PIC16/17 microcontrollers, Serial and Parallel EEPROMs, high-speed EPROMs, and peripherals in a broad range of product densities, speeds and packages.

MICROCONTROLLERS

PIC16/17 microcontrollers from Microchip combine high performance, low cost and small package size. They offer the best price/performance ratio in the industry. Large numbers of these devices are used in automotive and cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.



CMOS PIC16/17 Microcontroller Families

The widely-accepted CMOS PIC16C5X, PIC16CXX and PIC17CXX families are the industry's only 8-bit microcontrollers using a high-speed RISC architecture. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency. The CMOS PIC16C5X family is in high-volume production, shipping in the range of one million units per week, and has achieved more than twenty-five thousand design wins worldwide.

The PIC16CXX mid-range family is rapidly gaining acceptance with three of its members introduced: PIC16C71, PIC16C84 and PIC16C64.

The PIC17CXX family offers the world's fastest execution performance of any 8-bit microcontroller family. The PIC17CXX family extends the PIC16/17 microcontroller's high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. The first member of the family, the PIC17C42, includes a powerful array of intelligent and precise on-chip peripheral features that are ideally suited for many demanding real-time embedded control applications including motor control, process control, security, automotive and medical

applications. In addition, the PIC17C42 can function either as a stand-alone microcontroller or can execute instructions from up to 64K words of external program memory. The PIC17C42 features comprehensive timer/counter resources and I/O handling capabilities to address the requirements of complex embedded control applications.

Current CMOS PIC16/17 microcontroller product families include advanced features such as sophisticated timers, embedded A/D, extended instruction/data memory, inter-processor communication (I^2C™ bus, SPI and USARTs) and ROM, RAM, EPROM and EEPROM memories.

Both PIC16CXX and PIC17CXX families are supported by user-friendly development systems including assemblers, software simulators, programmers and emulators.

DEVELOPMENT SYSTEMS

The PICMASTER™ is an advanced real-time in-circuit emulator system running under Windows™ environment. The PICMASTER is a Microchip-designed universal emulator for both PIC16CXX and PIC17CXX families. The PRO MATE™ is an advanced full-featured programmer. PICSTART™ is a low-cost development kit which includes an assembler, simulator and development programmer.

Microchip's Serial EEPROM Designer's Kit includes everything necessary to develop a reliable Serial EEPROM-based design.

SOFTWARE SUPPORT

Both PIC16/17 microcontroller families are supported by assemblers, linker/loaders, libraries and a source-level debugger. The PIC16C5X and PIC16CXX families are also supported by a software simulator.

A full-featured C Compiler and Fuzzy Logic support are also available for all three families.

Customers can obtain on-line updates on Microchip Development Systems and Support Software via the Bulletin Board System (BBS). Please refer to the Microchip BBS product brief in Section 9 for specific access information.

SERIAL EEPROMS

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages. The company has developed the world's first 64K Smart Serial™ EEPROM which currently offers four times the speed, four times the memory and four times the features of any competitive 2-wire Serial EEPROM. Device densities range from 256K bits up to 64K bits. In addition to 5 volt-only operation, Microchip offers Serial EEPROMs that read and write down to 2.5, 2 or 1.8 volts. I^2C™, Microwire™ and 4-wire bus interface protocols are standard. Devices come in three standard operating temperature ranges; commercial, industrial and automotive. Small footprint packages include: 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths and 14-lead SOIC. Other key features of the Serial EEPROM product line include: electrostatic discharge (ESD) protection greater than 4K volts and endurance of 100K cycles minimum and one million typical.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide, including consumer, automotive, industrial, computer and communications. To date, more than 100 million units have been produced. Microchip is continuing to develop additional unique Serial EEPROMs.

PARALLEL EEPROMS

The CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 write and erase cycles typically. Data retention is more than 10 years. Fast write times are less than 200 μsec. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from –40°C to +80°C. Microchip's expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers' needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, pattern recognition and telecommunications.

EPROMS

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. High Speed EPROMs have access times as low as 55 nanoseconds. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in Surface Mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount one-time-programmable (OTP) EPROM market where Microchip is the #1 supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

APPLICATION SPECIFIC STANDARD PRODUCTS (ASSP)

Microchip's Application Specific Standard Product (ASSP) Division provides value-added embedded control solutions by combining PIC16/17 microcontroller architecture with innovative software, silicon and assembly technology. These products incorporate technology that offers a complete solution that is both unique to the customer and standard in manufacture to Microchip. In addition, Microchip ASSPs reduce or remove the barriers for customers to use Microchip solutions in their products through the use of software embedded in secure OTP- or ROM-based microcontrollers. The family is packaged to provide the highest integration to the customer at the best overall system cost.

The MTA11XXX family is the most accurate and most integrated battery management and charging solution available today. The family incorporates Microchip/SPAN patented *TrueGauge*™ technology which digitally integrates battery charge and discharge current to provide an accurate (>97% typical) state of charge indication. The family operates with NiCd, Pb acid and NiMH battery packs from 3 Vdc to 30 Vdc. These products are ideal for portable PC, cellular phone and portable consumer product applications.

Ease of use, low voltage and low cost make the MTA41XXX mouse and trackball MCU firmware solutions ideal for implementing new designs for both PCs and Apple® computers. The products in the MTA41XXX family are 18-lead, low-power CMOS microcontroller ICs combined with application-specific software. By adding a few external components, the user can easily realize a complete mouse or trackball system.

The MTA8XXXX PICSEE™ family of cost-effective system solutions integrate PIC16/17 microcontrollers with EEPROM technology. These PICSEE devices are ideally suited for automotive security, keyless entry, remote control, data acquisition and telecommunication applications. The combined product assembly techniques provide the user the highest performance solution in a compact and cost-effective package.

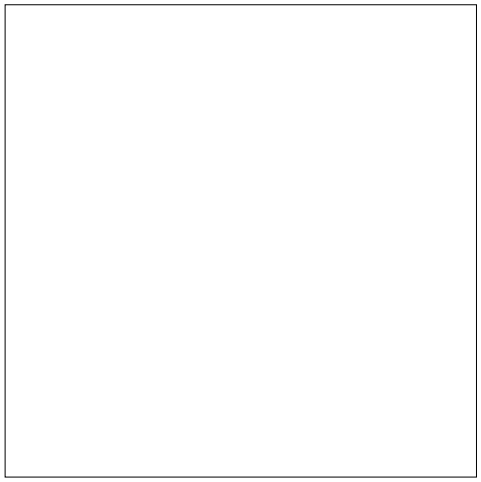
Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

OTHER MICROCHIP PRODUCTS

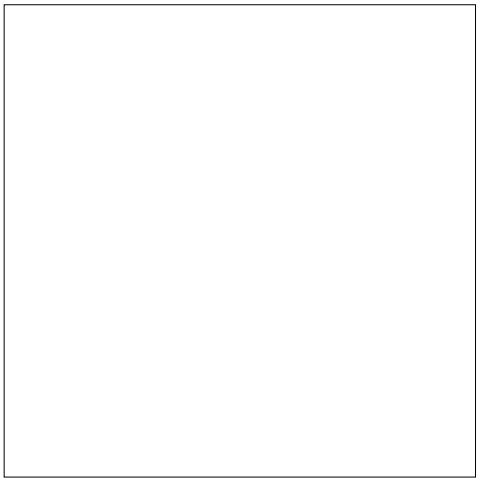
Other Microchip products, such as Liquid Crystal Display (LCD) drivers, are mature products with proven track record and a large, repeat customer base.

Microchip Technology Incorporated

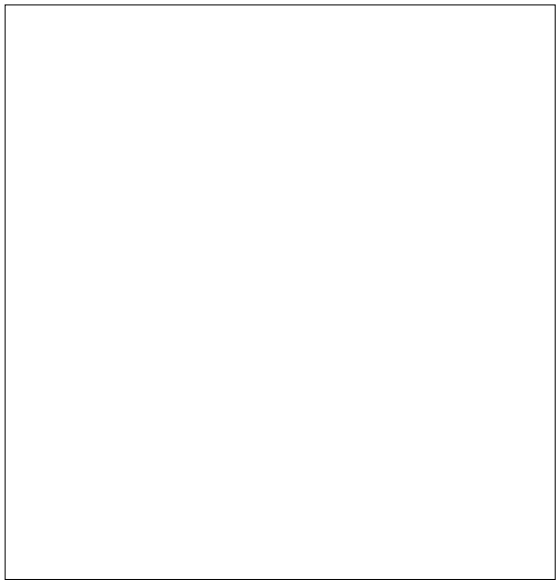
CHANDLER, AZ FACILITY

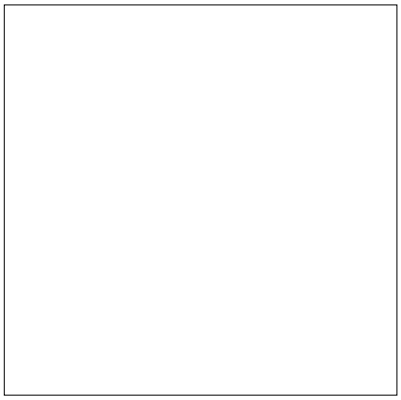


Chandler Wafer Fabrication: Diffusion Area



Chandler Wafer Fab: Sub-micron Alignment Area





The Microchip Kaohsiung plant's excellent track record and continuing efforts to achieve higher levels of quality and technological advancement has resulted in superior yields and fast turnaround.

TAIWAN FACILITY

Microchip's assembly and test operation in Kaohsiung, Taiwan received the prestigious Ishikawa Award for assembly and testing excellence.

Microchip Technology Incorporated

A HISTORY OF INNOVATION

Microchip has a long history of innovation in the semiconductor industry. For more than a quarter century, Microchip and its former parent company have been developers of leading-edge, cost-effective logic and memory products.

Microchip is credited with a number of firsts: The Metal-Oxide-Silicon (MOS) Integrated Circuit, DRAM, Serial EEPROM, Reduced Instruction Set Computer (RISC) microcontroller product family, UART, CMOS 64K EEPROM, and CMOS single chip DSP are all innovations that were originally developed and introduced by Microchip engineers.

FUTURE PRODUCTS AND TECHNOLOGY

New process technology is constantly being developed for microcontroller, ASSP, EEPROM and high-speed EPROM products. Advanced process technology modules are being developed that will be integrated into present product lines to continue to achieve a range of compatible processes. Current production technology utilizes dimensions down to 0.9 microns.

Microchip's research and development activities, include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools on line. Cycle times for new technology development are continuously reduced by using in-house mask making, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based microcontroller, ASSP and CMOS EEPROM and EPROM products. Objective specifications for new products are developed by listening to our customers and by close cooperation with our many customer-partners worldwide.

QUALITY WITHOUT COMPROMISE

Product reliability is designed into Microchip products at the outset. Wide design margins are established to guarantee that every product can be produced easily, error-free and within the tolerances of the manufacturing process.

All quality assurance tests are tighter than customer specifications. Products are tested at least two machine tolerances tighter than those specified by the customer.

Every new product is qualified under accelerated stress testing. Test samples encompass the full range of processed tolerances at each step. Data sheets detailing these processes enable customers to reach accurate decisions based on known quantitative values.

To determine whether a process is within normal manufacturing variation, industry-leading statistical control techniques are put to work at each process step. In-process controls are performed by operators in the wafer fabrication division and immediate corrective action is taken if they deem a process is out of tight control limits. Products are also sampled weekly through a variety of carefully monitored stress and accelerated life tests.

Microchip's documentation control program assures the correct document is always available at the point of use. Active documents are serialized and stamped to eliminate the possibility of performing a job from obsolete or incorrect instructions.

Individuals in all departments continuously analyze the methods employed at their positions and formulate plans to improve performance. In all areas of our business, everyone is expected to make continuous improvement.

Microchip Technology Incorporated

A QUALITY AND RELIABILITY ALLIANCE WITH CUSTOMERS

Microchip works together with customers to establish mutual programs to improve the performance of our products in their systems. We go beyond the incoming inspection level and specification by extending our quality and reliability support to the point where the customer ships the system. Microchip's quality programs ensure that our products can be used with such impunity, a customer can implement improvement programs based on Microchip as your leading supplier.

SECTION 1
INTRODUCTION TO EMBEDDED SOLUTIONS

Embedded Control Handbook

Introduction to *The Embedded Control Solutions Company*™

Microchip Technology's mission is to offer leadership semiconductor products for embedded control system applications. To do this we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: 8-bit PIC16/17 microcontrollers and Serial EEPROMS. These product lines provide the solutions to many of the problems facing designers of embedded control systems.

We publish this *Embedded Control Handbook* to assist our customers, existing and new, in their efforts to design and produce state-of-the-art embedded control systems.

EMBEDDED CONTROL OVERVIEW

Unlike "processor" applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are buried inside the application. The user of the product is only concerned with the very top-level user interface (such as keypads, displays and high-level commands). Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in virtually all market segments: consumer, commercial, PC peripherals, automotive, telecommunications (including fast-emerging personal telecom products) and industrial. Most often embedded control products must meet special requirements: cost-effectiveness, low power, small footprint and a high level of system integration.

Typically, most embedded control systems are designed around a microcontroller which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems also require serial EEPROM memories, display drivers, keypads, small displays, etc.

Microchip Technology has established itself as a leading supplier of field-programmable embedded control solutions. The combination of high-performance microcontrollers from both the PIC17CXX and PIC16CXX families, along with industry leading nonvolatile memory products provides the basis for this leadership. Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

PIC16/17 MICROCONTROLLER OVERVIEW AND ROADMAP

Microchip offers three families of 8-bit microcontrollers to best fit your needs:

- PIC16C5X: Base-Line 8-bit Family
- PIC16CXX: Mid-Range 8-bit Family
- PIC17CXX: High-End 8-bit Family

All families offer One-Time-Programmable, low-voltage and low-power options, as well as various packaging options. Selected members are available in ROM version.

PIC16C5X: BASE-LINE FAMILY

PIC16C5X is the well established base-line family offering the highest cost efficiency. This PIC16C5X products have a 12-bit wide instruction set and are currently offered in 18-, 20- or 28-pin packages. In SOIC and SSOP packaging options, these are the smallest footprint controllers. Low-voltage operation down to 2.0V makes this family ideal for battery operated applications.

PIC16CXX: MID-RANGE FAMILY

PIC16CXX mid-range family offers a wide-range of options, from 18-pin to 44-pin packages as well as low to high level of peripherals integration. This family has a 14-bit wide instruction set. PIC16C71 is a compact microcontroller in a 18-pin package with an 8-bit A/D converter. PIC16C84 offers on-chip EEPROM program and data memory. PIC16C64 is a full-featured 40-pin device with 128 bytes of RAM, three timer/counters, capture, compare, PWM and SPI/I²C™ serial communication port. PIC16C74 is a highly integrated 40-pin device with all the features of the PIC16C64, plus an 8-channel A/D converter and a full-featured USART.

FIGURE 1 - PIC16/17 MICROCONTROLLER MATRIX

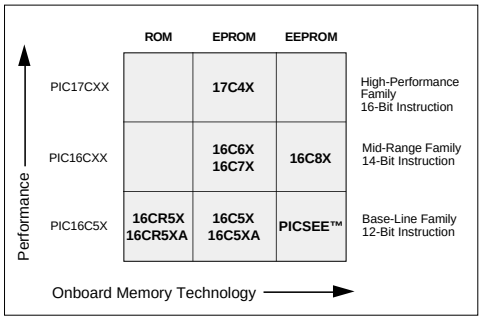


TABLE 1 - FEATURES OVERVIEW OF THE PIC16/17 MICROCONTROLLERS

		Clock	Memory							Peripherals								Features‡		
		Maximum Frequency of Operation (MHz)	Program Memory (words)		ROM	EPROM	EEPROM	RAM Data Memory (bytes)	EEPROM Data Memory (bytes)	Timer Module(s) †	Capture/Compare/PWM Module	Serial Port (s) (SPI/I²C, SCI)	Parallel Slave Port	Analog to Digital Converter (8-bit)	External Interrupts	Interrupt Sources	I/O*	Voltage Range (Volts)	Number of Instructions	Packages
	PIC17C42§	25	2K	—	—	128	—	TMR0, TMR1, TMR2,TMR3	—	SCI	—	—	Yes	11	33	4.5 - 5.5	55	40-pin DIP, 44-pin PLCC 44 pin QFP
High-end	PIC16C61	16	1K	—	—	36	—	TMR0	—	—	—	—	Yes	3	13	3.0 - 6.0	35	18-pin DIP, 18-pin SOIC
Mid-range	PIC16C64	20	2K	—	—	128	—	TMR0, TMR1, TMR2	1	SPI/I²C™	Yes	—	—	8	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C71	16	1K	—	—	36	—	TMR0	—	—	—	4 ch	Yes	4	13	3.0 - 6.0	35	18-pin DIP, 18-pin SOIC
	PIC16C74	20	4K	—	—	192	—	TMR0, TMR1, TMR2	2	SPI/I²C, SCI	Yes	8 ch	Yes	12	33	2.5 - 6.0	35	40-pin DIP, 44-pin PLCC 44 pin QFP
	PIC16C84	10	—	—	1K	36	64	TMR0	—	—	—	—	Yes	4	13	2.0 - 6.0	35	18 pin DIP, 18 pin SOIC
Base-Line	PIC16C54	20	512	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C54A	20	512	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16CR54	20	—	512	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C55	20	1K	—	—	25	—	RTCC	—	—	—	—	—	—	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C56	20	1K	—	—	25	—	RTCC	—	—	—	—	—	—	12	2.5 - 6.25	33	18-pin DIP, 18-pin SOIC 20-pin SSOP
	PIC16C57	20	2K	—	—	72	—	RTCC	—	—	—	—	—	—	20	2.5 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16CR57A	20	—	2K	—	72	—	RTCC	—	—	—	—	—	—	20	2.0 - 6.25	33	28-pin DIP, 28-pin SOIC 28-pin SSOP
	PIC16C58A	20	2K	—	—	73	—	RTCC	—	—	—	—	—	—	12	2.5	33	18-pin DIP, 18-pin SOIC

† PIC17C42 can concatenate Timer1 and Timer2 to form a 16-bit Timer. Timer0 is 16-bit with 8-bit prescaler.
‡ All PIC16/17 Family devices have Power-on Reset, fuse selectable Watchdog Timer and fuse selectable code protect.
§ The PIC17C42 can also operate in microprocessor or external microcontroller mode.
* All PIC16/17 devices offer 20-25mA source / sink current per pin.

PIC17CXX: HIGH-END FAMILY

The PIC17CXX high-end family currently is comprised of one member, the PIC17C42. Additional members with larger on-chip memories are planned. The PIC17CXX products have a 16-bit wide instruction set. At 25MHz clock rate (160ns instruction cycle). The PIC17C42 offers the highest level of computation power. The PIC17C42 is also highly integrated with peripheral resources.

FIGURE 2 - PIC16/17 MICROCONTROLLER MIGRATION PATH

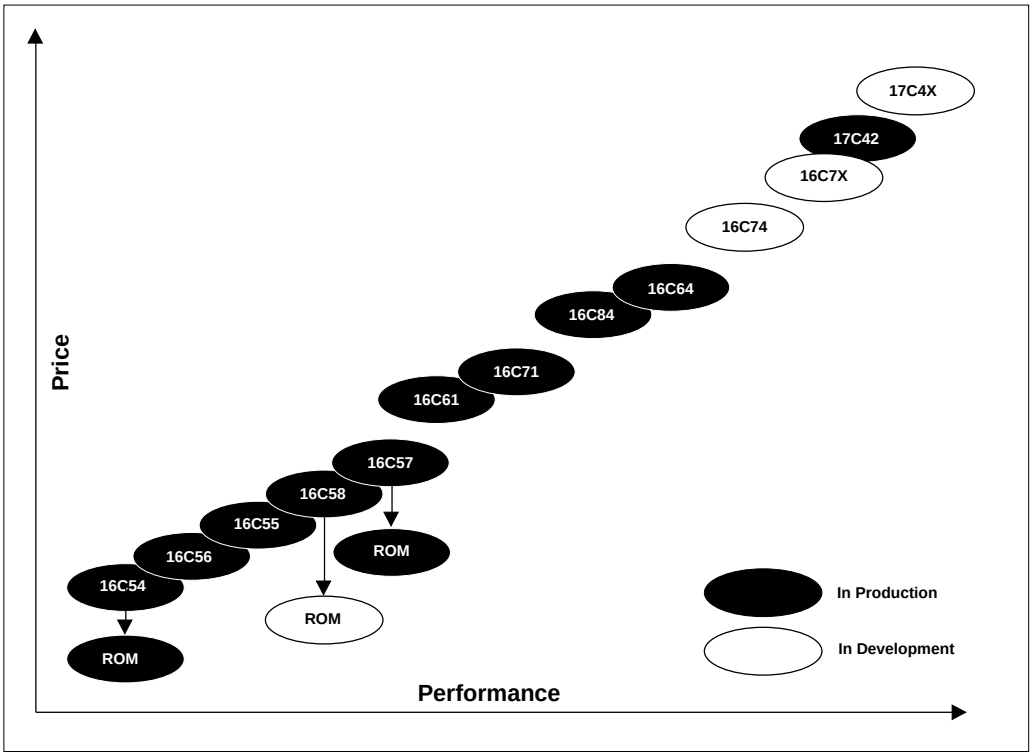


FIGURE 3 - PIC16/17 SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC16CXX	PIC16CXX	PIC17CXX
Assembler	MPASM	✔	✔	✔
Software Simulator	MPSIM	✔	✔	**
C Compiler*	MP-C	✔	✔	✔
Universal Programmer	PRO MATE™	✔	✔	✔
Universal In-Circuit Emulator	PICMASTER™	✔	✔	✔
Fuzzy Logic Development Tool	fuzzyTECH®-MP	✔	✔	✔

* Available from Bytecraft LTD in Canada.
** In development.
For an overview of development tools, see Section 9.

Introduction

PIC16/17 NAMING CONVENTION

The PIC16/17 architecture offers users a wide range of cost/performance options of any 8-bit microcontroller family. In order to identify the families, the following naming conventions have been applied to the PIC16/17 microcontrollers.

(@ 20 MHz)

TABLE 2 - PIC16/17 NAMING CONVENTION

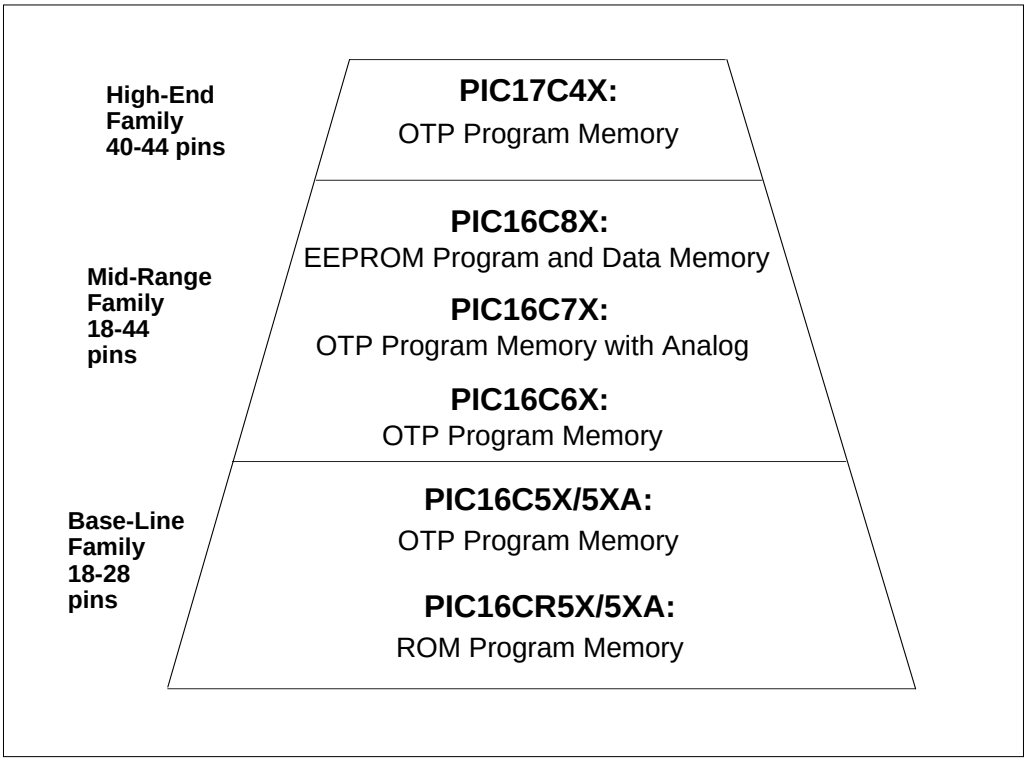
Family		Architectual Features	Name	Technology	Products
PIC16C5X	Base-Line 8-bit Microcontroller Family	• 12-bit wide instruction set • DC - 20MHz clock speed • 200ns instruction cycle	PIC16C5X	OTP program memory, digital only	PIC16C54 PIC16C54A PIC16C55 PIC16C56 PIC16C57 PIC16C58A
			PIC16CR5X	ROM program memory, digital only	PIC16CR54 PIC16CR57A
PIC16CXX	Mid-Range 8-bit Microcontroller Family	• 14-bit wide instruction set • Internal / external interrupts • DC - 20 MHz clock speed(3) • 200ns instruction cycle (@ 20 MHz)	PIC16C6X	OTP program memory, digital	PIC16C61 PIC16C64
			PIC16CR6X	ROM program memory, digital only	Planned
			PIC16C7X	OTP program memory, with analog functions (e.g. A/D)	PIC16C71 PIC16C74
			PIC16C8X	EEPROM program and data memory	PIC16C84
PIC17CXX	High-End 8-bit Microcontroller Family	• 16-bit wide instruction set • Internal / external interrupts • DC - 25 MHz clock speed • 160ns instruction cycle	PIC17C4X	OTP program memory, digital only	PIC17C42
			PIC17CR4X	ROM program memory, digital only	Planned

Notes: 1. "A" designates a more advanced process technology, generally offering customers the benefits of lower power, higher speed, etc. (example: PIC16C54, PIC16C54A).

2. The numbering system within each family is not necessarily significant.

3. The maximum clock speed for some devices is less than 20 MHz.

FIGURE 4 - PIC16/17 8-BIT MCU FAMILY



Introduction

SERIAL EEPROM OVERVIEW

Serial EEPROMs from Microchip come in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges, and space saving packages.

Densities:

Currently range from 1K to 64K with higher density devices in development.

Bus Interface Protocols:

All major protocols are covered: 2-wire, 3- wire and 4-wire.

Operating Voltages :

In addition to standard 5V devices there are two low voltage families. The “LC” devices operate down to 2.5V, while the breakthrough “AA” family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCad battery powered applications.

Temperature Ranges:

Like all Microchip devices, Serial EEPROMs are offered in Commercial (0˚C to 70˚C), Industrial (-40˚C to 85˚C) and Automotive (-40˚C to 125˚C) operating temperature ranges.

Packages:

The focus is on small packages. Most devices are available in 8-pin PDIP or 8-pin SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Endurance is specified at 1M Erase/Write typical and 100K cycles minimum with a data retention specification greater than 40 years. ESD protection is guaranteed up to 4K volts.

OTP EPROM OVERVIEW

Microchip also provides its customers with a number of CMOS OTP EPROMs. Densities offered include: 64K, 128K, 256K and 512K , all in a X8 organization. Our high speed 256K device also comes in a X16 organization. Access times range from a high-performance 55ns to a practical 200ns or greater. Low-voltage devices, capable of operating at 3V, are available at the 256K and 512K density levels. Surface mounted packages such as TSOP, PLCC and SOIC as well as the more traditional DIP packages are offered. All EPROMs are available in Commercial, Industrial and Automotive temperature ranges.

A full listing of the Serial EEPROM and EPROM product offerings are shown in Table 3. See the individual Microchip data sheet /data book for detailed information.

THE ADVANTAGES OF ONE-TIME-PROGRAMMABLE

In keeping with Microchip's goal of providing the embedded control system designer with the best tools available, Microchip has developed the industry's most economical OTP technology. Microchip offers a wide variety of OTP EPROM products. Similarly, by basing the PIC16/17 microcontrollers around an EPROM program memory capability, all of the advantages of OTP, both in development and production, have been made economically available to the systems manufacturer. The benefits of OTP technology include:

- Lower costs and shorter lead times
- Reduced time-to-market
- In-circuit programming capability
- Code protection via security fuse
- Reduction of inventory requirements at system manufacturing site
- Quick correction of bugs detected in manufacturing
- Quick product feature changes in response to customer requests
- Reduces wasted inventory

TABLE 3 - CMOS SERIAL EEPROMS PRODUCT SELECTION GUIDE

FAMILY	Device	Density Organization	Max. Clock Frequency	Endurance (cycles typ.)	Temp. Range	# Pins	Package Types	Operating Voltage
2-Wire (I^2C™) Bus Protocol								
STANDARD 2-WIRE FAMILY	24C01A	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C02A	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SN, SM	5.0V
	24C04A	4K bits (512 X 8)	100 kHz	1M 100 K	C, I	8 14	P, J SL	5.0V
	85C72	1K bits (128 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C82	2K bits (256 X 8)	100 kHz	1M 100 K	C, I E	8	P, J, SM	5.0V
	85C92	4K bits (512 X 8)	100 kHz	1M 100 K	C, I E	8 14	P, J, SM, SL	5.0V
LOW-VOLTAGE 2-WIRE FAMILY	24LC01B	1K bits (128 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC02B	2K bits (256 X 8)	400 kHz	1M	C, I	8	P, SN, SM	2.5V-5.5V
	24LC04B	4K bits (512 X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC08B	8K bits (1K X 8)	400 kHz	1M	C, I	8 14	P, SN, SM SL	2.5V-5.5V
	24LC16B	16K bits (2K X 8)	400 kHz	1M	C, I	8 14	P, SN SL	2.5V-5.5V
AA LOW-VOLTAGE 2-WIRE FAMILY	24AA01	1K bits (128 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA02	2K bits (256 X 8)	100 kHz	1M	C	8	P, SN, SM	1.8V-5.5V
	24AA04	4K bits (512 X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA08	8K bits (1K X 8)	100 kHz	1M	C	8 14	P, SN, SM SL	1.8V-5.5V
	24AA16	16K bits (2K X 8)	100 kHz	1M	C	8 14	P, SN SL	1.8V-5.5V
SMART SERIAL 2-WIRE FAMILY	24C32	32K bits (4K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	4.5V-5.5V
	24LC32	32K bits (4K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	2.5V-6.0V
	24C65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	4.5V-5.5V
	24LC65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	2.5V-6.0V
	24AA65	64K bits (8K X 8)	400 kHz	1M*	C, I	8	P, SM, SL	1.8V-6.0V
3-Wire (Microwire™)/4-Wire Bus Protocol								
STANDARD 3-WIRE FAMILY	93C06	256 bits (16 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C46	1K bits (64 X 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V
	93C56	2K bits (256 X 8) or (128 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
	93C66	4K bits (512 X 8) or (256 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.0V-5.5V
LOW-VOLTAGE 3-WIRE FAMILY	93LC46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C, I	8 14	P, SN, SM SL	2.0V-6.0V
	93LC46B	1K bits (64 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC56B	2K bits (128 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
	93LC66B	4K bits (256 x 16)	2 MHz	1M	C, I	8	P, SN, SM	2.0V-6.0V
AA LOW-VOLTAGE 3-WIRE FAMILY	93AA46	1K bits (128 X 8) or (64 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA56	2K bits (256 X 8) or (128 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
	93AA66	4K bits (512 X 8) or (256 x 16)	2 MHz	1M	C	8	P, SN, SM	1.8V-5.5V
4-WIRE	59C11	1K bits (128 X 8) or (64 x 16)	1 MHz	1M 100 K	C, I E	8	P, J, SN, SM	4.5V-5.5V

Product Selection as of August 1994

P = Plastic DIP
J = Ceramic DIP
SM = 150mil SOIC
SN = 207mil SOIC
SL = SOIC

C = Commercial (0°C to 70°C)
I = Industrial (–40°C to 85°C)
E = Automotive (–40°C to 125°C)

* High Endurance Block.

APPLICATION SPECIFIC STANDARD PRODUCTS

The Application Specific Standard Products (ASSP) Division complements and strengthens Microchip's leadership position in 8-bit microcontrollers and related specialty memory products for the embedded control market. The ASSP Division employs innovative multi-chip module packaging, applications expertise, firmware and new technology to create integrated, single-chip solutions for specific high-volume embedded control applications such as PC pointing devices, TrueGauge™ battery management, and PICSEE™ microcontrollers. By offering more complete solutions, Microchip can provide its customers with higher value-added products, with the additional benefits of faster time-to-market and lower design overhead. A full ASSP product listing is shown in Table 4.

EASE OF PRODUCTION UTILIZING QUICK TURN PROGRAMMING (QTP) AND SERIALIZED QUICK TURN PROGRAMMING (SQTP)

Recognizing the needs of high-volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick Turn Programming allows factory programming of OTP product prior to delivery to the system manufacturing operation. PIC16/17, EPROM and Serial EEPROM products can be automatically programmed with the users program during the final stages of the test operation at Microchip's assembly and test operations in Philippine Islands, Taiwan and Thailand. This low-cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PIC16/17 and memory products as custom ROM products. With one- to four-week lead times on QTP product, the user no longer needs to plan for the extended ROM masking lead times and masking charges associated with custom ROM products. This capability, combined with the off-the-shelf availability of standard OTP product, ensures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique in the 8-bit microcontroller market is Microchip's ability to enhance the QTP capability with Serialized Quick Turn Programming (SQTP). SQTP allows for the programming of devices with unique, random or serialized identification codes. As each PIC16/17 device is programmed with the customers program code, a portion of the program memory space can be programmed with a unique code, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network are essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or nonvolatile memory components. The SQTP offering, pioneered by Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

TABLE 4 - ASSP PRODUCT SELECTION GUIDE

FIRMWARE PRODUCTS							
	Device	**Interface**	**Feature**	**Operating Voltage**	**Temp. Range**	**Number of Pins**	**Package Types**
BATTERY MANAGE-MENT	MTA11200 TrueGauge™	1-wire and RS232	Monitor and Charge Control for NiCD, NiMH Lead Acid	3.0V-6.25V	C,I	28	PDIP, SPDIP SOIC, SSOP
POINTING DEVICES	MTA41300	Serial, PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41120	ADB	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41110	PS/2	2 Button Mouse, Trackball	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
	MTA41111	PS/2	Velocity Scaling Trackball Controller	3.0V-6.25V	C,I	18	PDIP, SOIC, SSOP
PICSEE PRODUCTS							
Device		**Speed**	**Feature**	**Operating Voltage**	**Temp. Range**	**Number of Pins**	**Package Types**
MTA81010-RC		DC to 4 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC, JW
MTA81010-XT		DC to 4 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC, JW
MTA8R1010-RC		DC to 4 MHz	512 x 12 ROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA8R1010-XT		DC to 4 MHz	512 x 12 ROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA81010-LP		DC to 40 KHz	512 x 12 EPROM	2.5V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA8R1010-LP		DC to 40 KHz	512 x 12 ROM	2.0V-6.25V 1K EEPROM	C,I	28	PDIP, SOIC
MTA85401		DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85411		DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85402		DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85412		DC to 20 MHz	512 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85801		DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85811		DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 1K EEPROM	C,I,E	20	SSOP
MTA85802		DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP
MTA85812		DC to 20 MHz	2048 x 12 EPROM	3.0V-6.25V 2K EEPROM	C,I,E	20	SSOP

Introduction

WHAT'S NEW IN MICROCHIP'S 1994/95 EMBEDDED CONTROL HANDBOOK

Organization

The book is reorganized for increasing complexity of Application Notes in each section. A new section for generic PIC16/17 Application Notes has been added.

New Application Notes

SECTION 2 - PIC16C5X

Implementing Long Calls (AN581)
Macros for Page and Bank Switching (AN586)
Frequency Counter Using PIC16C5X (AN592)
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs (AN590)
Serial Port Routines Without Using the RTCC (AN593)
PIC16C54A EMI Results (AN577)

SECTION 3 - PIC16CXX

Using the CCP Module (AN594)
Interfacing to an LCD Module (AN587)
Using the 8-Bit Parallel Slave Port (AN579)
Using Timer1 in Sleep (AN580)
Low Power Clock (AN582)
Apple Desktop Bus (AN591)
I^2C Multi-Master Mode (AN578)
Programming the PIC16C84 Microcontroller (AN589)

SECTION 4 - PIC17CXX

Implementation of the Data Encryption Standard Using PIC17C42 (AN583)

SECTION 5 - PIC16/17

Disabling Global Interrupts (AN576)
IEEE 754 Compliant Floating Point Routines (AN575)
Real Time Operating System (AN585)
PIC16/17 Oscillator Design Guide (AN588)
Using PICMASTER's DDE (AN584)

SECTION 6 - ASSP

Calibrating the MTA11200 (AN570)
Communicating with EEPROM in MTA85XXX (AN571)
Hardware and Software Resolution for a Pointing Device (AN569)

SECTION 8 - SERIAL EEPROMS

Questions and Answers Concerning Serial EEPROMs (AN572)

SECTION 9 - DEVELOPMENT TOOLS

System Information Hotline
PICDEM-2
MP-C
*fuzzy*TECH-MP

WHAT'S CHANGED IN MICROCHIP'S 1994/95 EMBEDDED CONTROL HANDBOOK

Modified / Corrected Application Notes

Many application notes have had many "minor" corrections implemented, such as spelling and source code alignment. The following list is intended to point out the application notes with "major" modifications or corrections.

SECTION 3 - PIC16CXX

Software Implementation of Asynchronous Serial I/O (AN555)

SECTION 4 - PIC17CXX

PIC17CXX Math Utility Routines (AN544)

SECTION 5 - PIC16/17

PIC16C5X / 16CXX Math Utility Routines (AN526)

SECTION 8 - SERIAL EEPROMS

Interfacing 24LCXXB Serial EEPROMS to the PIC16C54 (AN567)
Using the 24C65 and 24C32 with Stand-alone PIC16/17 Code (AN558)
Using the 93LC56 and 93LC66 (AN560)

SECTION 9 - DEVELOPMENT TOOLS

The Development Tools Section has been updated.

SECTION 2
PIC16C5X APPLICATION NOTES

AN520

A Comparison of Low End 8-Bit Microcontrollers

INTRODUCTION

The PIC16C5X Family of microcontrollers from Microchip Technology, Inc. provides significant execution speed and code-compaction improvement over any other 8-bit microcontroller in its price range.

The superior performance of the PIC16C5X microcontrollers can be attributed primarily to its RISC-like architecture. The PIC16C5X employs Harvard architecture, i.e., has separate program memory space (12-bit wide instructions) and data memory space (8-bit wide data). It also uses a two stage pipelining instruction fetch and execution. All instructions are executed in a single cycle (200 ns @ 20 MHz clock) except for program branches which take two cycles, and there are only 33 instructions to remember.

Separation of program and data space allows the instruction word to be optimized to any size (12-bit wide in case of PIC16C5X). This makes it possible, for example, to load an 8-bit immediate value in one cycle. First, because there is no conflict between instruction fetch and data fetch (as opposed to von Neumann architecture) and secondary because the instruction word is wide enough to hold the 8-bit data.

In the following sections we will compare the PIC16C5X @ 20 MHz with:

- SGS-Thomson ST62 @ 8 MHz
- Motorola MC68HC05 @ 4.2 MHz
- Intel 8048/8049 @ 11 MHz
- Zilog Z86CXX @ 12 MHz
- National COP800 @ 20 MHz

Several coding examples will be considered. While the comparisons are not entirely scientific, they will, nevertheless, demonstrate to the reader the relative superior performance of the PIC16C5X. The examples chosen here are used frequently in microcontroller applications.

PACKING BCD

This example will take two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

PIC16C5X

		Byte/Words	Cycles
SWAPF	REGHI,W	1	1
IORWF	REGLO	1	1
		2	2

0.4 µs

COP800

		Byte/Words	Cycles
X	A,[B+]	1	2
SWAP	A	1	1
OR	A,[B]	1	1
X	A,[B]	1	1
		4	5

B is pointing to the higher BCD digit initially. After auto-increment, it points to the lower BCD digit.

5 µs

ST62

		Byte/Words	Cycles
LD	A, REGHI	2	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
ADD	A, REGLO	1	4
LD	REGLO, A	2	4
		10	28

45.5µs

REGHI & REGLO are registers addressable by short direct addressing mode.

MC68HC05

		Byte/Words	Cycles
LDA	REGHI	2	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ADD	REGLO	2	3
STA	REGLO	2	4
		10	22

10.5 µs

Z86CXX

		Byte/Words	Cycles
SWAP	REGHI	2	8
OR	REGHI,REGLO	2	6
		4	14

5.33 µs

REGHI and REGLO are addressable via the working register addressing mode.

8048/8049

		Byte/Words	Cycles
MOV	A,Rx	1	1
SWAP	A	1	1
ORL	A,Ry	1	1
MOV	Ry,A	1	1
		4	4

Register Rx contains higher BCD digit, Ry holds lower BCD digit.

5.45 µs

LOOP CONTROL

This example is one of simple loop control where a register containing loop count is decremented, tested for zero and if not branched back to the beginning of the loop.

PIC16C5X

		Byte/Words	Cycles
DECFSZ	COUNT	1	1/2
GOTO	BEG_LOOP	1	2/-
		2	3/2

0.6µs/0.4 µs

COP800

		Byte/Words	Cycles
DRSZ	COUNT	1	3
JP	BEG_LOOP	1	3
		2	6

COUNT is Register (RAM F0h-FFh). 6 µs

ST62

		Byte/Words	Cycles
DEC	X	1	4
JRZ	BEG_LOOP	1	2
		2	6

9.75 µs

MC68HC05

		Byte/Words	Cycles
DECX		1	3
BEQ	BEG_LOOP	2	3
		3	6

2.86 µs

Z86CXX

		Byte/Words	Cycles
DJNZ	COUNT, BEG_LOOP	2	10/12

1.67 µs/2 µs

8048

		Byte/Words	Cycles
DJNZ	Rx, BEG_LOOP	2	2

2.73 µs

Bit Test & Branch

This example tests a single bit in a register or a RAM location and makes a conditional branch. We assume that MSB is tested and a branch is to be taken if the bit is set.

PIC16C5X

		Byte/Words	Cycles
BTFSC	REG, 7	1	1/2
GOTO	NEWADD	1	2/-
		2	3/2

0.6 µs/0.4 µs

COP800

		Byte/Words	Cycles
IFBIT	7, [B]	1	1
JP	NEWADD	1	3
		2	4

B points to the memory location under test. 4 µs

ST62

		Byte/Words	Cycles
JRR	7, NEWADD	3	5

8.125 µs

MC68HC05

		Byte/Words	Cycles
BRCLR	7, NEWADD	3	5

2.38 µs

Z86CXX

		Byte/Words	Cycles
BTJRT	NEWADD, REG, 7	3	16/18

2.67 µs/3.0 µs

8048/8049

		Byte/Words	Cycles
MOV	A, @Rx	1	1
ANL	A, #80H	2	2
JNZ	NEWADD	2	2
		5	5

Registers R1 is assumed to be pointing to the memory location under test. 6.82 µs

Shifting Out 8-Bit Data & Clock

We will now consider the task of serially shifting out an 8-bit data and clock. Data and clock outputs are generated under program control by toggling two output pins.

Data is transmitted on the rising edge of the clock. No attempt is made to make the clock output symmetrical in order to make the code efficient. Data out is guaranteed on the falling edge of the clock. These conditions are satisfactory for most applications.

PIC16C5X

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	MOVLW	08H	; Bit Count	1	1	1	
	MOVWF	BITCNT	;	1	1	1	
XM1	BCF	PORTB,0	; 0 → Data Out Pin	1	1	1	
	BCF	PORTB,1	; 0 → Clock Out Pin	1	1	1	
	RRF	XDATA	; Rotate Right thru Carry	1	1	1	
	BTFSC	STATUS,CARRY	; Test Carry Bit	1	2	1	
	BSF	PORTB,0	; 1→ Data Out Pin	1	–	1	Loop
	BSF	PORTB,1	; 1→ Clock Out Pin	1	1	1	
	DECFSZ	BITCNT	; Decrement Count	1	1	1	
			; Skip if Zero				
	GOTO	XM1	;	1	2	2	
	BCF	PORTC,1	; 0→ Clock	1	1	1	
				11	74	74	

Transmit time is the same for 00h or FFh: 74 Tcyc = 14.8 µs. Note that there was no need to load the data in the Accumulator (W) since the PIC16C5X can operate directing on file registers.

COP800

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LD	A,XDATA	; Load Data in Acc.	2	3	3	
	LD	BITCNT #08H	; Load Bit Count	2	3	3	
	LD	B,#D0H	; B Points to PORTL	2	3	3	
XM1	RBIT	0,[B]	; 0 → Clock	1	1	1	
	RBIT	1,[B]	; 0 → Data	1	1	1	
	RRCA		; Rotate A Right thru Carry	1	1	1	
	IFC		;	1	1	1	Loop
	SBIT	1,[B]	; 1 → Data	1	–	1	
	SBIT	0,[B]	; 0 → Clock	1	1	1	
	DRSZ	BITCNT	; Decrement Bit Count	1	3	3	
	JP	XM1	; and Go Back if ≠ 0	1	3	3	
	RBIT	0,[B]	;	1	3	3	
				16	100	108	

Accumulator A is first loaded with the data word. Transmit time is maximum for data = FFh: 105 Tcyc = 105 µs.

ST62

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
	LDI	A, #08		2	4	4	
	LD	X, A	; Bit Count	1	4	4	
	LD	A, W	;Xmit Data	1	4	4	
XM1	RES	0, DRB	; 0 → Clock	2	4	4	
	RES	1, DRB	; 0 → Data	2	4	4	
	SLA	A	;	2	4	4	
	JRNC	XM2	;	1	2	2	
XM2	SET	1, DRB	; 1 → Data	2	–	4	Loop
	SET	0, DRB	; 1 → CLK	2	4	4	
	DEC	X	;	1	4	4	
	JRNZ	XM1	;	1	2	2	
	RES	0, DRB	; 0 → Data	2	4	4	
				19	208	240	

Register W contains the Data word.
Transmit time for FFh = 240 cycles = 390 µs.

SHIFTING OUT 8-BIT DATA AND CLOCK (CONT.)

MC68HC05

```
                                                        Byte    Cycles   Cycles
                                                       /Words  Xmit 00h Xmit FFh
XMIT    LDA     XDATA   ; Load Xmit Data
        LDX     #$08    ; Load Bit Count                 2        3        3
                                                         2        2        2
XM1     BCLR,   0,PORTB ; 0 -> Clock                                        |
        BCLR    1,PORTB ; 0 -> Data                      2        5        5
        ROLA            ;                                2        5        5
        BCC     XM2     ;                                1        3        3   } Loop
        BSET    1,PORTB ; 1 -> Data                      2        3        3
        BSET    0,PORTB ; 1 -> Clock                     2        –        5
        DECX            ;                                2        5        5
XM2     BNE     XM1     ;                                1        3        3
        BCLR    0,PORTB ; 0 -> Data                      2        3        3
                                                         2        5        5
                                                        20      226      266
```

Transmit time is maximum for transmitting FFh = 266 cycles = 126.7 µs.

Z86CXX

```
                                                        Byte    Cycles   Cycles
                                                       /Words  Xmit 00h Xmit FFh
XMIT    LD      COUNT,#8 ; Load Bit Count                3       10       10
        AND     P2,#%FC  ; 0 -> Data,                    2        6        6
                         Clock
XM1     RRC     XDATA    ;
        JR      NC,XM2   ;
        OR      P2, #01  ; 1 -> Data                     2        6        6
                                                         2       12       10
                                                         3        –       10   } Loop
XM2     OR      P2, #02  ; 1 -> Clock                                       |
        DJNZ    COUNT,XM1;                               3       10       10
        AND     P2,#%FC  ; 0 -> Clock, Data              2       12       12
                                                         3       10       10
                                                        21      348      412
```

Transmit time is maximum for transmitting FFh = 412 cycles = 68.67 µs.

8048/8049

```
                                                        Byte    Cycles   Cycles
                                                       /Words  Xmit 00h Xmit FFh
XMIT    MOV     A,@R0   ; R0 Points to Data Word           1        1        1
        MOV     R1,#08H ; Load Bit Count                 2        2        2

XM1     ANL     PORT1,#0FCH ; 0 -> Data, Clock           2        2        2
        RRC     A       ; Rotate Right A thru Carry      1        1        1
        JC      XM2     ;                                2        2        2
        ORL     PORT1,#01H ; 1 -> Data                   2        –        2   } Loop

XM2     ORL     PORT1,#02H ; 1 -> Clock                  2        2        2
        DJNZ    R1,XM1  ; Decrement Count                2        2        2
                                                        14       75       91
```

Transmit time is maximum for transmitting FFh = 91 cycles = 124.1 µs.

Software Timer

Microcontrollers quite often need to implement time delays. Debouncing key input, pulse width modulation, and phase angle control are just a few examples. Implementing a 10 ms time delay loop subroutine will be considered in this section.

PIC16C5X

				Byte/Words	Cycles
DELAY	MOVLW	41H	;10 ms Delay Loop	1	1
	MOVWF	COUNT2	;	1	1
	CLRF	COUNT1	;	1	1
LOOP	INCFSZ	COUNT1	;This Inner Loop will be	1	2/1
	GOTO	LOOP	; Executed 256 Times	1	2
	DECFSZ	COUNT2	;	1	2/1
	GOTO	LOOP	;	1	2
	RET		;	1	2
				8	

Execution time for the routine = 5 + (255 x 3 + 5) • 65 = 20025 Tcyc = 10.011 ms. The PIC16C5X can implement delay times very precisely (when necessary) because of its fine instruction cycle resolution.

COP800

				Byte/Words	Cycles
DELAY	LD	COUNT1,#0BH	;10 ms Delay Loop	2	3
	LD	B,#0EH	;	1	1
LOOP	DRSZ	B	;	1	1
	JP	LOOP	;	1	1
	DRSZ	COUNT1	;	1	1
	JP	LOOP	;	1	1
	RET		;	1	5
				8	

Execution time for the routine = (6N2 + 6) N1 + 9 cycles. Here N1 = 0BH and N2 = 0EH, which gives us: 999 Tcyc = 9.99 ms.

ST62

				Byte/Words	Cycles
	LDI	A, #FF		2	4
	LD	X, A	; LOOP1 Count	1	4
	LDI	A, #04		2	4
	LD	Y, A	; LOOP2 Count	1	4
LOOP	DEC	X	; 0 CLK	1	4
	JRNZ	LOOP	;	1	2
	DEC	Y	; 0 CLK	1	4
	JRNZ	LOOP		1	2
				10	

Execution time for the subroutine = (6N1 + 6) N2 + 16 cycles, where N1 = FFh, N2 = 04 gives us 10.01 ms.

MC68HC05

				Byte/Words	Cycles
DELAY	LDX	$2D	;10 ms Delay Loop	2	2
	LDX	$5C	;	2	2
LOOP	DECA		;	1	3
	BNE		;	2	2
	DECX	LOOP	;	1	3
	BNE		;	2	2
	RTS	LOOP	;	1	6
				11	

Execution time for the subroutine = (5 x N1 + 5) N2 + 10, with N1 = 2DH, N2 = 5CH, time delay = 10.081 ms.

SOFTWARE TIMER (CONT.)

Z86CXX

				Byte/Words	Cycles
DELAY	LD	COUNT1,#%61	;10 ms Delay Loop	2	6
	LD	COUNT2,#%33	;	2	6
LOOP	DJNZ	COUNT1,LOOP	;		
	DJNZ	COUNT2,LOOP	;	2	10/12
	RET		;	2	10/12
				1	14
				9	

Total execution time = (12N1 + 10) N2, with N1 = 61H, N2 = 33H, time delay = 59976 cycles = 9.979 ms.

8048/8049

				Byte/Words	Cycles
DELAY	MOV	COUNT1,#13H	;10 ms Delay Loop	2	2
LOOP1	MOV	COUNT2,#AFH	;	2	2
LOOP2	DJNZ	COUNT2,LOOP2	;	2	2
	DJNZ	COUNT1,LOOP1	;	2	2
	RET		;	1	2
				9	

Execution time for the subroutine = (2N1 + 4) N2 + 4 cycles. Here N1 = 13H, N2 = AFH, which gives us: 7354 cycles = 10.028 ms.

A Comparison of Low-End 8-Bit Microcontrollers

SUMMARY

Table 1 summarizes code sizes for different microcontrollers. The overall relative code size number is an average of the individual relative code sizes. Given that the PIC16C5X's program word size is 12-bit, whereas all the other microcontrollers have 8-bit program memory, a compaction of 1.5 µs is expected. Clearly, the PIC16C5X meets this compaction (except for the COP800) and exceeds in most comparisons.

Table 2 summarizes relative execution speed. The overall speed is an average of relative speed numbers. For example, the COP800 will, on an average, exhibit 27% of the code execution speed of a PIC16C5X. In other words, the PIC16C5X will be 1/0.27 = 3.7 times faster than a COP800 on an average.

2

TABLE 1 - COMPARISON OF CODE EFFICIENCY*

Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800	4 2.00	2 1.00	2 1.00	16 1.46	8 1.00	1.29
ST62	10 5.00	2 1.00	3 1.50	19 1.73	10 1.25	2.10
MC68HC05	10 5.00	3 1.50	3 1.50	20 1.82	11 1.38	2.24
Z86CXX	4 2.00	2 1.00	3 1.50	21 1.91	9 1.125	1.51
8048/8049	4 2.00	2 1.00	5 2.51	14 1.28	9 1.13	1.58
PIC16C5X @ 8 MHz	2	2	2	11	8	1.00

***** In each box, the top number is the number of program memory locations required to code the application. The bottom number is relative code size compared to the PIC16C5X:

$$\frac{\text{\# program memory locations for other microcontroller}}{\text{\# program memory locations for the PIC16C5X}}$$

FIGURE 1 - CODE SIZE COMPARISON

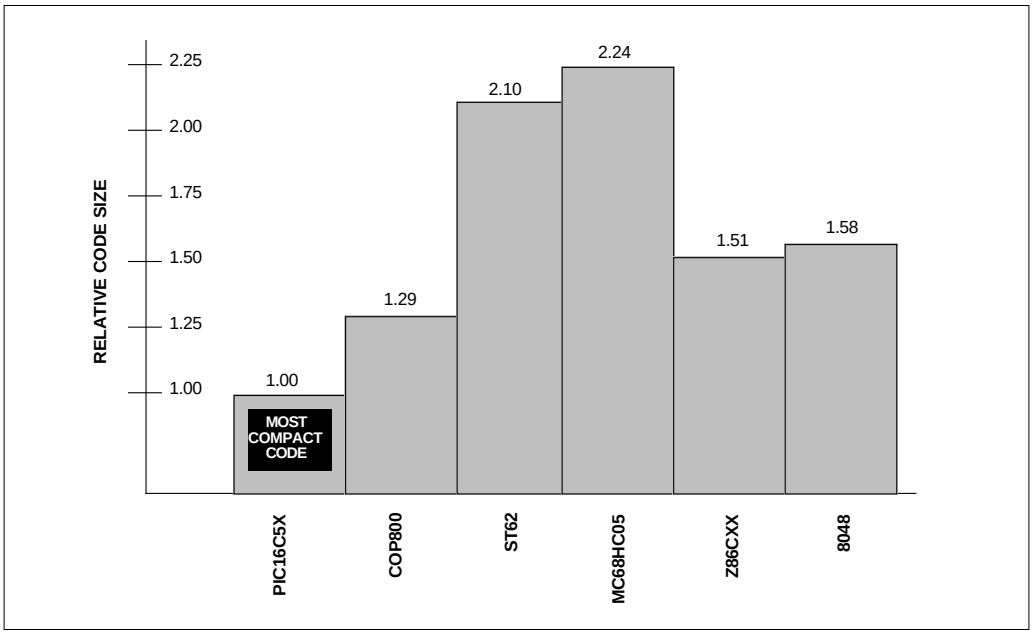


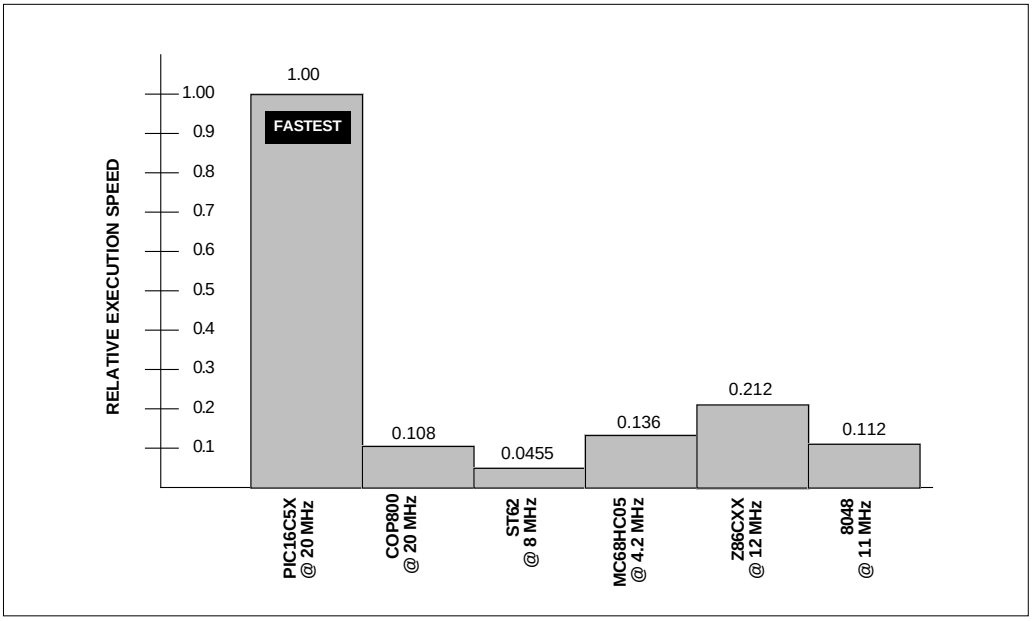
TABLE 2 - COMPARISON OF EXECUTION SPEED*

Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800 @ 20 MHz	5 µs 0.08	6 µs 0.0832	4 µs 0.1252	105 µs 0.1408	–	0.108
ST62 @ 8 MHz	45.5 µs 0.0088	9.75 µs 0.0615	8.125 µs 0.0738	390 µs 0.0329	–	0.0455
MC68HC05 @ 4.2 MHz	10.05 µs 0.038	2.86 µs 0.1748	2.38 µs 0.21	126.7 µs 0.1168	–	0.136
Z86CXX @ 12 MHz	2.33 µs 0.172	1.835 µs 0.272	2.835 µs 0.176	68.67 µs 0.224	–	0.212
8048/8049 @ 11 Mhz	5.45 µs 0.0732	2.73 µs 0.1824	6.82 µs 0.0732	124.1 µs 0.1196	–	0.112
PIC16C5X @ 20 MHz	0.4 µs	0.6/0.4 µs	0.6/0.4 µs	14.8 µs	–	1.00

***** In each box, the top number is the time required to execute the example code, while the bottom number is a measure of relative performance compared to the PIC16C5X:

$$\frac{\text{time required to execute code by the PIC16C5X}}{\text{time required to execute code by other microcontroller}}$$

FIGURE 2 - EXECUTION SPEED COMPARISON



NOTES:

AN522

Power-Up Considerations

INTRODUCTION

When powering up all microcontrollers it is necessary for the power supply voltage to traverse voltage ranges where the device is not guaranteed to operate before the power supply voltage reaches its final state. Since some circuits on the device (logic) will start operating at voltage levels lower than other circuits on the chip (memory), the device may power-up in an unknown state. To guarantee that the device starts up in a known state, it is necessary that it contain a power-up reset circuit. PIC16C5X microcontrollers are equipped with on-chip power-on reset circuitry, which eliminates the need for external reset logic. This circuit will function in most power-up situations where V_{CC} rise time is fast enough (50 ms or less). This application note describes the typical power-up sequence for PIC16C5X microcontrollers. Methods of assuring reset on power-up and after a brownout are discussed and simple, low cost external solutions are discussed for power-up situations where the PIC16C5X's internal circuitry cannot provide the reset.

POWER-UP SEQUENCE

The PIC16C5X incorporates complex power-on reset (POR) circuitry on-chip which provides solid, reliable internal chip reset for most power-up situations. To use this feature, the user merely needs to tie $\overline{MCLR}$ to V_{DD}. A simplified block diagram of the on-chip reset circuitry is shown in Figure 1. On power-up, the reset latch and the start-up timer are reset to appropriate states by the power-on reset (POR). The start-up timer will begin counting once it detects $\overline{MCLR}$ to be high (i.e., external chip reset goes inactive). After the time-out period, which is typically 18 ms long, the timer will reset the reset latch and thus end the on-chip reset signal.

Figures 2 and 3 are two power-up situations with relative fast rise time on V_{DD}. In Figure 1, V_{DD} is stable when $\overline{MCLR}$ is brought high (i.e., reset pulse is being provided by external source). The chip actually comes out reset about t_{OST} ms after that, where t_{OST} = oscillator start-up timer. (The timer is called oscillator start-up timer because the time-out was incorporated primarily to allow the crystal oscillator to stabilize on power-up.) In Figure 3, the $\overline{MCLR}$ and V_{DD} are tied together and clearly the on-chip rest mechanism is being utilized. The V_{DD} is stable before the start-up timer expires and there is no problem with proper reset.

Figure 4, where V_{DD} rise time is much greater than t_{OST} (typically 18 ms) clearly is the potentially problematic situation. The POR (power-on reset) pulse comes when V_{DD} is about 1.5V. Most CMOS logic, including the start-up timer starts functioning between 1.5V to 2.0V. When the start-up timer starts times out, the chip reset is ended and the chip attempts to execute. If by this time the V_{DD} has reached V_{DD} MIN value, then all circuits are guaranteed to function correctly and power-up reset is successful. If, however, the V_{DD} slope was too slow and had not reached V_{DD} MIN, then the chip may or may not function properly.

FIGURE 1 - PIC16C5X INTERNAL RESET CIRCUIT

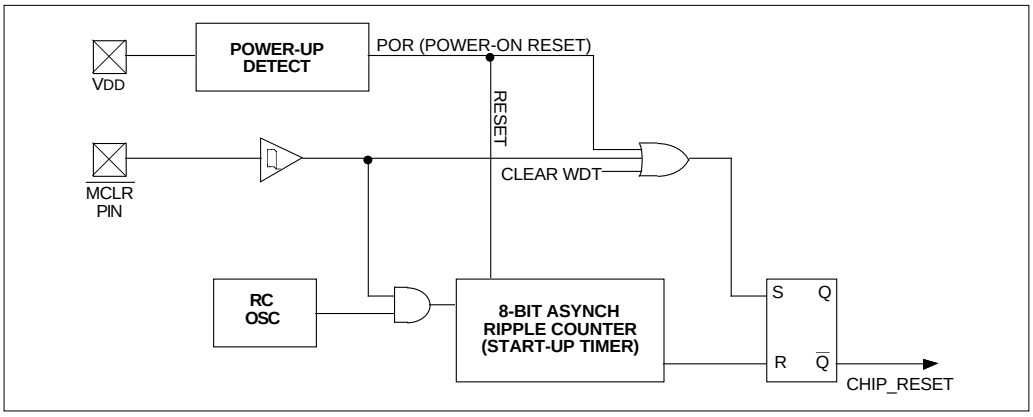


FIGURE 2 - EXTERNAL RESET PULSE

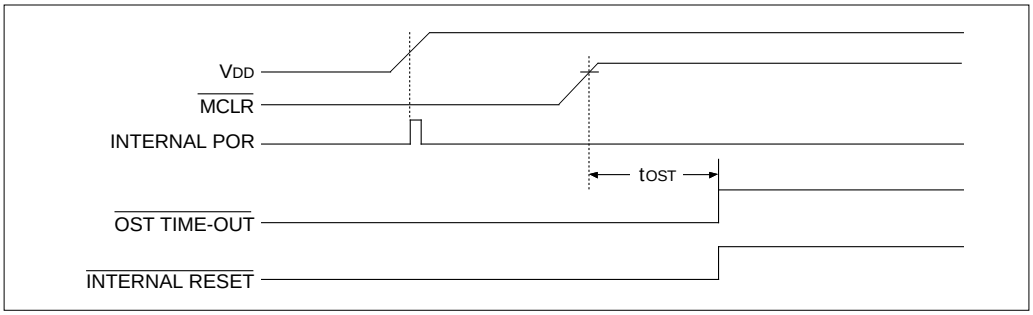


FIGURE 3 - INTERNAL RESET (V_{DD} AND $\overline{MCLR}$ TIED TOGETHER)

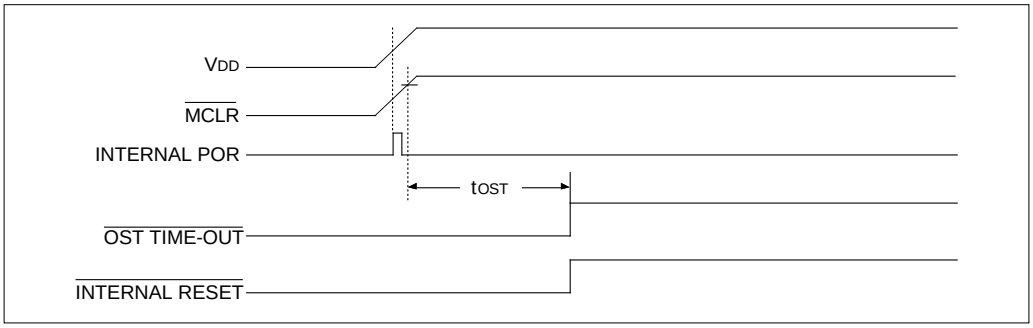
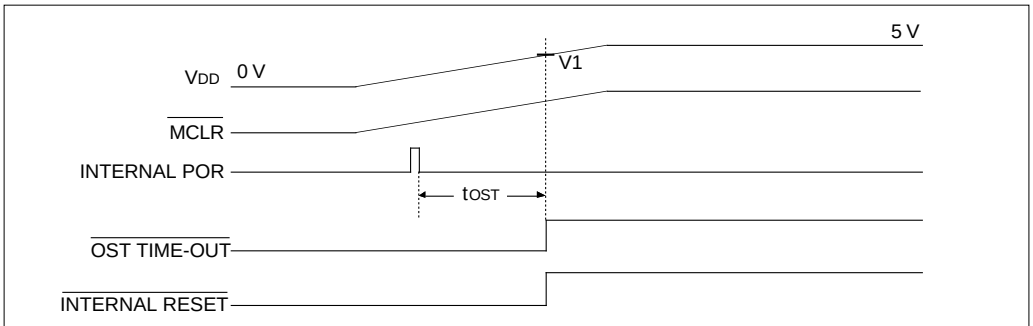


FIGURE 4 - INTERNAL RESET (V_{DD} AND $\overline{MCLR}$ TIED TOGETHER): SLOW V_{DD} RISE TIME

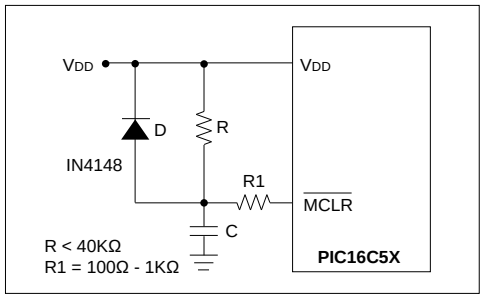


When V_{DD} rises slowly, the internal time-out period expires long before V_{DD} has reached its final value. In this example, the chip will reset properly if, and only if, $V1 \geq V_{DDMIN}$.

EXTERNAL POWER-ON RESET CIRCUIT

To use power supplies with slow rise times it is necessary to use an external power-on reset circuit such as the one shown in Figure 5. This circuit uses an external RC to generate the reset pulse. The time constant of the RC should be long enough to guarantee that the reset pulse is still present until V_{DD} has reached V_{DD} min. R should be 40K or less to guarantee that the $\overline{MCLR}$ will pull to within 0.2 volts of V_{DD}. (since the leakage spec on $\overline{MCLR}$ is ±5 µA, a resistor larger than 40K may cause input high voltage on this pin to be less than V_{DD} – 0.2V, the required spec). The diode D is used to rapidly discharge the capacitor on power-down. This is very important as a power-up reset pulse is needed after a short power-down (less than the time constant of RC) or after a power spike. The resistor R1 protects against high current flowing into $\overline{MCLR}$ pin from fully charged capacitor C in the event $\overline{MCLR}$ pin breakdown is induced through ESD or EOS. The circuit, however, does not protect against brown-out situations where the power does not drop to zero, but merely dips below V_{DD} MIN. In such a situation, voltage at the $\overline{MCLR}$ pin will not go low enough (i.e., below V_{IL}) to guarantee a reset pulse. The following section presents an example circuit to protect against such brown-outs.

FIGURE 5 - EXTERNAL POWER-ON RESET CIRCUIT



BROWNOUT PROTECTION

In many applications it is necessary to guarantee a reset pulse whenever V_{DD} is less than V_{DD} min. This can be accomplished using a brownout protection circuit such as the one shown in Figure 6. This is a simple circuit that causes a reset pulse whenever V_{DD} drops below the zener diode voltage plus the Vbe of Q1. A 3.3 volt zener will produce a reset pulse whenever V_{DD} drops below about 4 volts. This circuit has a typical accuracy of about ±100 mV. A less expensive, albeit less precise, brownout circuit is shown in Figure 7. Transistor Q1 turns off when Vbe = V_{DD} • R1/(R1+R2) falls below 0.7 V allowing R3 to pull down $\overline{MCLR}$ input.

FIGURE 6 - BROWNOUT PROTECTION CIRCUIT

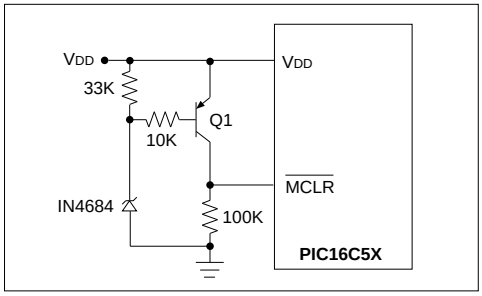
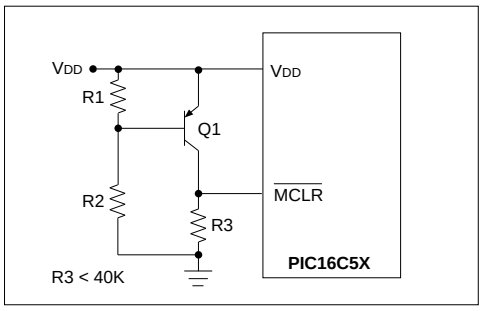


FIGURE 7 - BROWNOUT PROTECTION CIRCUIT



Author: Sumit Mitra
Logic Products Division

NOTES:

Software Interrupt Techniques

INTRODUCTION

This application note describes a unique method for implementing interrupts in software on the PIC16C5X series of microcontrollers. The method takes advantage of the PIC16C5X's architecture which allows changing the program counter under software control. Up to eight interrupt lines are possible, but the practical limit for simple code generation is six interrupts, or 64 possible input conditions. The interrupt detection time is under software control and standard I/O pins are used as the interrupt lines.

THEORY OF OPERATION

SOFTWARE POLLING OF I/O LINES REPLACES HARDWARE INTERRUPT

The interrupt conditions are determined by detecting changes on the I/O lines that have been selected to be the interrupt lines. These changes are used to create a jump table that allows a different program response to each interrupt condition. The interrupt response time is under software control and can be as short as ten to twenty microseconds, depending on main program and interrupt subroutine program length.

CREATING THE INTERRUPT SUBROUTINE JUMP TABLE

Each I/O condition may have its own unique subroutine to respond to changes on the interrupt lines. Direct access to these routines is achieved by using the PIC16C5X's ability to change the program counter under software control. Here is an example of how two I/O lines may be polled:

```
MOVF   CONDTN,W  ;LOAD I/0 CONDITION INTO W
                 ;REGISTER
ANDLW  3         ;MASK OFF TOP 6 BITS
ADDWF  2,1       ;ADD INPUT TO PROGRAM COUNTER
                 ;TO CREATE JUMP TABLE
GOTO   MAIN      ;FOR  NO CHANGE GO TO MAIN
                 ;PROGRAM
GOTO   INT1      ;FOR CHANGE IN BIT 0 GOTO INT1
GOTO   INT2      ;FOR CHANGE IN BIT 1 GOTO INT2
GOTO   INT3      ;FOR BOTH CHANGE GOTO INT3
```

The changes to the I/O lines have been used to create a two bit number that is added to the program counter. The GOTO that is executed depends on the new program counter address.

CREATING CONSTANT TIME POLLING

In most applications requiring interrupts, it is important to poll the interrupt lines at fixed time intervals, usually only a few microseconds in length. Two techniques may be used on the PIC16C5X to achieve this. They are dividing the main program into multiple sections and implementing an elapsed time counter (see flow chart). Both of these techniques use the same program jump table concept that was described above. First, the main program is divided into several sections based on the desired I/O polling time. When MAIN is called a branch register is added to the program counter. This determines which section of MAIN code should be executed next. At the end of execution the branch register is decremented so the next section of code will be executed after the next polling. If the branch register is zero then the number of sections of main code is added into it to start the main program over again.

An elapsed time counter can be implemented using the RTCC counter. At the beginning of I/O polling the RTCC register is cleared. It then starts counting the instruction cycles. Then after the main program subsection has been executed, the RTCC register is subtracted from the desired polling time. This determines how many instructions need to be executed before the next polling. A jump table is then created to execute these instructions before the next polling. An example is shown below. This example assumes from zero to 15 additional instruction cycles are needed. Actual numbers need to be computed for each individual application.

```
MOVLW  POLL    ;POLL=DESIRED POLL CYCLES - 15
SUBWF  RTCC,W  ;DETERMINE HOW MUCH TIME TO WAIT
ADDWF  2,1     ;ADD  WAIT   TIME  TO  PROGRAM
COUNTER
NOP            ;15 ADDITIONAL INSTRUCTION CYCLES
:
:              ;TOTAL OF 15 NOP'S
NOP            ;1 ADDITIONAL INSTRUCTION CYCLES
GOTO   START   ;0 ADDITIONAL INSTRUCTION CYCLES
```

For example, if the desired instruction time is 50 cycles and the subsection we just executed has a consumed a total of 40 instruction cycles (including all overhead cycles) the value of

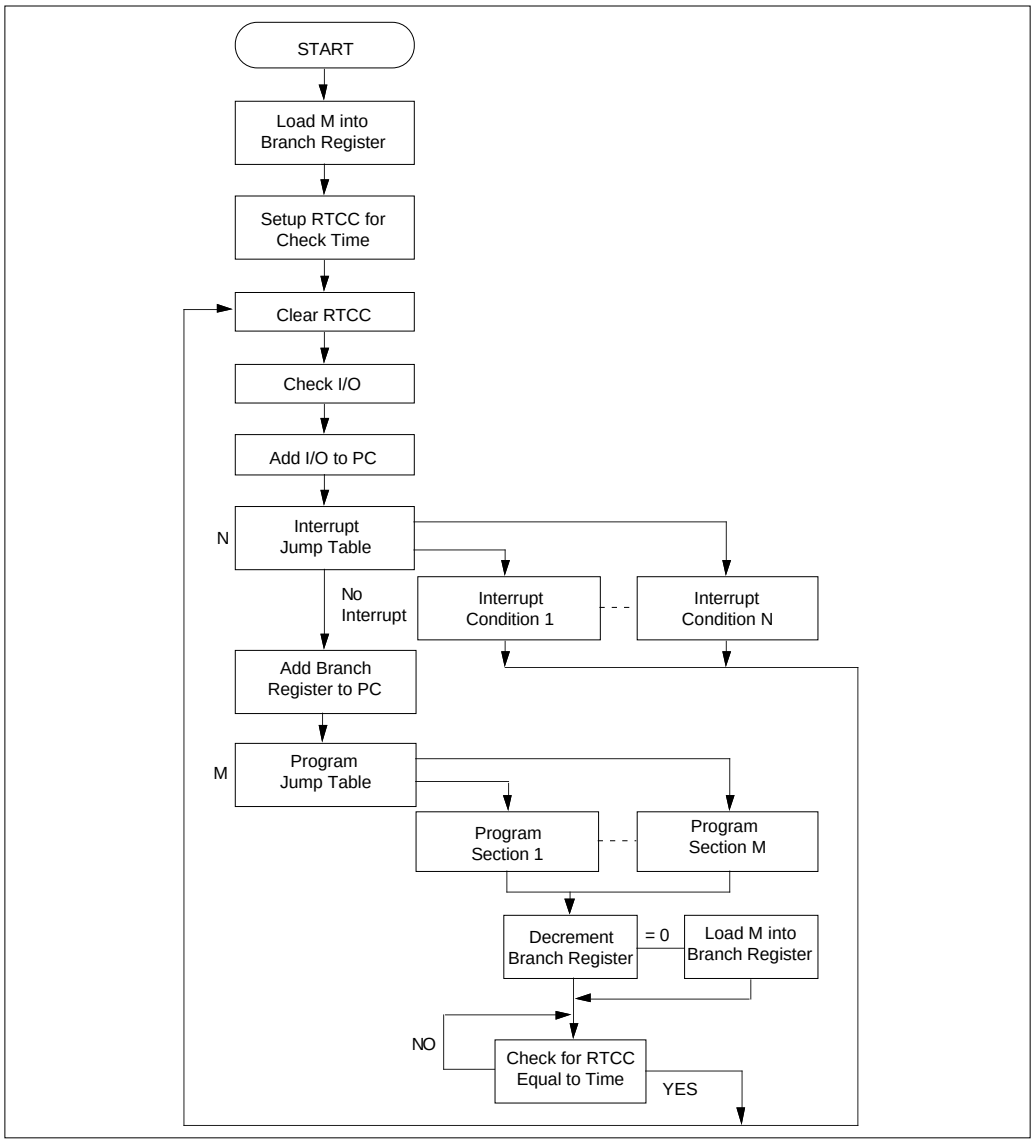
RTCC(40) - POLL(50-15(35)) =5

will be added to the program counter. The program will then jump to the sixth NOP. That NOP plus the 9 following it will be executed for a total of ten more instruction cycles. Note that the final GOTO has two instruction cycles and these must be included in the program overhead.

Example

The following example (see flow chart and code) is the core program for the software interrupt technique described above. This program assumes four interrupt conditions, four main program sections and an eight additional elapsed time instructions.

FIGURE 1 - SOFTWARE INTERRUPT FLOW CHART



APPENDIX A:

```
MPASM B0.54                                                    PAGE  1


                        LIST    P=16C54

                                        ;SOFTWARE INTERRUPT APPLICATIONS
                                        ;BRANCH IS MAIN PROGRAM REGISTER
0008                BRANCH EQU     8
0009                CNDTN  EQU     9
000A                IO     EQU     0A
000B                TEMP   EQU     0B

0000 0069           SETUP  CLRF    CNDTN
0001 0C04                  MOVLW   4
0002 0028                  MOVWF   BRANCH  ;FOUR MAIN PROGRAM SECTIONS
0003 0C08                  MOVLW   8
0004 0002                  OPTION          ;SET RTCC TO ONE COUNT PER INSTRUCTION CYCLE

0005 0061           START  CLRF    1       ;CLEAR RTCC REGISTER
0006 0206                  MOVF    6,W     ;READ I/O
0007 002A                  MOVWF   IO
0008 0109                  IORWF   CNDTN,W ;THIS SECTION OF CODE CALCULATES THE
0009 002B                  MOVWF   TEMP    ;JUMP TABLE. ANY INPUT THAT CHANGES FROM
000A 0209                  MOVF    CNDTN,W ;A ZERO TO A ONE IS CONSIDERED AN INTERRUPT.
000B 00AB                  SUBWF   TEMP,1  ;THE EQUATION IS:
000C 020A                  MOVF    IO,W    ;  (IO + CNDTN) - CNDTN = INTERRUPT
000D 0029                  MOVWF   CNDTN   ;WHERE IO IS CURRENT INPUT AND
000E 020B                  MOVF    TEMP,W  ;CNDTN IS PREVIOUS INPUT.
000F 0E03                  ANDLW   3       ;MASK OFF TOP 6 BITS
0010 01E2                  ADDWF   2,1     ;ADD INPUT TO PC TO CREATE JUMP TABLE
0011 0A1B                  GOTO    MAIN    ;FOR INPUT=00
0012 0A15                  GOTO    INT1    ;FOR INPUT=01
0013 0A17                  GOTO    INT2    ;FOR INPUT=10
0014 0A19                  GOTO    INT3    ;FOR INPUT=11

0015 0000           INT1   NOP             ;INTERRUPT LINE 1 CODE
0016 0A05                  GOTO    START
0017 0000           INT2   NOP             ;INTERRUPT LINE 2 CODE
0018 0A05                  GOTO    START
0019 0000           INT3   NOP             ;INTERRUPT LINES 1 AND 2 CODE
001A 0A05                  GOTO    START

001B 0208           MAIN   MOVF    BRANCH,W
001C 01E2                  ADDWF   2,1     ;ADD BRANCH TO PC TO CREATE JUMP TABLE
001D 0000                  NOP
001E 0A28                  GOTO    MAIN4   ;JUMP TABLE, LAST FIRST ON DECREMENT TABLE
001F 0A26                  GOTO    MAIN3
0020 0A24                  GOTO    MAIN2
0021 0A22                  GOTO    MAIN1

0022 0000           MAIN1  NOP             ;MAIN PROGRAM CODE BANK ONE
0023 0A2A                  GOTO    BRNCHK
0024 0000           MAIN2  NOP             ;MAIN PROGRAM CODE SECTION TWO
0025 0A2A                  GOTO    BRNCHK
0026 0000           MAIN3  NOP             ;MAIN PROGRAM CODE SECTION THREE
0027 0A2A                  GOTO    BRNCHK
0028 0000           MAIN4  NOP             ;MAIN PROGRAM CODE SECTION FOUR
0029 0A2A                  GOTO    BRNCHK

002A 02E8           BRNCHK DECFSZ  BRANCH,1 ;DECREMENT BRANCH REGISTER AND CHECK FOR ZERO
002B 0A2E                  GOTO    TIMCHK
002C 0C04                  MOVLW   4
002D 0028                  MOVWF   BRANCH  ;RELOAD BRANCH WITH 4 AT END OF MAIN

002E 0C29           TIMCHK MOVLW   D'41'   ;CHECK TO SEE IF RTCC HAS REACHED 50(50-7)
```

```
002F 0081                SUBWF   1,W     ;DETERMINE WAIT TIME
0030 01E2                ADDWF   2,1     ;ADD WAIT TIME TO PC
0031 0000                NOP
0032 0000                NOP
0033 0000                NOP
0034 0000                NOP
0035 0000                NOP
0036 0000                NOP
0037 0000                NOP
0038 0A05                GOTO    START
                         END


Errors   :    0
Warnings :    0
```

AN527

Software Stack Management

INTRODUCTION

The PIC16C5X has a stack which is only 2 deep, as a result of which only two nested calls can be made (i.e. only one call within a call routine). If more than two levels of subroutine nesting is required, this application note can be used to implement a stack manager to handle the flow of the calls.

Note: Since the amount of RAM on the PIC16CXX is limited, it would be prudent to determine the maximum number of nested calls which have to be made in a program and define the stack length appropriately.

IMPLEMENTATION

This application note implements a 5-deep stack, so 5 nested calls can be made without overflowing the stack. NCALL is defined as a MACRO which will be used instead of the mnemonic CALL, when a subroutine call is made. The NCALL routine, “pushes” the return PC value on the “stack” and then executes the called subroutine. At the end of the subroutine, instead of using the RETLW k instruction, a GOTO RETURN is executed, where RETURN is a routine which “pops” the return PC value from the “stack” and resumes the normal flow of the program.

Note: Since Software Stack Management utilizes the FSR register, and indirect addressing, the user should restore the "original" values to the FSR register if it is utilized elsewhere in the program.

The routines, as described in this application note, will work only if the called routine is within the first 256 words for each program. If the user desires to branch over to the other low 256-byte program pages, as in the PIC16C57, then the status byte should be saved along with the PC.

Author: Stanley D'Souza
Logic Products Division

Software Stack Management

```
MPASM 1.00 Released        SM.ASM   7-15-1994  13:9:35                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         list p=16c54,f=inhx8m
                     0002 ;*************************************************
                     0003 ;       sm.asm:
                     0004 ;       Routine, demonstrating how to implement a stack
                     0005 ;       manager capable of handling more than 2
                     0006 ;       subsequent subroutine calls.
                     0007 ;       Note: Since this is a demo, NOP has been used
                     0008 ;       where normally the body of the subroutine would
                     0009 ;       reside.
                     0010 ;*************************************************
                     0011 ;
0002                 0012         PC      EQU     2
0004                 0013 FSR     EQU     4
                     0014 ;
                     0015 ;
0008                 0016 STACK   EQU     8       ;define stack top
                     0017 ;*************************************************************
                     0018 ;NOTE: the next 5 locations in RAM should be reserved for the
                     0019 ;"STACK" implementation. Please do not use any ram locations
                     0020 ;from decimal 8 to decimal 12.
                     0021 ;*************************************************************
                     0022 ;
                     0023         ORG     01FF
01FF 0A07            0024         GOTO    START
                     0025 ;
                     0026         ORG     0
                     0027 ;
0000 0C08            0028 INIT    MOVLW   STACK   ;load "stack" as indirect pointer
0001 0024            0029         MOVWF   FSR     ;       /
0002 0A07            0030         GOTO    START   ;      /
                     0031 ;
                     0032 ;*************************************************
                     0033 ;define NCALL as a MACRO used instead of the
                     0034 ;mnemonic CALL.
                     0035 ;
                     0036 NCALL   MACRO   LABEL
                     0037         MOVF    PC,W    ;save PC on "stack"
                     0038         MOVWF   0       ;       /
                     0039         INCF    FSR     ;Inc. "stack"  pointer.
                     0040         GOTO    LABEL   ;jump to routine
                     0041         ENDM
                     0042 ;
                     0043 ;return from subroutine NCALL
                     0044 ;
0003 00E4            0045 RETURN  DECF    FSR     ;point to last "stack" location
0004 0C03            0046         MOVLW   3       ;add 3 and output value from FSR
0005 01C0            0047         ADDWF   0,W     ;       /
0006 0022            0048         MOVWF   PC      ;load in PC as next executable
                     0049 ;                       instruction
                     0050 ;
                     0051 ;*************************************************
                     0052 ;
```

```
MPASM 1.00 Released        SM.ASM   7-15-1994  13:9:35               PAGE  2

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0054 ;
0007 0000            0055 START   NOP
                     0056         NCALL   TOM
0008 0202               M         MOVF    PC,W    ;save PC on “stack”
0009 0020               M         MOVWF   0       ;       /
000A 02A4               M         INCF    FSR     ;Inc. “stack”  pointer.
000B 0A0F               M         GOTO    TOM   ;jump to routine
000C 0000            0057         NOP     ;body of main routine
000D 0000            0058         NOP     ;       /
000E 0003            0059         SLEEP
                     0060 ;
000F 0000            0061 TOM     NOP
                     0062         NCALL   DICK
0010 0202               M         MOVF    PC,W    ;save PC on “stack”
0011 0020               M         MOVWF   0       ;       /
0012 02A4               M         INCF    FSR     ;Inc. “stack”  pointer.
0013 0A16               M         GOTO    DICK   ;jump to routine
0014 0000            0063         NOP     ;body of routine TOM
0015 0A03            0064         GOTO    RETURN
                     0065 ;
0016 0000            0066 DICK    NOP
                     0067         NCALL   HARRY
0017 0202               M         MOVF    PC,W    ;save PC on “stack”
0018 0020               M         MOVWF   0       ;       /
0019 02A4               M         INCF    FSR     ;Inc. “stack”  pointer.
001A 0A1D               M         GOTO    HARRY   ;jump to routine
001B 0000            0068         NOP     ;body of routine DICK
001C 0A03            0069         GOTO    RETURN
                     0070 ;
001D 0000            0071 HARRY   NOP     ;body of routine HARRY
001E 0000            0072         NOP     ;       /
001F 0A03            0073         GOTO    RETURN
                     0074 ;
                     0075 ;
                     0076         END
                     0077
                     0078
```

```
MPASM 1.00 Released        SM.ASM   7-15-1994  13:9:35               PAGE  3


SYMBOL TABLE

LABEL                             VALUE

DICK                               0016
FSR                                0004
HARRY                              001D
INIT                               0000
PC                                 0002
RETURN                             0003
STACK                              0008
START                              0007
TOM                                000F


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ________ ________
0040 : ________ ________ ________ ________

0180 : ________ ________ ________ ________
01C0 : ________ ________ ________ _______X

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

AN581

Implementing Long Calls

INTRODUCTION

This application note discusses how to implement "long Calls" in the PIC16C5X architecture. The use of long call can simplify the partitioning of the application program with minimal software overhead.

In the PIC16C5X architecture, the program memory page size is 512 words. Depending on the device, the program memory may be as large as 2K words (as in the PIC16C57 or PIC16C58 devices). The program counter (and stack) width range from 9- to 11-bits, depending on the amount of program memory the device has. Table 1 shows the width of the Program Counter (PC) and Stack for the various devices.

TABLE 1: PC AND STACK WIDTH

Device	Width (Bits)		Program Memory (Words)
	Program Counter	Stack	
PIC16C54 / PIC16C55	9	9	512
PIC16C56	10	10	1K
PIC16C57 / PIC16C58	11	11	2K

The low order 8-bits of the program counter are accessible by the user program. These bits are contained in the PC register. The entire Program Counter is shown in Figure 1.

Since A8 is forced to 0 by CALL instructions, the start address of subroutines must be in the first 256 words of each program memory page. Depending on the size and number of called subroutines, this limitation may become a burden to the software developer. The implementation of a "long call" eases this, by allowing the subroutine to be anywhere in the program memory page. The three important concepts, to understand the implementation of the long call are:

1. A CALL instruction loads the entire PC onto the Stack
2. A GOTO instruction does not affect the Stack
3. A GOTO instruction can branch to any location in a program memory page.

Also to select the desired page, the RP1 and RP0 bits (STATUS<6:5>) must be programmed accordingly. These bits do not get loaded into A10:A9 , of the PC, until one of the following occurs:

1. A CALL instruction
2. A GOTO instruction
3. An instruction that modifies the PC register (PC<:7:0>), such as ADDWF PC, F.

So a CALL instruction followed by a GOTO instruction will always remain in the same page as the intended call. This allows the developer to place "call vectors" at the first 256 words of each page. The instruction at the "call vector" then executes a GOTO instruction to the subroutine anywhere in that page. The RETLW instruction, of the subroutine, will then POP the stack. The Stack contained the PUSHed PC from the CALL instruction.

Figure 2 shows an example of a "long call" sequence in a device with 2K-words.

FIGURE 1: PROGRAM COUNTER STRUCTURE

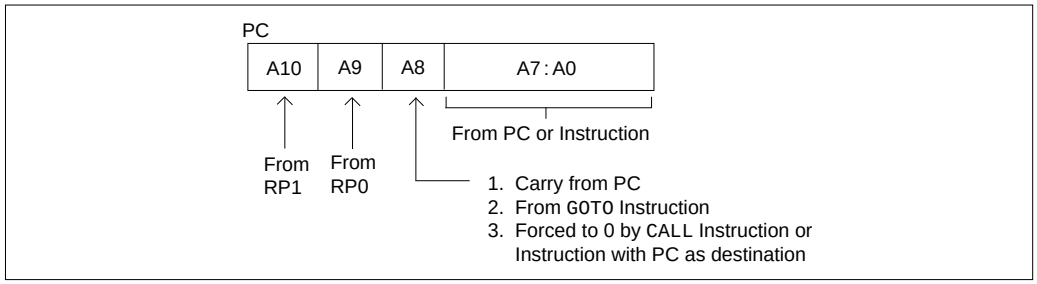
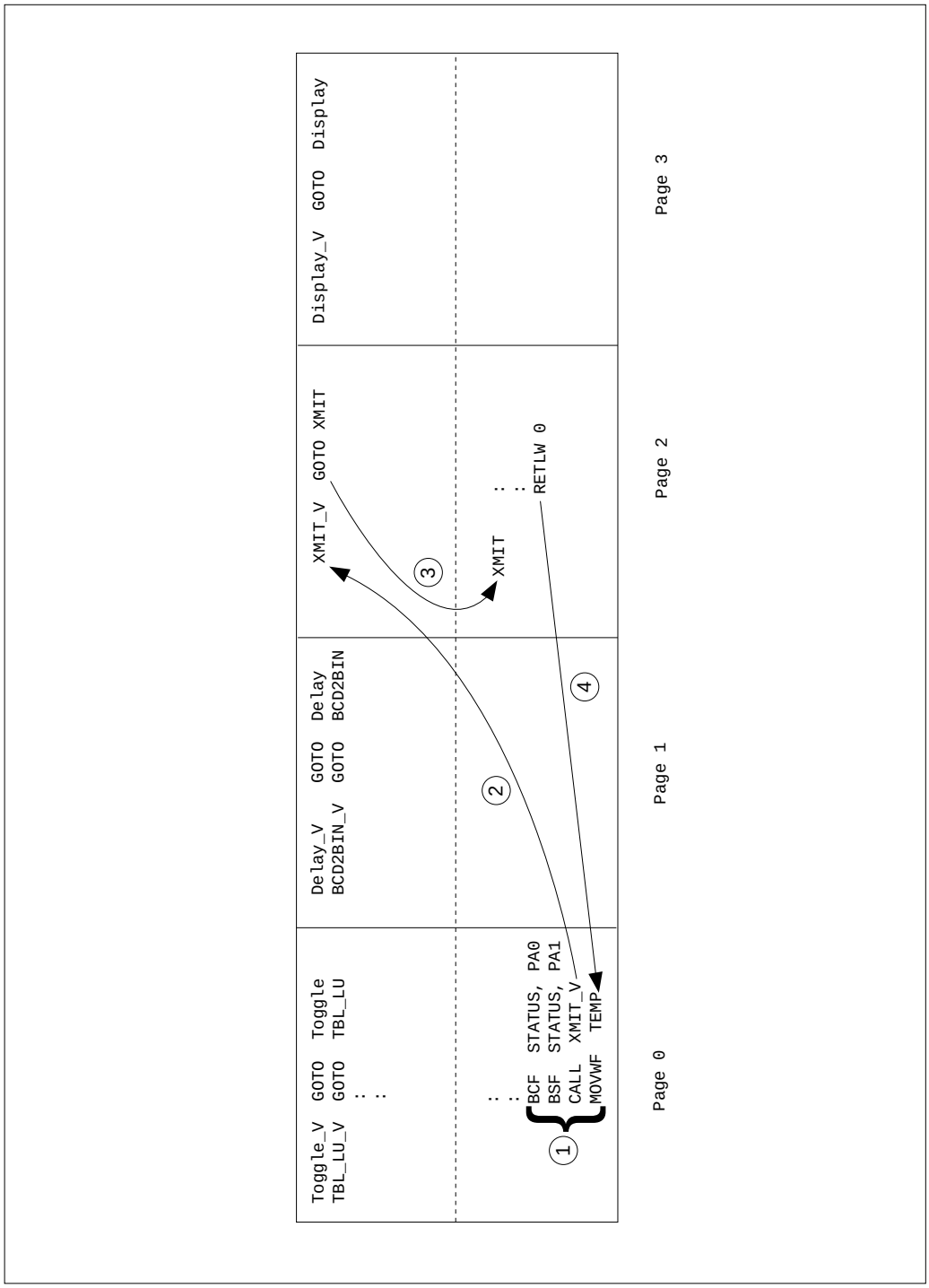


FIGURE 2: EXAMPLE OF A “LONG CALL” SEQUENCE



The flow that occurs in Figure 2 is as follows:

1. Select the program memory page of the desired subroutine and execute the call to that subroutine.
2. The program loads the Stack with the PC+1 address, branches to the selected page and specified address of the “call vector” (must be in the first 256 locations of the page)
3. Executes a GOTO instruction, to have access to the entire program memory page. Then executes the subroutine.
4. Executes the RETLW instruction, which POPs the new PC from the Stack. This causes program execution to continue at the instruction after the CALL instruction.

The use of “long calls” could be used to place all the subroutines in selected page(s), since the entire page can contain the subroutines (not restricted to the top half of the page). The placing of all subroutines in fewer program memory pages can reduce the overhead of specifying the required pages, since they are changed less frequently.

Use of the MPASM assembler can ease in the verification that call vectors and the call routine are in the same program memory page. Example 1 shows the use of assembler directives to print user defined warning or error messages in the listing file. These are shown as the shaded conditional statements. These messages are only printed in the listing file, and no indication of these messages is shown at the completion of assembly.

EXAMPLE 1: USE OF ASSEMBLER DIRECTIVES

```
;
P1_TOP                  EQU             0x0000          ; First address in page 0
P2_TOP                  EQU             0x0200          ; First address in page 1
P3_TOP                  EQU             0x0400          ; First address in page 2
P4_TOP                  EQU             0x0600          ; Reset vector address in page 1
RESET_V                 EQU             0x07FF
;
     org    P1_TOP
;
     :
     :
     :
;
     org    P3_TOP
;
My_Subroutine_V     GOTO    My_Subroutine    ; Vector for My_Subroutine
     :
     :
     :
;
My_Subroutine                                ; My_Subroutine routine
;
    if ( ( My_Subroutine_V & 0x0600 ) != ( My_Subroutine & 0x0600 ) )
       MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
    endif
;
     :
     :
     :
;
My_Subroutine_END    RETURN
;
    if ( ( My_Subroutine_V & 0x0600) != (My_Subroutine_END & 0x0600) )
       MESSG   "Warning - User Defined: Call routine crosses page boundry"
    endif
;
     :
     :
     :
;
       org   RESET_V                         ; Program memory address for the reset vector
;
              GOTO           START          ; Goto the beginning of the program
```

CONCLUSION

The use of "long calls" may ease the development of application programs. For minimal overhead, the application program can execute a subroutine from anywhere in the program memory, and return to the desired location. This eases the development of the application program, by reducing the mapping of subroutine in the first 256 words of each program memory page. The use of "long calls" is possible in any of the PIC16C5X devices, but is most useful in the devices with more than one program memory page. For device with more than one page of program memory, the assembler directives can be used to verify that the subroutines are in the program memory page.

Author: Mark Palmer - Sr. Application Engineer

AN586

Macros for Page and Bank Switching

INTRODUCTION

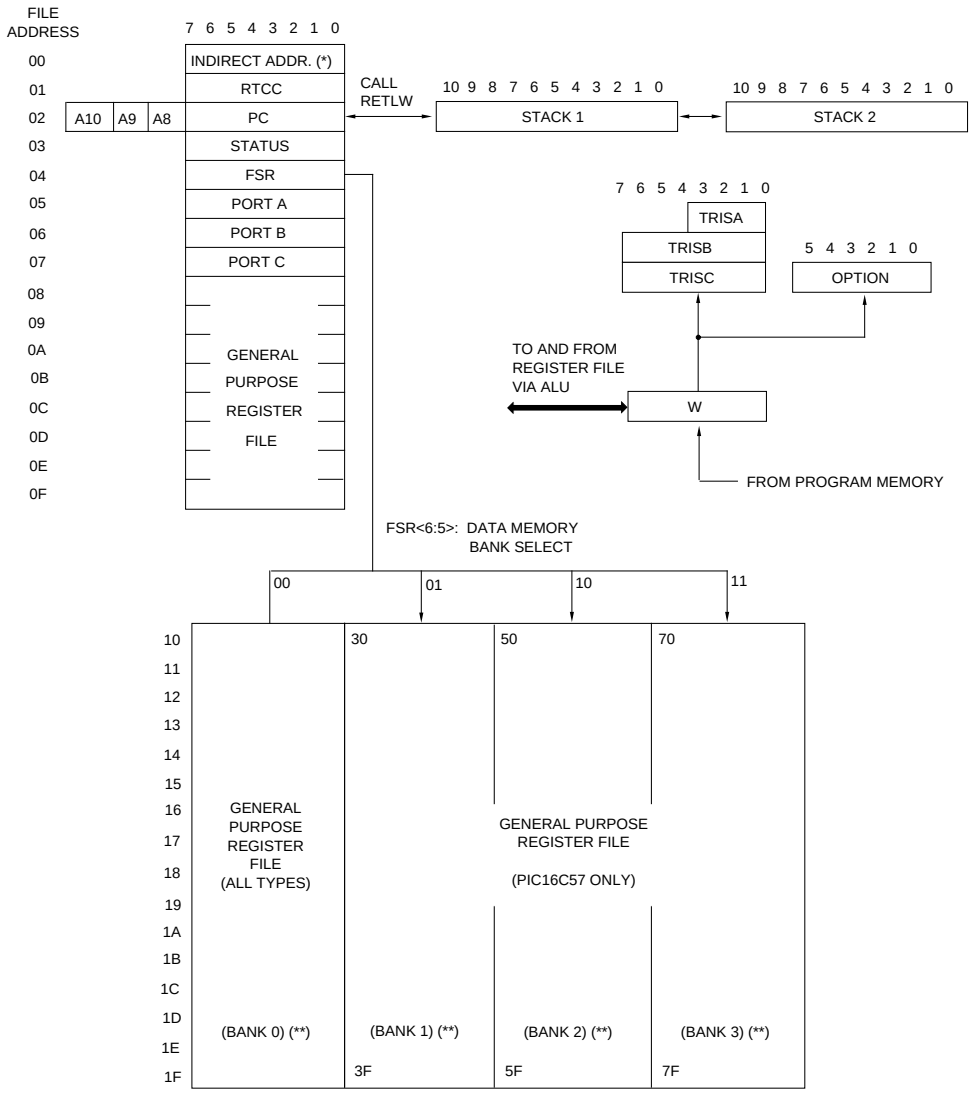
This application note discusses the use of the MPASM assembler's conditional assembly to automatically switch between program memory pages or to set the data memory banks. These macros, along with the long call technique (see Application Note AN581), ease the development of software. Though the use of these macros can simplify the program memory paging and data memory banking with minimal software overhead. The use of these macros without thought can causes unnecessary (duplicate) instructions to be used, by setting page or bank bits unnecessarily.

The PIC16C5X family of devices has an architecture where the program memory has up to four pages of program memory (512 words / page) and four banks of data memory (16 bytes / bank). Two bits in the STATUS register, PA1 and PA0, are used to manage the program memory page. Two bits of the FSR register, bits 6 and 5, manage the data memory bank. We will call the FSR<5> bit RP0 and the FSR<6> bit RP1 (for Register Page 0 and 1). The naming of these bits RP1 and RP0 should not be confused with the similarly named bits in the PIC16CXX family (PIC16C64, PIC16C71, etc.). The RP bits for the PIC16CXX family are found in the STATUS register, as opposed to the FSR register for the PIC16C5X family. The use of these macros can be modified to support the PIC16CXX family.

The program memory organization is shown in Figure 1 and the data memory organization is shown in Figure 2. To use the macros for the data memory, the data memory locations must be EQUated for the absolute address, and not the relative address in the bank. The relative address is the lower 5-bits of the data memory address.

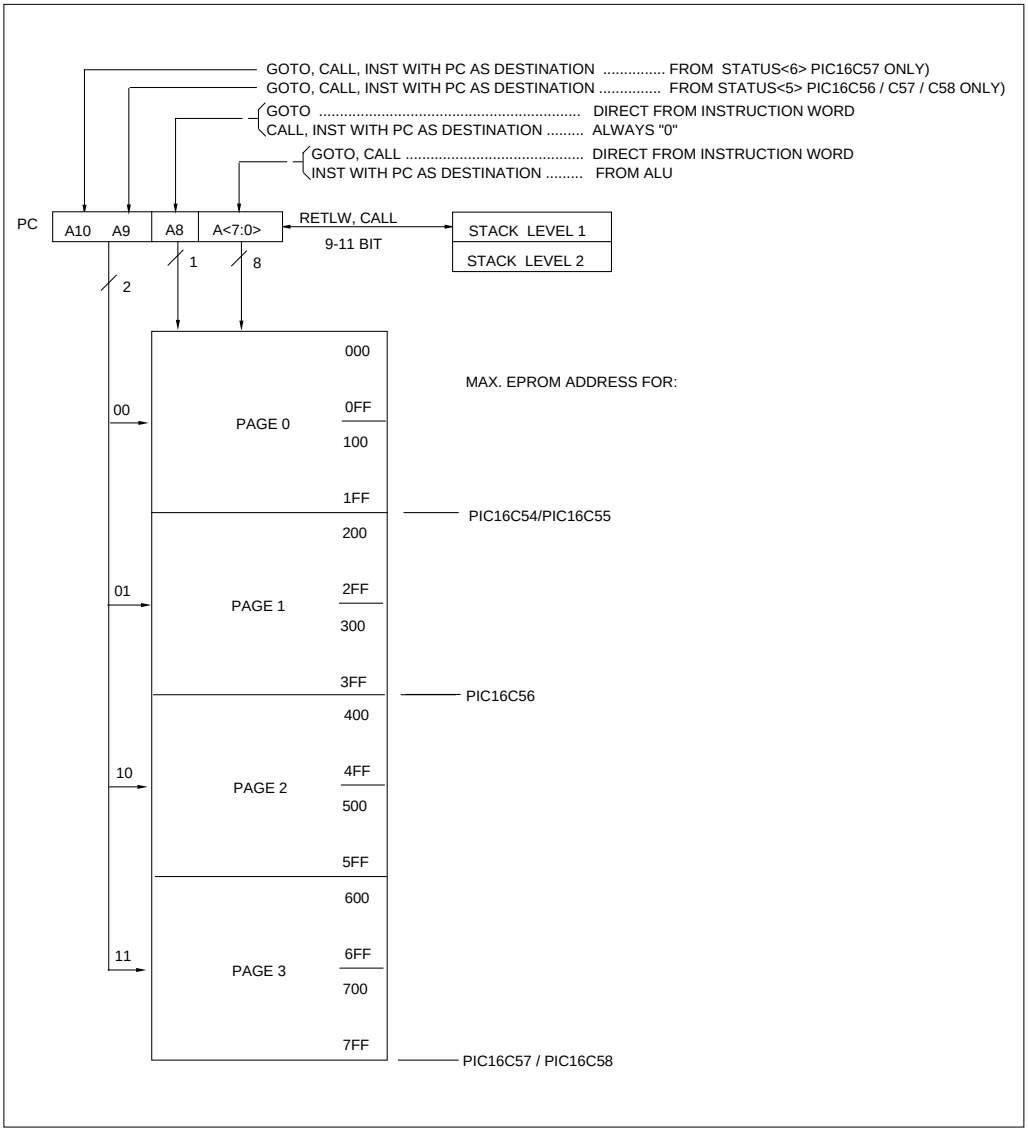
When the address of the data memory has the MSb (bit 4) of the direct address is cleared, or FSR<4> cleared (for indirect addressing), the address 0h through 0Fh is accessed. That is when accessing addresses 0h through 0Fh, the bank selection (FSR<6:5>) bits are ignored. This means that data memory addresses 'xxx0 xxxx'b access the data memory address 0xh (x is 0 - Fh).

FIGURE 1: PROGRAM MEMORY ORGANIZATION



(*) NOT A PHYSICALLY IMPLEMENTED REGISTER. SEE SECTION 4.0 OR DETAILS
(**) BANK 0 IS AVAILABLE ON ALL MICROCONTROLLERS WHILE BANK 1 TO BANK 3 ARE ONLY AVAILABLE ON THE PIC16C57 AND PIC16C58.

FIGURE 2: DATA MEMORY MAP



Macros for Page and Bank Switching

The use of MPASM's conditional assembly, allows the selection of source code to be assembled based on the address of the symbol / label. The Macros supplied are show in Table 1. They can be grouped into three categories:

1. Configuring of the program memory pages
2. Configuring of the data memory banks
3. Other

TABLE 1: MACROS

Program Calling Paging	Operands	Operation
CALLM	address	Sets page bits, then CALLs the specified routine
GOTOM	address	Sets page bits, then GOTOs the specified address
PAGE_MAC	address	Sets the specified page bits
Data Memory Banking		
ADDWF_MAC	Reg, dest	Sets Bank bits, then executes the ADDWF
ANDWF_MAC	Reg, dest	Sets Bank bits, then executes the ANDWF
BCF_MAC	Reg, bit	Sets Bank bits, then executes the BCF
BSF_MAC	Reg, bit	Sets Bank bits, then executes the BSF
BTFSC_MAC	Reg, bit	Sets Bank bits, then executes the BTFSC
BTFSS_MAC	Reg, bit	Sets Bank bits, then executes the BTFSS
CLRF_MAC	Reg	Sets Bank bits, then executes the CLRF
COMF_MAC	Reg, dest	Sets Bank bits, then executes the COMF
DECF_MAC	Reg, dest	Sets Bank bits, then executes the DECF
DECFSZ_MAC	Reg, dest	Sets Bank bits, then executes the DECFSZ
INCF_MAC	Reg, dest	Sets Bank bits, then executes the INCF
INCFSZ_MAC	Reg, dest	Sets Bank bits, then executes the INCFSZ
IORWF_MAC	Reg, dest	Sets Bank bits, then executes the IORWF
MOVF_MAC	Reg, dest	Sets Bank bits, then executes the MOVF
MOVWF_MAC	Reg	Sets Bank bits, then executes the MOVWF
RLF_MAC	Reg, dest	Sets Bank bits, then executes the RLF
RRF_MAC	Reg, dest	Sets Bank bits, then executes the RRF
SUBWF_MAC	Reg, dest	Sets Bank bits, then executes the SUBWF
SWAPF_MAC	Reg, dest	Sets Bank bits, then executes the SWAPF
XORWF_MAC	Reg, dest	Sets Bank bits, then executes the XORWF
BANK_MAC	Reg	Sets the specified Bank bits
Other		
SAVE_W_STATUS	-	Saves the W and STATUS registers
RESTORE_W_STATUS	-	Restores the W and STATUS registers

These macros (see Appendix A) ease the development of programs, but care should be taken in their use so that redundant instructions are not caused. An example of this is if you wanted to do the operations, INCF and BTFSS, on data memory location CNTR (in bank 3) and the FSR was pointing to some other bank. The use of the macros for both operations would cause six program memory locations to be assembled, while with some thought only four words are needed (see Example 1).

CONCLUSION

The use of these macros simplify the program development by managing the memory resources of the PIC16C5X device. If the application program becomes too large for the desired device program memory, it is recommended to study the listing file for any unnecessary code due to non-optimum usage of these macros. The MAC_TST.ASM file, is supplied to show how these macros work in a program.

EXAMPLE 1A: GENERATION OF UNNECESSARY CODE

```
INCF_MAC        CNTR, F         —>      BSF     FSR, 5
                                        BSF     FSR, 6
                                        INCF    CNTR, F
BTFSS_MAC       CNTR, 5         —>      BSF     FSR, 5          ; Unnecessary, already in bank
                                        BSF     FSR, 6          ; Unnecessary, already in bank
                                        BTFSS   CNTR, 5
```

EXAMPLE 1B: GENERATION OF OPTIMUM CODE

```
INCF_MAC        CNTR, F         —>      BSF     FSR, 5
                                        BSF     FSR, 6
                                        INCF    CNTR, F
BTFSS           CNTR, 5         —>      BTFSS   CNTR, 5
```

Written By: Mark Palmer - Sr. Application Engineer
Contributions by: Mike Morse - Sr. Field Application Engineer (Dallas)

APPENDIX A: MACRO FILE

```
   nolist
;*************************************************************************
; This file contains MACROs to ease in the use of the Program Memory
; paging and the Data Memory bank switching for the PIC16C5x devices
;
;    File Name:   AUTO_PG.MAC
;    REVISION:    5-20-94
;*************************************************************************
;
;
PAGE1_OR_3   EQU       0x0200         ; Program Memory in page 1 or page 3
PAGE2_OR_3   EQU       0x0400         ; Program Memory in page 2 or page 3
;
BANK1_OR_3   EQU       0x020          ; Data Memory in Bank 1 or Bank 3
BANK2_OR_3   EQU       0x040          ; Data Memory in Bank 2 or Bank 3
;
;*************************************************************************
;**                            CALLM     program_address
;** Configures the PA1 and PA0 bits as required, ensures that the CALLed
;** “routine” is in the first 256 locations of the program memory page.
;** If the “routine” is in the second 256 locations of the program memory page,
;** an User Defined ERROR Message is placed in the LISTING file. MPASM
;** presently only places this message in the listing file (i.e. no indication
;** is shown when MPASM completes execution in the ERROR / WARNING listed.
;**
;*************************************************************************
;
CALLM          macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
     if ( ( routine & 0x0100 ) == 0x0100 )
       MESSG “Error - User Defined: CALLed routine in 2nd 256 locations of the
              program memory page”
     endif
;
               CALL    routine
               endm
```

```
;
;*****************************************************************************
;**                              GOTOM     program_address
;** Configures the PA1 and PA0 bits as required, and GOTOs the specified
;** locations of the program memory page.
;**
;*****************************************************************************
;
GOTOM          macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
               GOTO    routine
               endm
;
;*****************************************************************************
;**                   ADDWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “ADDWF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
ADDWF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               ADDWF    address, d
               endm
```

```
;
;*****************************************************************************
;**                   ANDWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “ANDWF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
ANDWF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               ANDWF    address, d
               endm
;
;*****************************************************************************
;**                   BCF_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “BCF data_address, bit” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
BCF_MAC        macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BCF     address, b
               endm
```

```
;
;*******************************************************************************
;**                   BSF_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “BSF data_address, bit” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
BSF_MAC        macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BSF     address, b
               endm
;
;*******************************************************************************
;**                   BTFSC_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “BTFSC data_address, bit” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*******************************************************************************
;
BTFSC_MAC      macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BTFSC     address, b
               endm
```

Macros for Page and Bank Switching

```
;
;*****************************************************************************
;**                   BTFSS_MAC     data_address, bit
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “BTFSS data_address, bit” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
BTFSS_MAC      macro       address, b
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               BTFSS    address, b
               endm
;
;*****************************************************************************
;**                   CLRF_MAC     data_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “CLRF data_address” instruction. The data_address
;** must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
CLRF_MAC       macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               CLRF    address
               endm
```

```
;
;*****************************************************************************
;**                   COMF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “COMF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
COMF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               COMF    address, d
               endm
;
;*****************************************************************************
;**                   DECF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “DECF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
DECF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               DECF    address, d
               endm
```

Macros for Page and Bank Switching

```
;
;*****************************************************************************
;**                   DECFSZ_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “DECFSZ data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
DECFSZ_MAC     macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               DECFSZ  address, d
               endm
;
;*****************************************************************************
;**                   INCF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “INCF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
INCF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               INCF    address, d
               endm
```

```
;
;*****************************************************************************
;**                    INCFSZ_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "INCFSZ data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
INCFSZ_MAC     macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               INCFSZ  address, d
               endm
;
;*****************************************************************************
;**                    IORWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "IORWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
IORWF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               IORWF    address, d
               endm
```

Macros for Page and Bank Switching

```
;
;*****************************************************************************
;**                   MOVF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “MOVF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
MOVF_MAC       macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               MOVF    address, d
               endm
;
;*****************************************************************************
;**                   MOVWF_MAC     data_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “BSF data_address” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
MOVWF_MAC      macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               MOVWF    address
               endm
```

```
;
;*****************************************************************************
;**                   RLF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “RLF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
RLF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               RLF     address, d
               endm
;
;*****************************************************************************
;**                   RRF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “RRF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
RRF_MAC        macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               RRF     address, d
               endm
```

```
;
;*****************************************************************************
;**                   SUBWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “SUBWF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
SUBWF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               SUBWF   address, d
               endm
;
;*****************************************************************************
;**                   SWAPF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the “SWAPF data_address, destination” instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
SWAPF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               SWAPF   address, d
               endm
```

```
;
;*****************************************************************************
;**                   XORWF_MAC     data_address, destination
;** Configures the FSR<6:5> bits as required for the Data Memory addressing
;** and then executes the "XORWF data_address, destination" instruction. The
;** data_address must be the absolute address and NOT the relative address in
;** the data memory page.
;**
;*****************************************************************************
;
XORWF_MAC      macro       address, d
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               XORWF   address, d
               endm
;
;*****************************************************************************
;**                   PAGE_MAC     program_address
;** Configures the PA1 and PA0 bits as required
;**
;*****************************************************************************
;
PAGE_MAC          macro       routine
;
     if ( ( routine & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
     else
               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
     endif
;
     if ( ( routine & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
     else
               BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
     endif
;
               endm
```

Macros for Page and Bank Switching

```
;
;*****************************************************************************
;**                   BANK_MAC     program_address
;** Configures the FSR<6:5> bits as required for the Data Memory addressing.
;** The data_address must be the absolute address and NOT the relative address
;** in the data memory page.
;**
;*****************************************************************************
;
BANK_MAC      macro       address
;
     if ( ( address & PAGE1_OR_3 ) == PAGE1_OR_3 )
               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
     else
               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
     endif
;
     if ( ( address & PAGE2_OR_3 ) == PAGE2_OR_3 )
               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
     else
               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
     endif
               endm
;
;*****************************************************************************
;**                   SAVE_W_AND_STATUS
;** Saves the contects of the W register and the STATUS register to two
;** temporary RAM locations W_TEMP and STATUS_TEMP. These temporary RAM
;** locations should be in the NON-Banked part of Data Memory (8h to Fh).
;** This Macro generates a User Defined Warning (seen only in listing file)
;** if the Data RAM location is in Banked RAM.
;**
;*****************************************************************************
;
SAVE_W_AND_STATUS      macro
;
               MOVWF   W_TEMP
               SWAPF   W_TEMP, F
               SWAPF   STATUS, W
               MOVWF   STATUS_TEMP
     if ( ( W_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: W_TEMP register is defined to be in BANKed
             memory"
     endif
;
     if ( ( STATUS_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: STATUS_TEMP register is defined to be in BANKed
             memory"
     endif
               endm
```

```
;
;*************************************************************************
;**                   RESTORE_W_AND_STATUS
;** Saves the contects of the W register and the STATUS register to two
;** temporary RAM locations W_TEMP and STATUS_TEMP. These temporary RAM
;** locations should be in the NON-Banked part of Data Memory (8h to Fh).
;** This Macro generates a User Defined Warning (seen only in listing file)
;** if the Data RAM location is in Banked RAM.
;**
;*************************************************************************
;
RESTORE_W_AND_STATUS      macro
;
               SWAPF   STATUS_TEMP, W
               MOVWF   STATUS
               SWAPF   W_TEMP, W
     if ( ( W_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: W_TEMP register is defined to be in BANKed
             memory"
     endif
;
     if ( ( STATUS_TEMP & 0x0F0 ) != 0x00 )
       MESSG "Warning - User Defined: STATUS_TEMP register is defined to be in BANKed
             memory"
     endif
               endm
;

     list
```

NOTES:

AN528

Implementing Wake-Up on Key Stroke

2

INTRODUCTION

In certain applications, the PIC16CXX is exercised only when a key is pressed, eg. remote keyless entry. In such applications, the battery life can be extended by putting the PIC16CXX to sleep during the inactive state and when a key is pressed, the PIC16CXX wakes up, does the task, and then goes back to sleep.

IMPLEMENTATION

The circuit in Figure 1 depicts an application with two keys. The PIC16C54 is normally in SLEEP mode consuming very little operating current. If either of the two keys is pressed, the PIC16C5X 'wakes up', scans the keys and turns on one of the two LED's. When SW1 is pressed, the green LED is turned on and when SW2 is pressed the red LED is turned on. The LED's are used purely for demonstration purposes. In real life application, a transmission will be completed before putting the PIC16C5X back in sleep.This example can be extended to handle more than two keys.

In the sleep mode, the scan outputs (SCAN1 and SCAN2) are both set to a low logic level. In this state, the capacitor C is fully charged and a high logic level is present at the MCLR pin of the PIC16C5X. When a key is pressed, C discharges through either R2 or R3 (depending on SW1 or SW2 being pressed) and the voltage across C falls rapidly (approx. 1 ms), causing a low at the MCLR pin of the PIC16C5X, which in turn causes the PIC16C5X to wake up and enter its reset state. In reset, the SCAN1 and SCAN2 outputs default to a high impedance mode, so the discharge path for capacitor C is blocked and it charges to a high level through resistor R1. Note that the RC values have been chosen such, that the discharge and charge cycles times are less than the reset time for the PIC16C5X (approx. 18 ms), and certainly far less than the minimum duration of a keypress (approx. 50-100 ms).

After the reset cycle is completed, the code execution momentarily takes the SCAN1 and SCAN2 outputs low in order to sample the key stroke(s). This does not cause the capacitor to discharge since the duration of the low is of the order of 10 micro seconds.

Once the keystroke function has been executed, the program loops until the key has been released, sets the SCAN1 and SCAN2 outputs low and "goes back to sleep". Resistors R4-R8 are not required for functionality, but are recommended to provide protection from electrostatic discharge (ESD). Switches SW1 and SW2, when pressed may frequently pass ESD to the PIC16C54.

Author: Stan D'Souza
Logic Products Division

FIGURE 1 - TWO KEY INTERFACE TO PIC16C5X

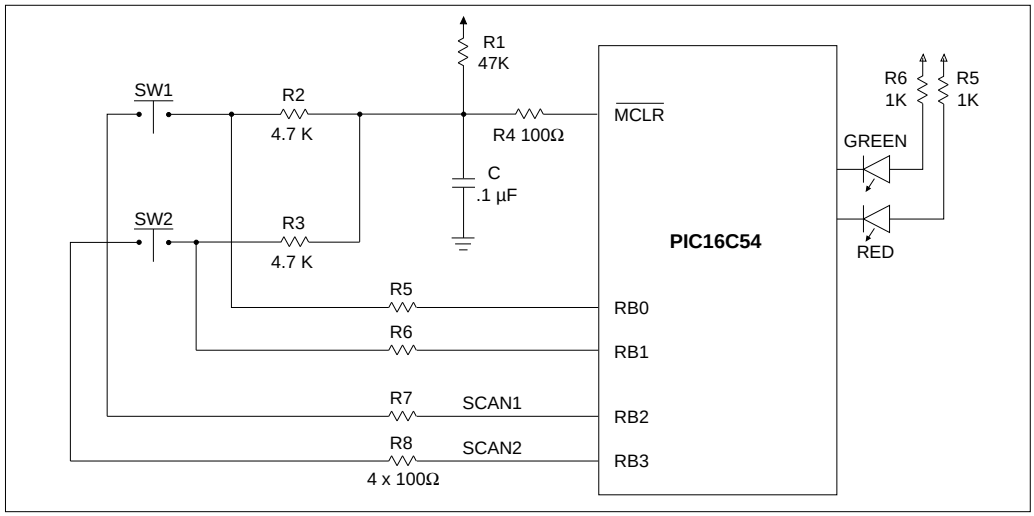


FIGURE 2 - TWO KEY SCAN/WAKE-UP TIMING DIAGRAM

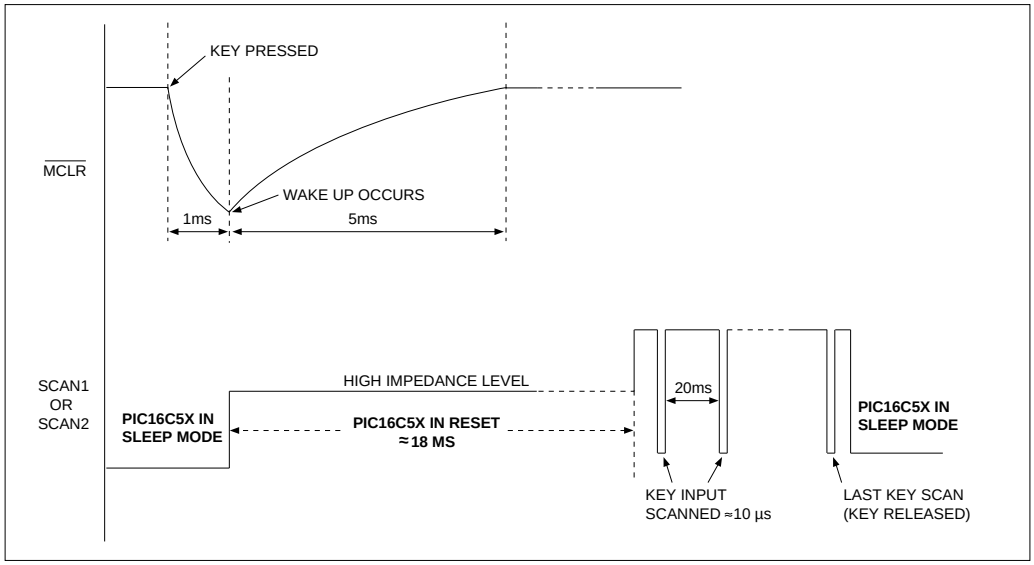
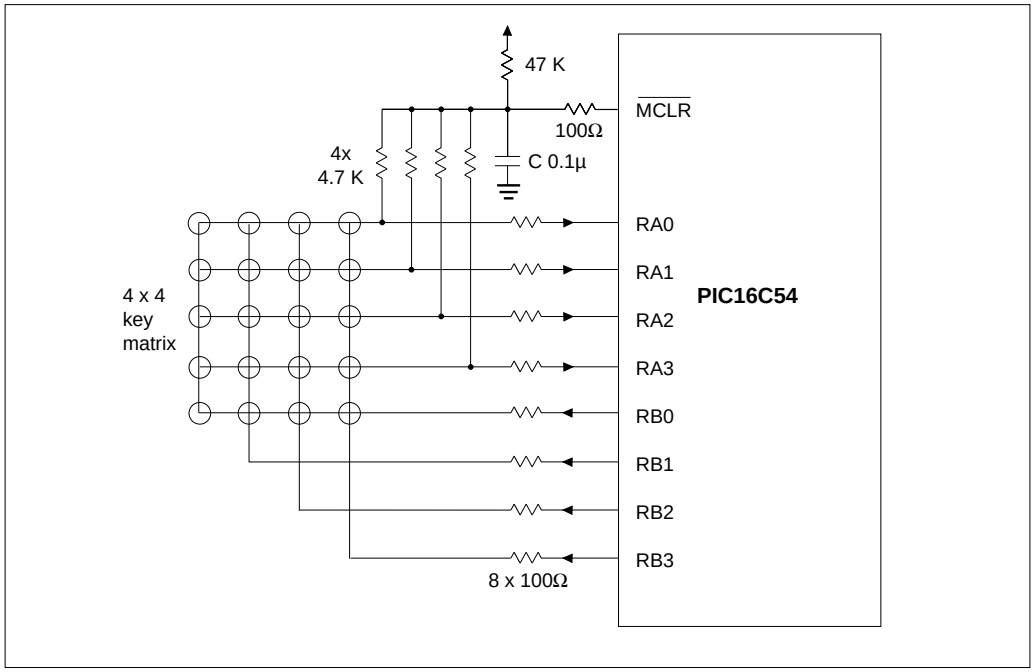


FIGURE 3 - PIC16C5X INTERFACE TO 4 X 4 KEY MATRIX



```
MPASM 1.00 Released        WU.ASM   7-15-1994  13:10:54                PAGE  1
Key Stroke Wake Up

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001                 TITLE        “Key Stroke Wake Up”
                     0002          LIST  P = 16C54,f=inhx8m
                     0003 ;*************************************************
                     0004 ;       Program demonstrating key stroke wake up for
                     0005 ;       the PIC16CXX. Program has been implemented for
                     0006 ;       two keys, but can be extended for more keys.
                     0007 ;       When SW1 is pressed a green LED lights up.
                     0008 ;       When SW2 is pressed a red LED lights up.
                     0009 ;*************************************************
                     0010 ;
                     0011 ; Define equates
                     0012 ;
0002                 0013          PC       EQU    2
0006                 0014          PORT_B   EQU    6
0002                 0015          SCAN1    EQU    2
0003                 0016          SCAN2    EQU    3
0000                 0017          SW1      EQU    0
0001                 0018          SW2      EQU    1
0004                 0019          GRN_LED  EQU    4
0005                 0020          RED_LED  EQU    5
0014                 0021          MSEC_20  EQU    D’20'
0008                 0022          DB1      EQU    8
0008                 0023          GP       EQU    8
0009                 0024          DB2      EQU    9
                     0025 ;
                     0026 ;PORT_B ASSIGNMENTS:
                     0027 ;       0 –> SW1       INPUT
                     0028 ;       1 –> SW2       INPUT
                     0029 ;       2 –> SCAN1     OUTPUT
                     0030 ;       3 –> SCAN2     OUTPUT
                     0031 ;       4 –> GRN_LED   OUTPUT
                     0032 ;       5 –> RED_LED   OUTPUT
                     0033 ;     6&7 –> ASSIGNED AS DUMMY OUTPUTS
                     0035 ;
                     0036 ;
                     0037          ORG      0
                     0038 ;
                     0039 START
0000 0910            0040          CALL     INIT_PORT_B     ;INITIALIZE PORT B
0001 0920            0041          CALL     DELAY           ;DELAY 20 MSECS
0002 0915            0042          CALL     SCAN_KEYS       ;GET KEY VALUES
0003 0028            0043          MOVWF    GP              ;SAVE IN RAM
0004 0608            0044          BTFSC    GP,SW1          ;SKIP IF SW1 NOT PRESSED
0005 0929            0045          CALL     TURN_GREEN_ON   ;ELSE DO ROUTINE
0006 0628            0046          BTFSC    GP,SW2          ;SKIP IF SW2 NOT PRESSED
0007 092B            0047          CALL     TURN_RED_ON     ;ELSE DO ROUTINE
                     0048 CHK_FOR_KEY
0008 0920            0049          CALL     DELAY           ;DELAY FOR 20 MSEC
0009 0915            0050          CALL     SCAN_KEYS       ;GET KEY HIT
000A 0F00            0051          XORLW    0               ;EXCL. OR WITH 0
000B 0743 0A08       0052          BNZ      CHK_FOR_KEY     ;KEY STILL PRESSED
                     0053                                   ;THEN LOOP
                     0054 NO_KEY_PRESSED
000D 0446            0055          BCF      PORT_B,SCAN1    ;SET SCAN LINES LOW
000E 0466            0056          BCF      PORT_B,SCAN2    ;       /
000F 0003            0057          SLEEP                    ;SLEEP
                     0058 ;
                     0060 ;
                     0061 INIT_PORT_B
0010 0C03            0062          MOVLW    B’00000011'     ; config RB0, 1 as i/p’s
0011 0006            0063          TRIS     PORT_B          ;  and RB2-7 as o/p’s
0012 0CFF            0064          MOVLW    0FFh
0013 0026            0065          MOVWF    PORT_B          ;DEFAULT VALUES FOR PORT_B
0014 0800            0066          RETLW    0               ;RETURN WITH NO ERROR
```

```
                    0067 ;
                    0068 ;This routine, scans two keys and returns the following:
                    0069 ;       0 if no key is pressed
                    0070 ;       1 if SW1 is pressd
                    0071 ;       2 if SW2 is pressed
                    0072 ;       3 if SW1 and SW2 are pressed
                    0073 ;
                    0074 SCAN_KEYS
0015 0446           0075         BCF     PORT_B,SCAN1    ;ENABLE SCAN FOR SW1
0016 0466           0076         BCF     PORT_B,SCAN2    ;EANBLE SCAN FOR SW2
0017 0C03           0077         MOVLW   B'00000011'     ;LOAD MASK IN W
0018 0146           0078         ANDWF   PORT_B,0        ;AND WITH PORT
0019 0546           0079         BSF     PORT_B,SCAN1    ;DISABLE SCAN
001A 0566           0080         BSF     PORT_B,SCAN2    ;       /
001B 01E2           0081         ADDWF   PC,1            ;GET OFFSET TO TABLE
001C 0803           0082         RETLW   3               ;SW1 AND SW2 PRESSED
001D 0802           0083         RETLW   2               ;SW2 PRESSED
001E 0801           0084         RETLW   1               ;SW1 PRESSED
001F 0800           0085         RETLW   0               ;NO KEY PRESSED
                    0086 ;
                    0087 ;DELAY, IS A APPROX. WAIT FOR 20.4mSECS, FOR A SYSTEM
                    0088 ;USING A 2 Mhz CRYSTAL CLOCK.
                    0089 DELAY
0020 0C14           0090         MOVLW   MSEC_20
0021 0028           0091         MOVWF   DB1
                    0092 DLY1
0022 0069           0093         CLRF    DB2
0023 02E8           0094         DECFSZ  DB1
0024 0A26           0095         GOTO    DLY2
0025 0800           0096         RETLW   0
                    0097 DLY2
0026 02E9           0098         DECFSZ  DB2             ;INNER LOOP = 1.02 MSEC.
0027 0A26           0099         GOTO    DLY2            ;       /
0028 0A22           0100         GOTO    DLY1
                    0101 ;
                    0102 ;
                    0103 TURN_GREEN_ON
0029 0486           0104         BCF     PORT_B,GRN_LED
002A 0800           0105         RETLW   0
                    0106 ;
                    0107 TURN_RED_ON
002B 04A6           0108         BCF     PORT_B,RED_LED
002C 0800           0109         RETLW   0
                    0110 ;
                    0111         END
                    0112
                    0113


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXX- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

AN521

Interfacing to AC Power Lines

INTRODUCTION

This application note describes a simple method for measuring parameters from the AC power line. Parameters such as zero crossing, frequency, and relative phase can be measured. The method is useful for measurements on 50, 60, and 400 Hz power systems with voltages up to several hundred volts. The method requires only one external component, a resistor, and is more reliable than previously published methods using capacitors or bulky, expensive transformers.

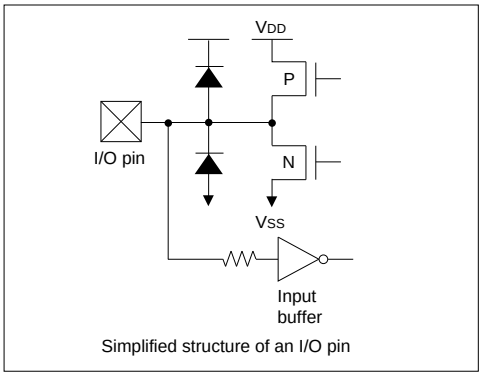
APPLICATIONS

This measurement method can be used in any application where power line parameters are used for system measurements or control. Typical applications are for switch timing (what part of the power cycle should the system be activated), power factor correction, power measurement, and power line monitor. An additional application is to generate timing or clock functions using the relatively stable power line frequency. The method is also useful for calibrating the oscillator frequency for accurate timing measurements when an inaccurate reference such as an RC oscillator is used to clock the PIC16C5X.

THEORY OF OPERATION

The application takes advantage of the input static protection circuitry that exists on all I/O pins of a CMOS PIC16C5X. These protection circuits are designed to short the inputs to the power supplies when a large overvoltage is applied, thus protecting the chip from static electricity spikes. On the PIC16C5X microcontrollers, this protection circuit is two large P-N diodes on each input (see Figure 1). These diodes will short any voltage higher than V_{DD} to the V_{DD} supply and any voltage less than V_{SS} to the V_{SS} supply. They can take several milliamps of current without any damage to the chip. High voltages can be applied directly to the chip inputs as long as they are current limited.

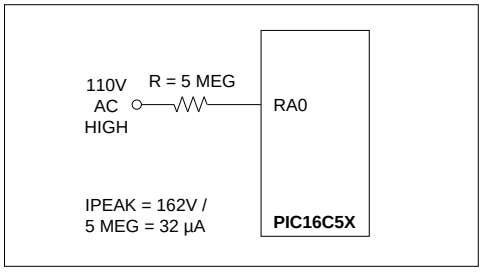
FIGURE 1 - PIC16C5X SERIES INPUT PROTECTION CIRCUIT ON I/O PINS



Simplified structure of an I/O pin

The least expensive method of current limiting is using a high value resistor. This method is shown schematically in Figure 2. The power line voltage is current limited by the resistor and then clamped by the input protection diodes internal to the PIC16C5X. A typical input waveform is shown in Figure 5. A 115V AC, 60 cycle sine wave will traverse from 0 to 2 volts in 32 µs so a typical threshold of 2 volts on the PIC16C5X I/O port will permit zero crossing detection accuracy of about 30 µs. If the typical capacitance on an I/O pin is 5 pF, then R should be (T = RC) 6 MEGohm or less for best zero crossing accuracy. A 5 MEGohm resistor with 115V AC applied to it will limit current to 32 µA, a value which is well within the safety margin of the PIC16C5X.

FIGURE 2 - CURRENT LIMITING USING AN EXTERNAL RESISTOR



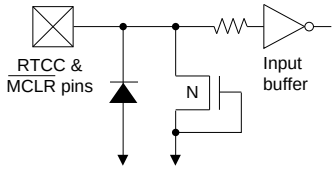
Interfacing to AC Power Lines

The user needs to be aware that the circuit required to connect the RTCC input to AC power line is slightly different. Each of the I/O pins has two diodes for input protection whereas RTCC pin has only one protection diode connecting to V_{SS} (see Figure 3). Therefore, it is necessary to connect a diode externally between RTCC pin and V_{DD} in order to clamp the voltage on RTCC pin to V_{DD} + 0.6V (approx.). See Figure 4. It is also recommended that resistor R be at least 2MΩ.

RELIABILITY

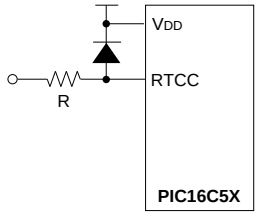
Reliability of production devices that are directly connected to AC power is always a concern. Two failure modes are possible. First, the series resistor of Figure 1 might fail short, destroying the microcontroller. This is the most unlikely failure mode of a resistor, and resistors are more reliable than transformers or capacitors, which are the alternate components for measuring line parameters. This reliability can be enhanced even further by using two resistors in series. Both would have to fail short to cause catastrophic failure, a very unlikely event. The second possible failure mode is that excessive injection of current into the PIC16C5X input might cause the protection diode to open. This would allow the input to go to the power line peak voltage (162V) and short the input transistor gate oxide, causing device failure. The maximum continuous injection current into an I/O pin is specified ±500 μA. An I/O pin is also capable of handling larger injection current (>100 mA) for a very short period (transient) of time. Therefore higher transient currents due to line voltage surges will be easily handled.

FIGURE 3 - INPUT STRUCTURE AND RTCC PIN



Simplified structure of RTCC and $\overline{MCLR}$ pins. Grounded gate NMOS device provides ESD and overvoltage protection.

FIGURE 4 - CONNECTING AC POWER LINE TO RTCC PIN



Connecting to RTCC input. R ≥ 2MΩ

FIGURE 5 - INPUT WAVEFORM

Place Input Waveform figure here.

Waveform at part pin (RA0) •

R = 100K; Line: 60 Hz, 110 V

Author: Doug Cox
Logic Products Division

Frequency Counter Using PIC16C5X

INTRODUCTION

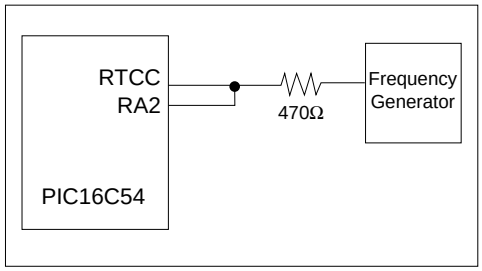
The PIC16C5X has one 8-bit timer (RTCC), which can be used with an 8-bit prescaler. The prescaler runs asynchronously, hence it can count a very high frequency. The minimum rise and fall times of the input frequency are specified to be 10nS, so the fastest clock rate the RTCC can count is 50 MHz. The prescaler must be used when measuring high frequency. Since the prescaler can be configured as a divide by 256 counter, the maximum resolution which the input frequency can be measured is 16 bits. However, the prescaler cannot be directly read like a file register. This application note depicts a unique method by which the user can "extract" the 8-bit value in the prescaler, whereby the resolution of the measurement is 16 bits with the high 8 bits in the RTCC and the low 8 bits in the prescaler.

IMPLEMENTATION

A frequency counter which can read frequencies from 50 MHz to 50 Hz was implemented in this application note to demonstrate this method of measuring the 16-bit counter value from the prescaler and RTCC.

The basic hardware for the measurement circuit is depicted in Figure 1. It consists of the frequency input at RTCC or RA4 (pin 3 in a PIC16C54). RA4 is connected to RA2. The input frequency is connected to RTCC through a 470 ohm resistor.

FIGURE 1



The RTCC is configured to measure the input frequency at RA4 of the PIC16C54. The input frequency is "gated" for a precise duration of time. Before starting this precise "gate", the RTCC is cleared (which also clears the prescaler), and the RA2 pin is configured as an input. The precise "gate" is implemented in software as an accurate delay. At the end of the delay, the RA2 pin is configured as an output going low. This will cause the input to the RTCC to be "halted" or "stopped". A 16-bit value of the input frequency is now saved in RTCC and the 8-bit prescaler. The high 8 bits are in RTCC and can be easily read. The low 8 bits have to be "shifted out". The 8-bits in the prescaler are "shifted out" by toggling RA2 with a "BSF" and "BCF" instruction. After every toggle, the value in RTCC is checked to see if the RTCC has incremented. If the number of toggles required to cause the RTCC to increment by 1 is N, then the 8-bit value in the pre-scaler can be calculated to be = (256 - N). By concatenating the calculated value and the original value in RTCC, the 16-bit value for the frequency is determined.

To measure a wide range of frequency, the following intermediate steps were taken:

Frequency Range	Precise "gate" Delay	Resolution
50 MHz - 10 MHz	1 ms	±10 KHz
10 MHz - 1 MHz	5 ms	±2 KHz
1 MHz - 100 KHz	50 ms	±200 Hz
100 KHz - 10 KHz	200 ms	±50 Hz
50 KHz - 50 Hz	50 ms†	±2 Hz

† In this case, the RTCC uses the internal 4 MHz clock and counts the number of instance of the external clock. Maximum Time required is 50 ms to make a ± 2 Hz accurate measurement for 10 KHz input frequency.

The check for the correct frequency is done automatically starting with the high frequency and going down to the low frequency. The maximum time required for each conversion is approximately 310 ms. In other words, three frequency checks are done every second.

CONCLUSION

The PIC16C5X family can be used to make a 16-bit measurement of input frequency with a small overhead of one resistor and one I/O port.

Author: Stan D'Souza
Logic Products Division

APPENDIX A

```
MPASM 00.00.66 Beta               6-16-1994  23:59:52                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;Title DISPLAY.ASM
                     0002 ;This file displays a binary value found in the display
                     0003 ;register “DisplayRegister”. The binary value is converted
                     0004 ;to BCD and then displayed.
                     0005         list p=16c71,f=inhx8m
                     0006         include “16cxx.h”
                     0179
                     0180
                     0181
                     0006
                     0007 ;
0001                 0008 TRUE    equ     1
0000                 0009 FALSE   equ     0
0000                 0010 FUZZY   equ     FALSE
                     0011         if FUZZY
                     0012         #define _ledEn  _portb,3
                     0013         #define _ledData _portb,1
                     0014         #define _ledClk _portb,2
                     0015         else
0046                 0016         #define _ledEn  _portb,3
0047                 0017         #define _ledData _portb,2
0048                 0018         #define _ledClk _portb,1
                     0019         endif
0011                 0020 HighFreq        equ     0x11
0012                 0021 LowFreq         equ     0x12
001A                 0022 acca    equ     1a
001B                 0023 accb    equ     1b
001C                 0024 accc    equ     1c
001D                 0025 accd    equ     1d
001E                 0026 acce    equ     1e
001F                 0027 accf    equ     1f
0009                 0028 time    equ     09
0010                 0029 temp    equ     10
                     0030 ;
                     0031 ;
                     0032         org     0
                     0033 start
0000 3004            0034         movlw   .4              ;initialize time
0001 0089            0035         movwf   time            ;       /
0002 019B            0036         clrf    accb            ;
0003 0186            0037         clrf    _portb          ;init ports
0004 0185            0038         clrf    _porta          ;       /
0005 1586            0039         bsf     _ledEn          ;disallow writes to display
0006 3017            0040         movlw   B’00010111'     ;RA3 as output, rest as inputs
0007 3070            0042         movlw   B’01110000'     ;RB4-6 as inputs rest outputs
0008 3087            0044         movlw   B’10000111'     ;ps with RTCC for Tcyl/256
0009 0062            0045         option                  ;       /
000A 0181            0046         clrf    _rtcc           ;start timer
                     0047 wait
000B 202D            0048         call    Display         ;display on leds
000C 280B            0049         goto    wait
                     0050 ;
                     0051
                          ;*****************************************************************
                     0052 ;       This subroutine converts a 8 bit binary word
                     0053 ;       into a 3 digit BCD
                     0054 ;       The input is in accb
                     0055 ;       output is in accc and accd with  lsd in ACCD.
                     0056 ;       The basic idea is that a 8 bit binary # has a value
                     0057 ;       between 0 and 255. First we check if the # is > 99
                     0058 ;       then if it is > 199. After each check we inc the MSD
                     0059 ;       Lastly we convert the LSD which will have a value
                     0060 ;       between 0 and 99.
```

```
                    0061 ;
                    0062 ;*****************************************************************
                    0063 ;
                    0064 Bin8toBcd3
000D 3002           0065         movlw   2
000E 009C           0066         movwf   accc
000F 0190           0067         clrf    temp
0010 30C7           0068         movlw   .199       ;check if # is > 199
0011 021B           0069         subwf   accb,w     ;       /
0012 1903           0070         btfsc   _z         ;= 199?
0013 2816           0071         goto    Bcd99B
0014 1803           0072         btfsc   _c         ;       /
0015 2821           0073         goto    Bcd199     ;yes then do >200 #
                    0074 Bcd99B
0016 039C           0075         decf    accc       ;else inc Msd of BCD
0017 3063           0076         movlw   .99        ;and see > 99
0018 021B           0077         subwf   accb,w     ;       /
0019 1903           0078         btfsc   _z         ; == 99?
001A 281D           0079         goto    Bcd99A     ;yes then skip over
001B 1803           0080         btfsc   _c         ;       /
001C 2821           0081         goto    Bcd199     ;no then do 99
                    0082 Bcd99A
001D 039C           0083         decf    accc
                    0084 Bcd99
001E 081B           0085         movf    accb,w
001F 009D           0086         movwf   accd
0020 2823           0087         goto    get10th
                    0088 Bcd199
0021 009D           0089         movwf   accd       ;get result in ACCD
0022 039D           0090         decf    accd       ;dec to get correct value
                    0091 get10th
0023 300A           0092         movlw   .10
0024 021D           0093         subwf   accd,w     ;reduce by 10
0025 1C03           0094         btfss   _c         ;see if done
0026 282A           0095         goto    BcdOver    ;yes then end
0027 009D           0096         movwf   accd       ;get new value in ACCD
0028 0A90           0097         incf    temp       ;inc 10s count
0029 2823           0098         goto    get10th    ;do next
                    0099 BcdOver
002A 0E10           0100         swapf   temp,w     ;get in w
002B 049D           0101         iorwf   accd       ;or with 1s
002C 0008           0102         return
                    0103 ;
                    0104 ;*************************************************************
                    0105 ;       This routine displays 3 digits on a LT8522 display.
                    0106 ;       Three wires are required to drive the display
                    0107 ;       Enable –> active low when writing to display
                    0108 ;       Clock  –> 1 start followed by 35 more (36 total)
                    0109 ;                   36 clock required for load to occur.
                    0110 ;                   Rising edge of Clock is active.
                    0111 ;       Data   –> start data bit = high;
                    0112 ;                   1st data bit –> segment A of MSD
                    0113 ;                   2nd data bit –> segment B of MSD
                    0114 ;                   so on...
                    0115 ;                   8th data bit –> d.p. of MSD
                    0116 ;                   9th data bit –> segment A of 2nd digit
                    0117 ;                   10th data bit –> segment B of 2nd digit
                    0118 ;                   so on...
                    0119 ;                   16th data bit –> d.p. of 2nd digit
                    0120 ;                   17th data bit –> segment A of LSD
                    0121 ;                   18th data bit –> segment B of LSD
                    0122 ;                   so on ...
                    0123 ;                   24th data bit –> d.p. of LSD
                    0124 ;                   25th data bit –> appears on pin 4 of display
                    0125 ;                   26th data bit –> appears on pin 5 of display
                    0126 ;                   so on ...
                    0127 ;                   34th data bit –> appears on pin 13 of display.
                    0128 ;                   to dirve segment set data = high.
```

```
                  0129 ;
                  0130 ;       The routine does a leading zero blanking.
                  0131 ;       The 3 BCD nibbles should be available in accc and accd,
                  0132 ;       with the MSD in the low nibble of accc.
                  0133 ;*************************************************************
                  0134 Display
002D 205E         0135         call    StartDisplay
002E 0E11         0136         swapf   HighFreq,w
002F 390F         0137         andlw   0x0f
0030 2040         0138         call    LedValue
0031 2052         0139         call    DisplayW
0032 0811         0140         movf    HighFreq,w
0033 390F         0141         andlw   0x0f
0034 2040         0142         call    LedValue
0035 2052         0143         call    DisplayW
0036 0E12         0144         swapf   LowFreq,w
0037 390F         0145         andlw   0x0f
0038 2040         0146         call    LedValue
0039 2052         0147         call    DisplayW
003A 0812         0148         movf    LowFreq,w
003B 390F         0149         andlw   0x0f
003C 2040         0150         call    LedValue
003D 2052         0151         call    DisplayW
003E 2064         0152         call    EndDisplay
003F 3400         0153         retlw   0
                  0154 ;
                  0155 ;
                  0156 ;
                  0157 LedValueAddress
                  0158         if LedValueAddress < 0x100
                  0159 LedValue
0040 018A         0160         clrf    _pclath
0041 0782         0161         addwf   _pcl
0042 34FC         0162         retlw   0xfc    ;code for 0
0043 3460         0163         retlw   0x60    ;code for 1
0044 34DA         0164         retlw   0xda    ;code for 2
0045 34F2         0165         retlw   0xf2    ;code for 3
0046 3466         0166         retlw   0x66    ;code for 4
0047 34B6         0167         retlw   0xb6    ;code for 5
0048 34BE         0168         retlw   0xbe    ;code for 6
0049 34E0         0169         retlw   0xe0    ;code for 7
004A 34FE         0170         retlw   0xfe    ;code for 8
004B 34E6         0171         retlw   0xe6    ;code for 9
004C 34EE         0172         retlw   0xee    ;code for A
004D 343E         0173         retlw   0x3e    ;code for b
004E 349C         0174         retlw   0x9c    ;code for C
004F 347A         0175         retlw   0x7a    ;code for d
0050 349E         0176         retlw   0x9e    ;code for E
0051 348E         0177         retlw   0x8e    ;code for F
                  0178         endif
                  0179 ;
                  0180 ;
                  0181 DisplayW
0052 0090         0182         movwf   temp
0053 3008         0183         movlw   .8
0054 009A         0184         movwf   acca
                  0185 DisplayLoop
0055 0D90         0186         rlf     temp
0056 1803         0187         btfsc   _c
0057 1506         0188         bsf     _ledData
0058 1486         0189         bsf     _ledClk
0059 1086         0190         bcf     _ledClk
005A 1106         0191         bcf     _ledData
005B 0B9A         0192         decfsz  acca
005C 2855         0193         goto    DisplayLoop
005D 3400         0194         retlw   0
                  0195 ;
                  0196 ;
```

```
                     0197 StartDisplay
005E 1186            0198         bcf     _ledEn
005F 1506            0199         bsf     _ledData
0060 1486            0200         bsf     _ledClk
0061 1086            0201         bcf     _ledClk
0062 1106            0202         bcf     _ledData
0063 3400            0203         retlw   0
                     0204 ;
                     0205 EndDisplay
0064 1486            0206         bsf     _ledClk
0065 1086            0207         bcf     _ledClk
0066 1486            0208         bsf     _ledClk
0067 1086            0209         bcf     _ledClk
0068 1486            0210         bsf     _ledClk
0069 1086            0211         bcf     _ledClk
006A 1586            0212         bsf     _ledEn
006B 3400            0213         retlw   0
                     0214
                     0215 ;
                     0216         org     0x1ff
01FF 2800            0217         goto    start
                     0218
                     0219 ;
                     0220         end
                     0221 ;
                     0222
                     0223


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

NOTES:

AN513

Analog to Digital Conversion

INTRODUCTION

This application note describes a method for implementing analog to digital conversion on the PIC16C5X series of microcontrollers. The converter requires only five external components and is software and hardware configurable for conversion resolutions from 6 bits up to 10 bits and conversion times of 250µs or longer. The method is useable for both voltage and current conversion and uses a software calibration technique that compensates for time and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X microcontrollers are offered as One-Time-Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X's Sleep mode.
* PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

The application uses a capacitive charging circuit (see Figure 1) to convert the input voltage to time, which can be easily measured using a microcontroller. First, the reference voltage is applied to the input voltage to current converter (U1). The equivalent circuit is shown in Figure 2. This circuit provides a linearly variable current as a function of input voltage. The logarithmic characteristic that would occur if the input voltage was applied directly to an RC is not present. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the resistor and capacitor, changes in the input threshold voltage and temperature variations. After the software calibration value is measured, the capacitor is discharged (see Figure 3) and the input voltage is connected to V_{IN}. The time to trip the threshold is measured for the input voltage and compared to the calibration value to determine the actual input voltage.

FIGURE 1 - VOLTMETER A TO D CONVERTER

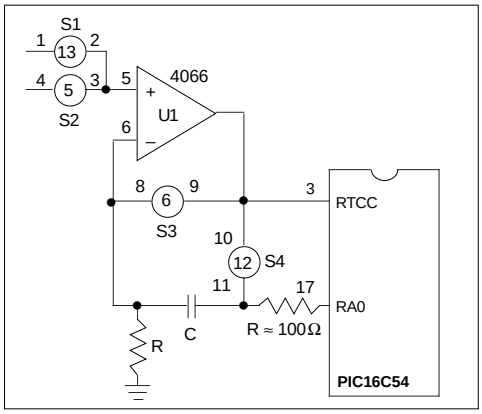


FIGURE 2 - VOLTMETER MEASUREMENT CYCLE

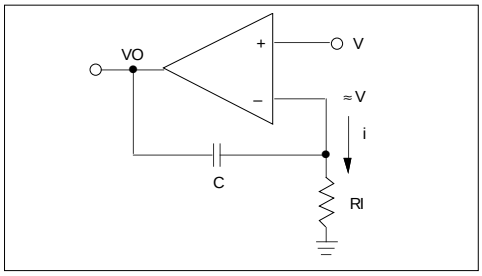
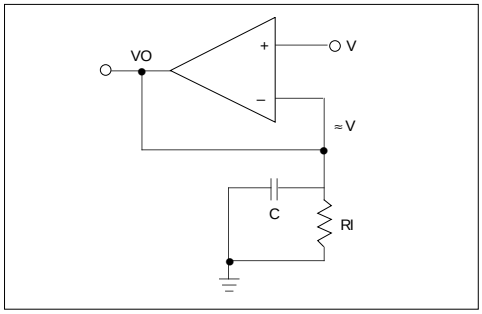


FIGURE 3 - VOLTMETER DISCHARGE CYCLE



Analog to Digital Conversion

CIRCUIT CONFIGURATION

The values of R and C are selected based upon the number of bits of resolution required.

RC = (Vi ˛ T)/Vt

Where:

Vi = Lowest voltage to be measured (at least ten lsb's)

T = Time to do the number of bits of resolution desired

Vt = Threshold voltage of the PIC16C5X input being used

Actual value for RC should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use a 3 volt input and 8 bits resolution with a 8 MHz clock and 6 instruction cycles per count:

Vi = 100 mV

T = 256 *1/8 MHz * 4 clocks/cycle * 6 cycles = 768µS

Vt = 3.0V (est)

For input voltages greater than 3 volts a resistor divider network should be used to keep the maximum voltage on VIN to less than 3 Volts. For best performance the reference voltage should be between 2 and 3 volts.

The circuit can also be used as a current mode A to D converter. In this case the input voltage to current converter is not needed and the reference current and input current are both routed via analog switches directly into the capacitor.

CIRCUIT PERFORMANCE

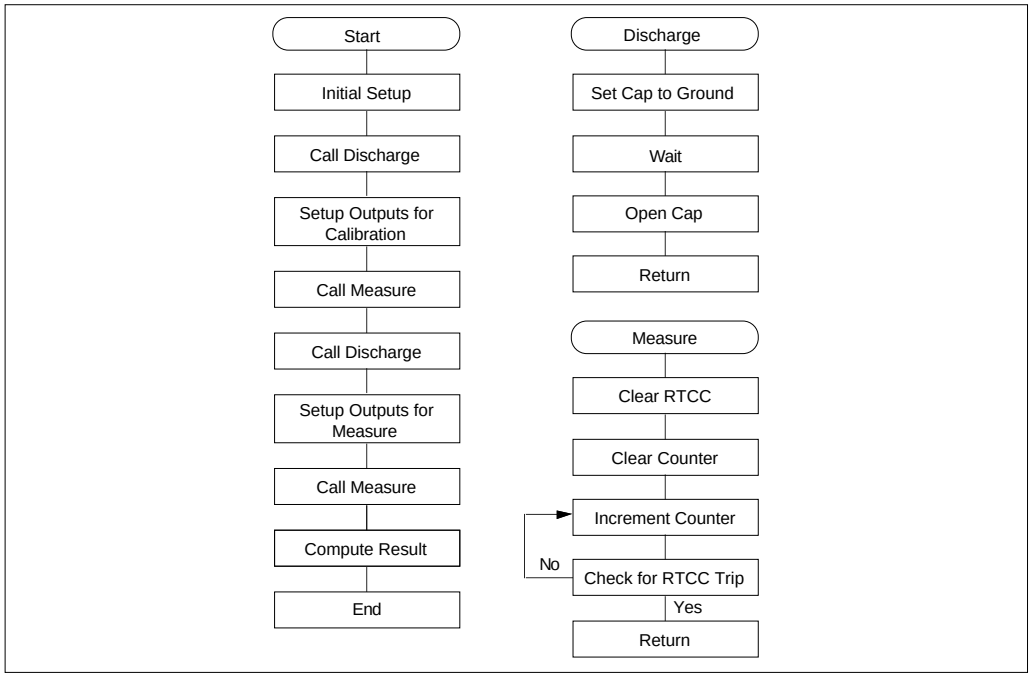
The calibration cycle removes all first order errors (offset, gain, R and C inaccuracy, power supply voltage and temperature) except the reference voltage drift. Any change in the reference voltage, including noise, between the calibration cycle and the measurement cycle may result in measurement errors. Other error sources are analog switch leakage, resistor and capacitor nonlinearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the converter to be accurate within +/- 1% of full scale.

Example

Assembly code implementing the circuit of Figure 1 is listed in Appendix A: This code measures the time up to 16 bits and calculates the results using 16-bit multiply and divide subroutines. In actual applications, if measurement accuracy permits, it may be advantageous to use 8 bits. The math code can be substantially reduced and the measure time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 4 - TRANSMISSION FLOW CHART



APPENDIX A:

```
MPASM B0.54                                                    PAGE  1
VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90


                        TITLE  'VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90'
                        LIST   P=16C54,F=inhx16,n=0

0008                ACCA    EQU     8
000A                ACCB    EQU     0A
000C                ACCC    EQU     0C
000E                ACCD    EQU     0E
0010                ACCE    EQU     10
0012                TMEAS   EQU     12
0014                TEMP    EQU     14

0060                VCALMS  EQU     60              ;VCAL MSB VALUE IN HEX
00A4                VCALLS  EQU     0A4             ;VCAL LSB VALUE IN HEX

                        ORG 1FF
01FF 0A58               GOTO    VOLTS           ;PROGRAM CODE
                        ORG     0               ;SUBROUTINES

0000 0209           MADD    MOVF    ACCA+1,W
0001 01EB               ADDWF   ACCB+1          ;ADD LSB
0002 0603               BTFSC   3,0             ;ADD IN CARRY
0003 02AA               INCF    ACCB
0004 0208               MOVF    ACCA,W
0005 01EA               ADDWF   ACCB            ;ADD MSB
0006 0800               RETLW   0
0007 0000               NOP

0008 0915           MPY     CALL    SETUP           ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E           MLOOP   RRF     ACCD            ;ROTATE D RIGHT
000A 032F               RRF     ACCD+1
000B 0603               SKPNC                   ;NEED TO ADD?
000C 0900               CALL    MADD
000D 032A               RRF     ACCB
000E 032B               RRF     ACCB+1
000F 032C               RRF     ACCC
0010 032D               RRF     ACCC+1
0011 02F4               DECFSZ  TEMP            ;LOOP UNTIL ALL BITS CHECKED
0012 0A09               GOTO    MLOOP
0013 0800               RETLW   0

0014 0000               NOP
0015 0C10           SETUP   MOVLW   10
0016 0034               MOVWF   TEMP
0017 020A               MOVF    ACCB,W          ;MOVE B TO D
0018 002E               MOVWF   ACCD
0019 020B               MOVF    ACCB+1,W
001A 002F               MOVWF   ACCD+1
001B 020C               MOVF    ACCC,W
001C 0030               MOVWF   ACCE
001D 020D               MOVF    ACCC+1,W
001E 0031               MOVWF   ACCE+1
001F 006A               CLRF    ACCB
0020 006B               CLRF    ACCB+1
0021 0800               RETLW   0

0022 0000               NOP
0023 0915           DIV     CALL    SETUP
0024 0C20               MOVLW   20
0025 0034               MOVWF   TEMP
0026 006C               CLRF    ACCC
0027 006D               CLRF    ACCC+1
```

```
0028 0403           DLOOP   CLRC
0029 0371                   RLF     ACCE+1
002A 0370                   RLF     ACCE
002B 036F                   RLF     ACCD+1
002C 036E                   RLF     ACCD
002D 036D                   RLF     ACCC+1
002E 036C                   RLF     ACCC
002F 0208                   MOVF    ACCA,W
0030 008C                   SUBWF   ACCC,W          ;CHECK IF A>C
0031 0743                   SKPZ
0032 0A35                   GOTO    NOCHK
0033 0209                   MOVF    ACCA+1,W
0034 008D                   SUBWF   ACCC+1,W        ;IF MSB EQUAL THEN CHECK LSB
0035 0703           NOCHK   SKPC                    ;CARRY SET IF C>A
0036 0A3E                   GOTO    NOGO
0037 0209                   MOVF    ACCA+1,W        ;C-A INTO C
0038 00AD                   SUBWF   ACCC+1
0039 0703                   BTFSS   3,0
003A 00EC                   DECF    ACCC
003B 0208                   MOVF    ACCA,W
003C 00AC                   SUBWF   ACCC
003D 0503                   SETC                    ;SHIFT A 1 INTO B (RESULT)
003E 036B           NOGO    RLF     ACCB+1
003F 036A                   RLF     ACCB
0040 02F4                   DECFSZ  TEMP            ;LOOP UNTILL ALL BITS CHECKED
0041 0A28                   GOTO    DLOOP
0042 0800                   RETLW   0

0043 0C0E           DSCHRG  MOVLW   B'00001110'     ;DISCHARGE C (RA0 ON)
0044 0005                   TRIS    5
0045 0CFF                   MOVLW   0FF
0046 0034                   MOVWF   TEMP
0047 02F4           LOOP    DECFSZ  TEMP            ;WAIT
0048 0A47                   GOTO    LOOP
0049 0C0F                   MOVLW   B'00001111'     ;ALL RA HIGH Z
004A 0005                   TRIS    5
004B 0800                   RETLW   0

004C 0061           M_TIME  CLRF    1               ;CLEAR RTCC REGISTER
004D 0069                   CLRF    ACCA+1          ;CLEAR 16 BIT COUNTER
004E 0068                   CLRF    ACCA
004F 03E9           TLOOP   INCFSZ  ACCA+1
0050 0A54                   GOTO    ENDCHK
0051 03E8                   INCFSZ  ACCA
0052 0A54                   GOTO    ENDCHK
0053 0A56                   GOTO    END_M
0054 0701           ENDCHK  BTFSS   1,0             ;CHECK FOR RTCC TRIP
0055 0A4F                   GOTO    TLOOP
0056 0201           END_M   MOVF    1,W
0057 0800                   RETLW   0

0058 0C06           VOLTS   MOVLW   B'00000110'     ;SET S2 AND S3 HIGH(ON WHEN ACTIVATED)
0059 0026                   MOVWF   6
005A 0CF0                   MOVLW   B'11110000'     ;ACTIVATE SWITCHES S1-S4
005B 0006                   TRIS    6
005C 0C28                   MOVLW   B'00101000'     ;SELECT POSITIVE EDGE FOR RTCC
005D 0002                   OPTION
005E 0C00                   MOVLW   B'00000000'
005F 0025                   MOVWF   5               ;SET RA0 LOW (ON WHEN ACTIVATED)

0060 0943           MEAS    CALL    DSCHRG          ;CHARGE CAPACITOR TO VIN
0061 0C0A                   MOVLW   B'00001010'     ;S2 AND S4 ON
0062 0026                   MOVWF   6
0063 094C                   CALL    M_TIME          ;MEASURE TIME
0064 0209                   MOVF    ACCA+1,W
0065 0033                   MOVWF   TMEAS+1         ;STORE LSB
0066 0208                   MOVF    ACCA,W
0067 0032                   MOVWF   TMEAS           ;STORE MSB
```

```
0068 0C05            CAL      MOVLW   B'00000101'     ;S1 AND S3 ON
0069 0026                     MOVWF   6
006A 0943                     CALL    DSCHRG          ;CHARGE CAPACITOR TO VREF
006B 0C09                     MOVLW   B'00001001'     ;S1 AND S4 ON
006C 0026                     MOVWF   6
006D 094C                     CALL    M_TIME          ;MEASURE TIME

006E 0CA4                     MOVLW   VCALLS
006F 002B                     MOVWF   ACCB+1
0070 0C60                     MOVLW   VCALMS
0071 002A                     MOVWF   ACCB

0072 0908                     CALL    MPY             ;MULTIPLY ACCA(TCAL) * ACCB(VREF)
0073 0213                     MOVF    TMEAS+1,W
0074 0029                     MOVWF   ACCA+1
0075 0212                     MOVF    TMEAS,W
0076 0028                     MOVWF   ACCA

0077 0923                     CALL    DIV             ;DIVIDE ACCB(TCAL * V) BY ACCA(TMEAS)

0078 0A58                     GOTO VOLTS

                              END

Errors   :    0
Warnings :    0
```

NOTES:

AN512

Implementing Ohmeter/Temperature Sensor

INTRODUCTION

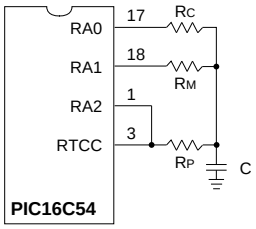
This application note describes a method for implementing an ohmmeter or resistance type temperature sensor using the PIC16C5X series of microcontrollers. The ohmmeter requires only two external components and is software and hardware configurable for resistance measurement with resolutions from 6 bits up to 10 bits with measurement times of 250 µs (6 bits at 8 MHz) or longer. The method uses a software calibration technique that compensates for voltage, time, and temperature drift as well as component errors. The PIC16C5X microcontrollers are ideal for simple analog applications because:

* Very low cost.
* Few external components required.
* Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* Calibration in software for improved measurement accuracy.
* Power savings using PIC16C5X's Sleep mode.
* PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

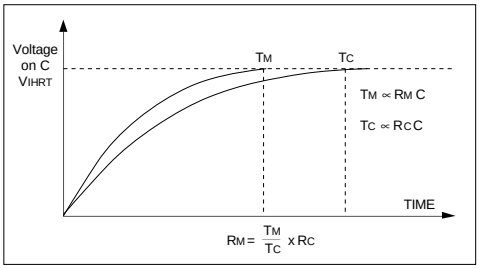
The application uses a capacitive charging circuit (Figure 1) to convert the resistance to time, which can be easily measured using a microcontroller. First, a reference voltage (usually V_{DD}) is applied to a calibration resistor, Rc. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the capacitor, changes in the input threshold voltage and temperature variations. After C is discharged, the reference voltage is applied to the resistance to be measured (or thermistor). The time to trip the threshold is then measured and compared to the calibration value to determine the actual resistance (Figure 2). In the temperature sensing mode, the temperature is calculated using a lookup table.

FIGURE 1 - OHMETER/TEMPERATURE SENSOR



Note: R_P is a small resistor (100-200Ω) to limit peak current through RA2 while discharging or through RA2 or RTCC in the event of an ESD or EOS related breakdown.

FIGURE 2 - OHMETER/TEMPERATURE SENSOR



CIRCUIT CONFIGURATION

The values of Rc and C are selected based upon the number of bits of resolution required. Rc should be approximately one half the largest value resistance to be measured and:

$$C = \frac{-T}{R_M * \ln\left(1 - \frac{Vt}{Vr}\right)}$$

Where:

Vr = Reference voltage

T = Time to do the number of bits of resolution desired

Vt = Threshold voltage of the PIC16C5X input being used

R_M = Maximum resistance value to be measured

Actual value for C should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use R_M=200K for 8-bits resolution with an 8 MHz clock, Vr = 5V, Vt = 3V, Rc =100K and 6 instruction cycles per count:

T = 256 counts * 1/8 MHz * 4 clocks/instruction * 6 instructions/count = 768 µs

C = 4200 pF [Use 3900 pF]

CIRCUIT PERFORMANCE

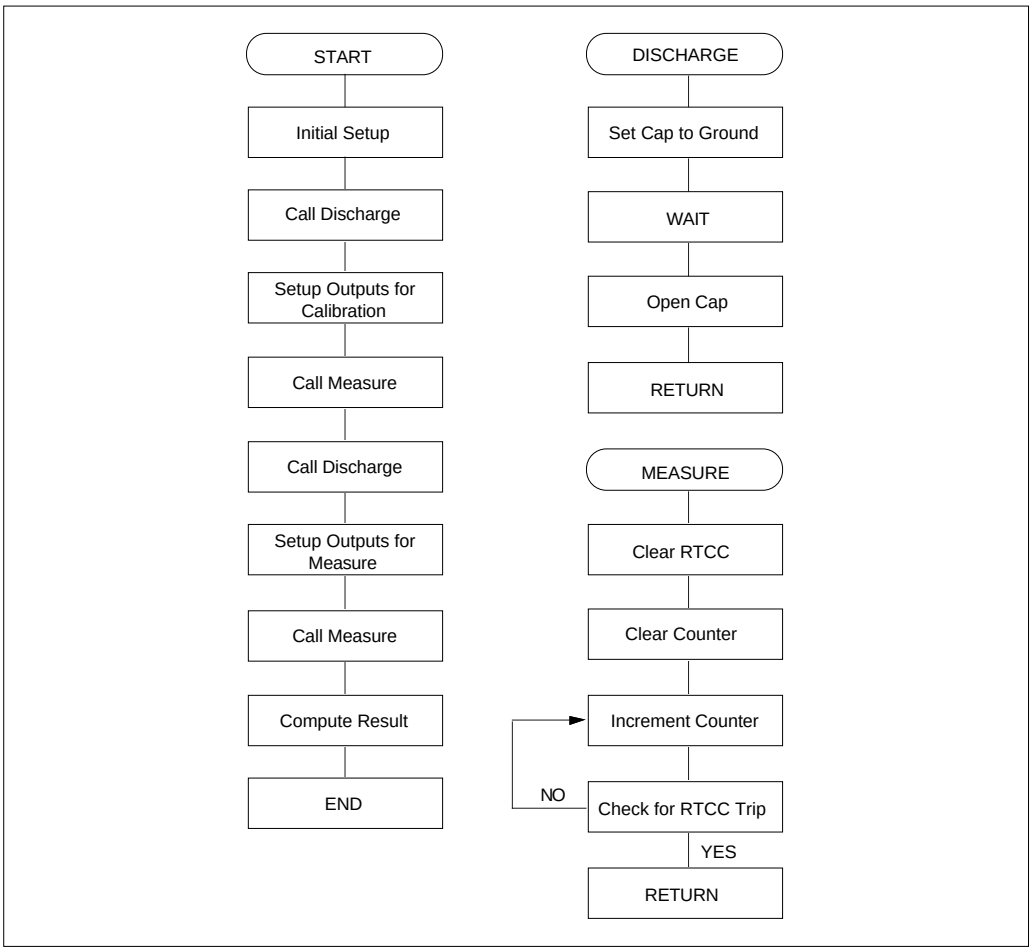
The calibration cycle removes all first order errors (offset, gain, C inaccuracy, power supply voltage and temperature) except R absolute accuracy. A low drift resistor should be selected for R and its value stored in software to reduce measurement errors. Other error sources are I/O pin leakage, resistor and capacitor non-linearities, input threshold uncertainty and time measurement uncertainty (+/- one instruction cycle time). Measured performance shows the ohmmeter to be accurate within +/- 1% over one decade.

Example

The assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures time up to 16 bits (65535 measure cycles) and calculates the results using 16 bit multiply and divide subroutines. In actual applications, it is more efficient to use 8 bit measurements if application accuracies permit. The math code will be substantially reduced and measurement time is reduced by the simpler code and shorter count.

Author: Doug Cox
Logic Products Division

FIGURE 3 - TRANSMISSION FLOW CHART



APPENDIX A:

```
MPASM B0.54                                                    PAGE  1


                        LIST   P=16C54,F=inhx8M

0008                ACCA   EQU     8
000A                ACCB   EQU     0A
000C                ACCC   EQU     0C
000E                ACCD   EQU     0E
0010                ACCE   EQU     10
0012                TCAL   EQU     12
0014                TEMP   EQU     14

002F                RCALMS EQU     2F              ;RCAL MSB VALUE IN HEX
003C                RCALLS EQU     3C              ;RCAL LSB VALUE IN HEX

                        ORG     1FF
01FF 0A58               GOTO    OHMS
                        ORG     0

0000 0209           MADD   MOVF    ACCA+1,W
0001 01EB                  ADDWF   ACCB+1          ;ADD LSB
0002 0603                  BTFSC   3,0             ;ADD IN CARRY
0003 02AA                  INCF    ACCB
0004 0208                  MOVF    ACCA,W
0005 01EA                  ADDWF   ACCB            ;ADD MSB
0006 0800                  RETLW   0
0007 0000                  NOP

0008 0915           MPY    CALL    SETUP           ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E           MLOOP  RRF     ACCD            ;ROTATE D RIGHT
000A 032F                  RRF     ACCD+1
000B 0603                  SKPNC                   ;NEED TO ADD?
000C 0900                  CALL    MADD
000D 032A                  RRF     ACCB
000E 032B                  RRF     ACCB+1
000F 032C                  RRF     ACCC
0010 032D                  RRF     ACCC+1
0011 02F4                  DECFSZ  TEMP            ;LOOP UNTIL ALL BITS CHECKED
0012 0A09                  GOTO    MLOOP
0013 0800                  RETLW   0

0014 0000                  NOP
0015 0C10           SETUP  MOVLW   10
0016 0034                  MOVWF   TEMP
0017 020A                  MOVF    ACCB,W          ;MOVE B TO D
0018 002E                  MOVWF   ACCD
0019 020B                  MOVF    ACCB+1,W
001A 002F                  MOVWF   ACCD+1
001B 020C                  MOVF    ACCC,W
001C 0030                  MOVWF   ACCE
001D 020D                  MOVF    ACCC+1,W
001E 0031                  MOVWF   ACCE+1
001F 006A                  CLRF    ACCB
0020 006B                  CLRF    ACCB+1
0021 0800                  RETLW   0
0022 0000                  NOP
0023 0915           DIV    CALL    SETUP
0024 0C20                  MOVLW   20
0025 0034                  MOVWF   TEMP
0026 006C                  CLRF    ACCC
0027 006D                  CLRF    ACCC+1
0028 0403           DLOOP  CLRC
0029 0371                  RLF     ACCE+1
002A 0370                  RLF     ACCE
002B 036F                  RLF     ACCD+1
```

```
002C 036E                 RLF     ACCD
002D 036D                 RLF     ACCC+1
002E 036C                 RLF     ACCC
002F 0208                 MOVF    ACCA,W
0030 008C                 SUBWF   ACCC,W          ;CHECK IF A>C
0031 0743                 SKPZ
0032 0A35                 GOTO    NOCHK
0033 0209                 MOVF    ACCA+1,W
0034 008D                 SUBWF   ACCC+1,W        ;IF MSB EQUAL THEN CHECK LSB
0035 0703          NOCHK  SKPC                    ;CARRY SET IF C>A
0036 0A3E                 GOTO    NOGO
0037 0209                 MOVF    ACCA+1,W        ;C-A INTO C
0038 00AD                 SUBWF   ACCC+1
0039 0703                 BTFSS   3,0
003A 00EC                 DECF    ACCC
003B 0208                 MOVF    ACCA,W
003C 00AC                 SUBWF   ACCC
003D 0503                 SETC                    ;SHIFT A 1 INTO B (RESULT)
003E 036B          NOGO   RLF     ACCB+1
003F 036A                 RLF     ACCB
0040 02F4                 DECFSZ  TEMP            ;LOOP UNTILL ALL BITS CHECKED
0041 0A28                 GOTO    DLOOP
0042 0800                 RETLW   0

0043 0C0B          DSCHRG MOVLW   B'00001011'     ;ACTIVATE RA2
0044 0005                 TRIS    5
0045 0CFF                 MOVLW   0FF
0046 0034                 MOVWF   TEMP
0047 02F4          LOOP   DECFSZ  TEMP            ;WAIT
0048 0A47                 GOTO    LOOP
0049 0C0F                 MOVLW   B'00001111'     ;ALL OUTPUTS OFF
004A 0005                 TRIS    5
004B 0800                 RETLW   0

004C 0061          M_TIME CLRF    1               ;CLEAR RTCC
004D 0069                 CLRF    ACCA+1
004E 0068                 CLRF    ACCA
004F 03E9          TLOOP  INCFSZ  ACCA+1
0050 0A54                 GOTO    ENDCHK
0051 03E8                 INCFSZ  ACCA
0052 0A54                 GOTO    ENDCHK
0053 0A56                 GOTO    END_M
0054 0701          ENDCHK BTFSS   1,0             ;CHECK FOR RTCC TRIP
0055 0A4F                 GOTO    TLOOP
0056 0201          END_M  MOVF    1,W
0057 0800                 RETLW   0

0058 0C03          OHMS   MOVLW   B'00000011'     ;SET RA0 AND RA1 HIGH (ON WHEN ACTIVATED)
0059 0025                 MOVWF   5
005A 0C28                 MOVLW   B'00101000'     ;SELECT POSITIVE EDGE FOR RTCC
005B 0002                 OPTION

005C 0943          CAL    CALL    DSCHRG          ;DISCHARGE CAPACITOR
005D 0C0E                 MOVLW   B'00001110'     ;ACTIVATE RA0
005E 0005                 TRIS    5
005F 094C                 CALL    M_TIME          ;MEASURE TIME
0060 0209                 MOVF    ACCA+1,W
0061 0033                 MOVWF   TCAL+1          ;STORE LSB
0062 0208                 MOVF    ACCA,W
0063 0032                 MOVWF   TCAL            ;STORE MSB

0064 0943          MEAS   CALL    DSCHRG          ;DISCHARGE CAPACITOR
0065 0C0D                 MOVLW   B'00001101'     ;ACTIVATE RA1
0066 0005                 TRIS    5
0067 094C                 CALL    M_TIME          ;MEASURE TIME

0068 0C3C                 MOVLW   RCALLS          ;CALIBRATION LSB VALUE
0069 002B                 MOVWF   ACCB+1
006A 0C2F                 MOVLW   RCALMS          ;CALIBRATION MSB VALUE
```

```
006B 002A                   MOVWF   ACCB

006C 0908                   CALL    MPY             ;MULTIPLY ACCA(MEAS) * ACCB(RCAL)
006D 0213                   MOVF    TCAL+1,W
006E 0029                   MOVWF   ACCA+1
006F 0212                   MOVF    TCAL,W
0070 0028                   MOVWF   ACCA

0071 0923                   CALL    DIV             ;DIVIDE ACCB(MEAS * R) BY ACCA(TCAL)

0072 0A58                   GOTO    OHMS

                            END


Errors   :    0
Warnings :    0
```

NOTES:

AN519

Implementing a Simple Serial Mouse Controller

INTRODUCTION

The mouse is becoming increasingly popular as a standard pointing data entry device. It is no doubt that the demand of the mouse is increasing. Various kinds of mice can be found in the market, including optical mouse, opto-mechanical mouse, and its close relative, trackball. The mouse interfaces to the host via an RS-232 port or a dedicated interface card. Their mechanisms are very similar. The major electrical components of a mouse are:

- Microcontroller
- Photo-transistors
- Infrared emitting diode
- Voltage conversion circuit

The intelligence of the mouse is provided by the microcontroller, hence the features and performance of a mouse is greatly related to the microcontroller used.

This application note describes the implementation of a serial mouse using the PIC16C54. The PIC16C54 is a high speed 8-bit CMOS microcontroller offered by Microchip Technology Inc. It is an ideal candidate for a mouse controller.

THEORY OF OPERATION

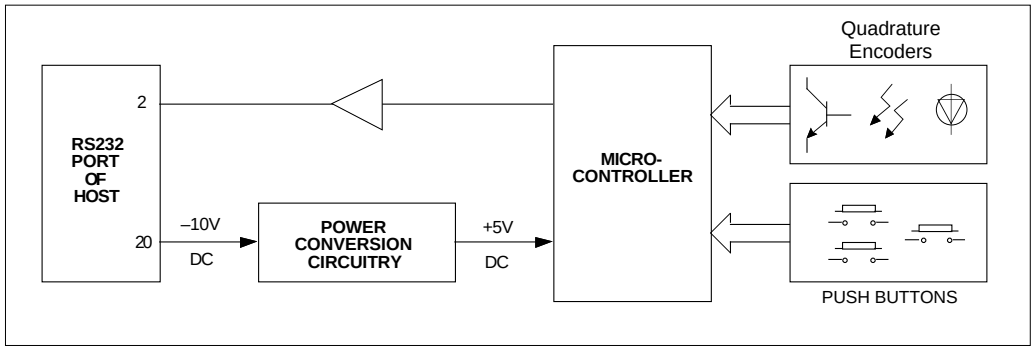
A mouse can be divided into several functional blocks:

- Microcontroller
- Button detection
- Motion detection
- RS-232 signal generation
- 5V DC power supply unit

A typical functional block diagram is shown in Figure 1.

In Figure 2, three push buttons are connected to the input ports of the PIC16C54. When a switch opening or closure is detected, a message is formatted and sent to the host. The X and Y movements are measured by counting the pulses generated by the photo-couplers. In the case of an opto-mechanical mouse, the infrared light emitted by the infrared diode is blocked by the rotating wheel, so that the pulses are generated on the photo-transistor side. In case of an optical mouse, the infrared light emitted by the infrared diode is reflected off the reflective pad patterned with vertical and horizontal grid lines. It is then received by the photo-transistor in the mouse. When any X or Y movement is detected, a message is formatted and sent to the host.

FIGURE 1 - FUNCTIONAL BLOCKS OF A SERIAL MOUSE



Implementing a Simple Serial Mouse Controller

The Microsoft® Mouse System and the Mouse Systems® device both use serial input techniques. The Mouse System protocol format contains five bytes of data. One byte describes the status of three push buttons, two bytes for the relative X movements and two bytes for the relative Y movements. The Microsoft protocol format contains three bytes of data describing the status of two push buttons and the relative X and Y movements. The details of these protocols are given in Table 1.

Three lines are connected to the host via the RS-232 port:

- Signal Ground
- Received Data
- Request to Send

"Received Data" carries the message sent by the mouse. While "Request to Send" provides a –10V DC for voltage conversion circuitry. A voltage of +5V DC is required for electronic components inside the mouse, however, +5V DC is not part of an RS-232 port, so voltage conversion circuitry is required. This circuit is typically composed of a 555 timer, Zener diodes, and capacitors. An example circuit is shown in Figure 3. Since the current supplied through the RS-232 port is limited to 10 mA, the mouse cannot be designed to consume more than 10 mA current unless an external power supply is provided. The PIC16C54, running at 4 MHz (1 µs instruction cycle) can provide a very high tracking speed. An 8 MHz version of PIC16C54 is also available if higher performance is desired.

FIGURE 2 - PIC16C54 PIN ASSIGNMENT

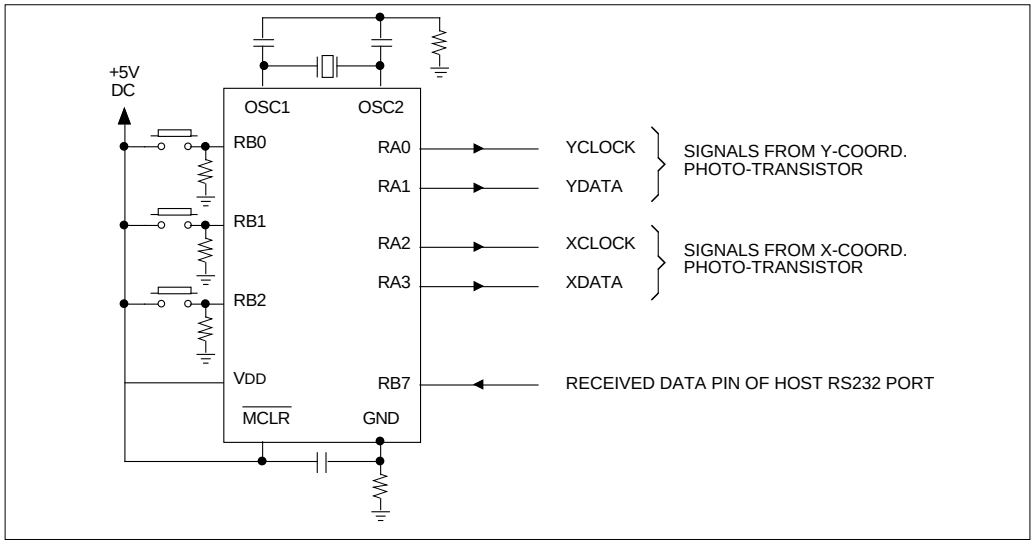
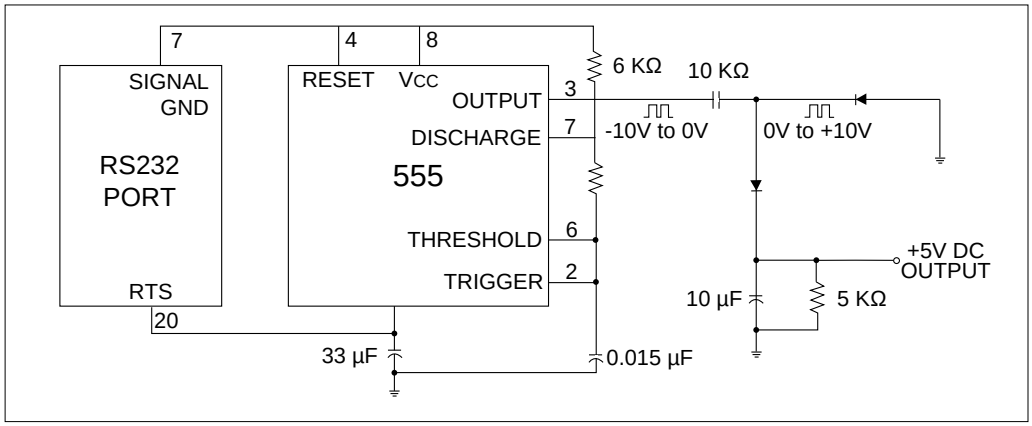


FIGURE 3 - VOLTAGE CONVERSION CIRCUITRY



ABOUT THE SOFTWARE

The major tasks performed by the software are button scanning, X and Y motion scanning, formatting and sending serial data to the host. These tasks need to be performed in parallel in order to gain better tracking speed. The pulses generated by the photo-couplers are counted while transmitting the serial signals to the RS-232 port. The number of pulses reflects the speed of the movement. The more number of pulses, the faster the movement is.

The directions of the movement are determined by the last states and the present states of the outputs of the photo-transistors. In Figure 4, XCLOCK and XDATA are outputs from the photo-transistors corresponding to the X-axis movement. XDATA is read when a rising or a falling edge of XCLOCK is detected. For right movement, XDATA is either LOW at the rising edge of XCLOCK or HIGH at the falling edge of XCLOCK. The up and down movement detections follow the same logic. In Table 1, X7:X0 are data for relative movement. If X is positive, it implies that the mouse is moving to the right. If X is negative, it implies a movement to the left. Similarly, if Y is positive, it indicates that the mouse is moving down and if Y is negative, it indicates that the mouse is moving up. The pulses generated by the photo-couplers are checked before every bit is sent. A bit takes 1/1200 second to send, if the distance between the grid lines is 1 mm, the tracking speed will be up to 1200 mm/second.

FIGURE 4 - VOLTAGE CONVERSION CIRCUITRY

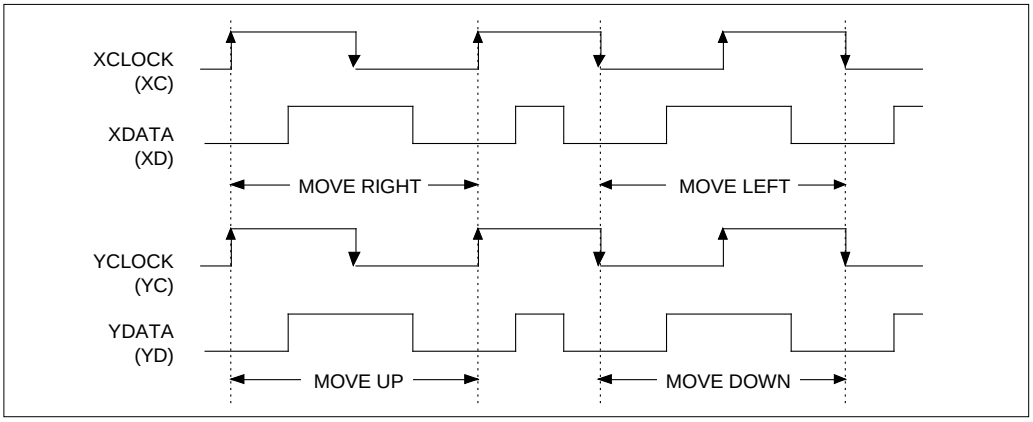


TABLE 1 - MOUSE SYSTEM AND MICROSOFT PROTOCOLS

	Mouse System Format*								Microsoft Format*							
Bit Position	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R	1	1	L	R	Y7	Y6	X7	X6
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0	0	0	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0	0	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0								
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0								

***** L = Left Key Status
M = Middle Key Status
R = Right Key Status

1 = Pressed
0 = Released

X7-X0 = X-Axis Movement Data
Y7-Y0 = Y-Axis Movement Data

The buttons are scanned after a message is sent and the time used to send the message is used as the debouncing time. The message is in an RS-232 format with 1200 baud, eight data bits, no parity, and two stop bits.

The flow charts of the main program, subroutine BYTE and subroutine BIT are shown in Figures 5, 6, and 7. Figure 5 shows the Trigger Flag is set when any change of button status or X/Y movement is detected. Subroutine BYTE is called in the main program five times to send five bytes of information. Subroutine BYTE controls the status of the “Received Data” (RD) pin. If Trigger Flag is clear, RD will always be HIGH. Hence, no message will be sent even when subroutine BYTE is called. Figure 7 shows that subroutine BIT counts the number of pulses from outputs of the photo-transistors, determines the directions, and generates 1/1200 second delay to get 1200 baud timing.

The mouse has been tested in Mouse System Mode and is functioning properly. A completed listing of the source program is given in Appendix A.

SUMMARY

The PIC16C54 from Microchip Technology Inc. provides a very cost-effective, high performance mouse implementation. Its low power (typically < 2 mA at 1 µs instruction cycle), small package (18-pin) and high reliability (on-chip watchdog timer to prevent software hang-ups) are among several reasons why the PIC16C54 is uniquely suitable for mouse applications.

This application note provides the user with a simple, fully functional serial mouse implementation. The user may use this as a starting point for a more comprehensive design. For fully implemented and compliant mouse products see Microchip's ASSP device family (MTA41XXX).

FIGURE 5 - FLOW CHART OF THE MAIN PROGRAM

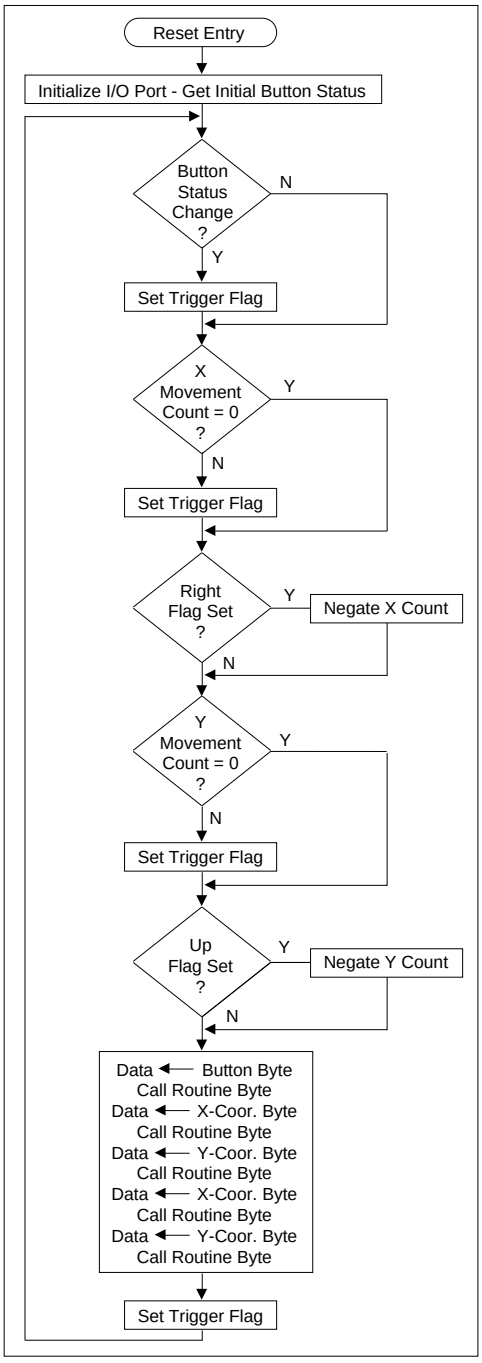


FIGURE 6 - FLOW CHART OF ROUTINE BYTE

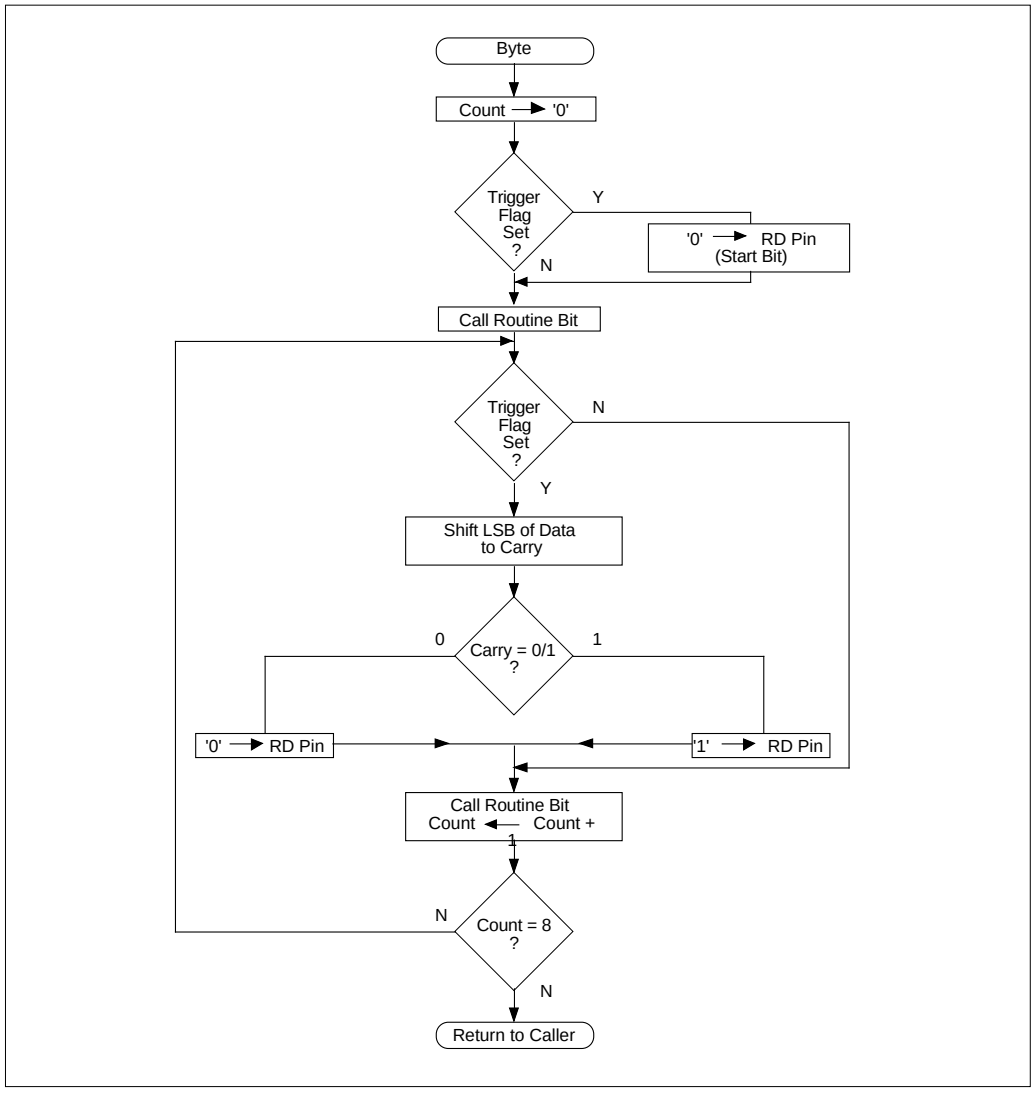
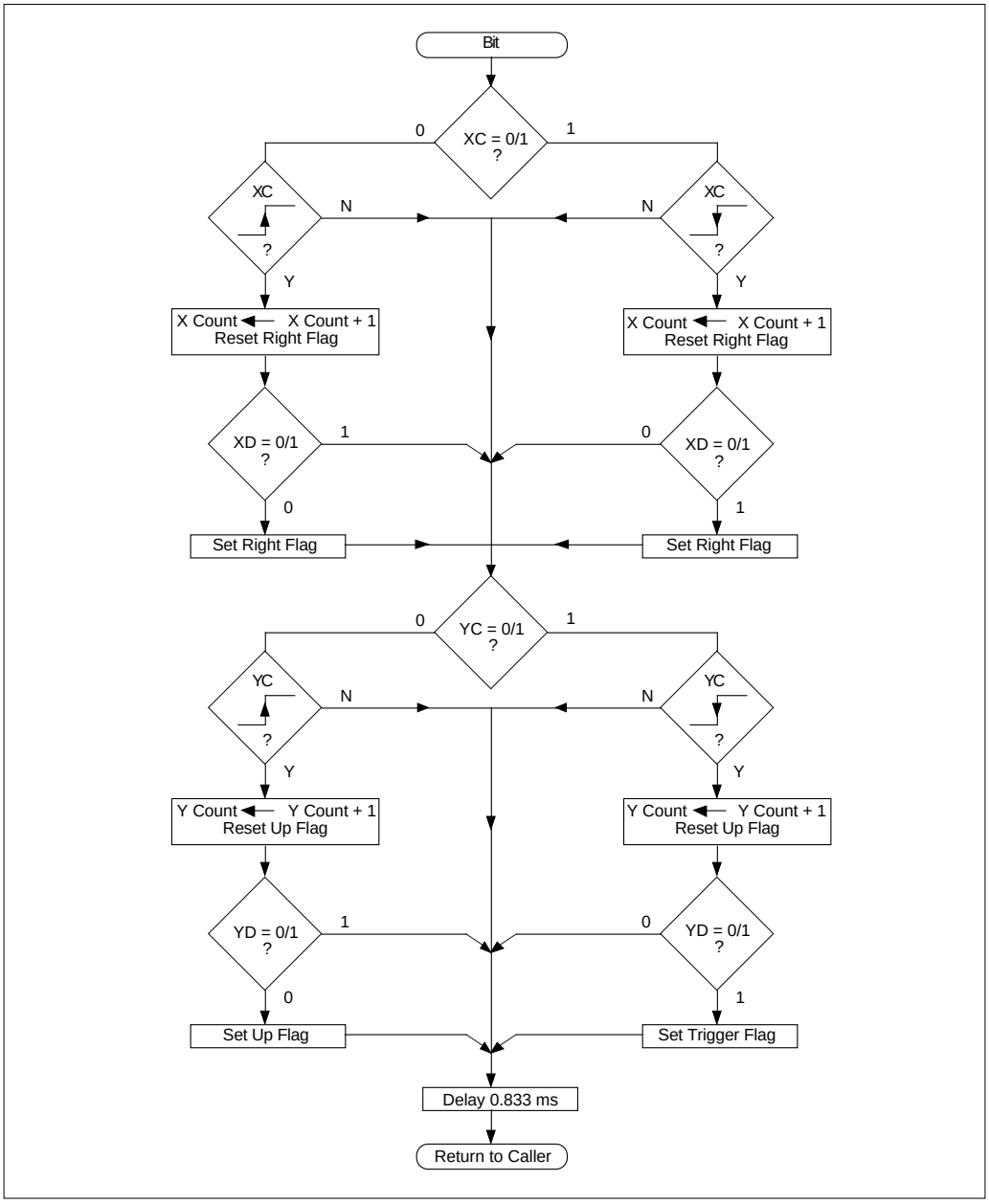


FIGURE 7 - FLOW CHART OF ROUTINE BIT



APPENDIX A:

```
MPASM B0.54                                                          PAGE  1
 MOUSE

                    TITLE " MOUSE "
                     LIST   P=16C54,R=O
                    ;
                    ;*****************************************
                    ;*                                       *
                    ;*   MOUSE CONTROLLER                    *
                    ;*                                       *
                    ;*   VERSION : 25 APRIL, 1990            *
                    ;*                                       *
                    ;*   MODE = PIC16C54XT   CLK=4.0MHZ    *
                    ;*****************************************
                    ;
                    ;_____________________
                    ;       FILES ASSIGNMENT
                    ;_____________________
                    ;
0003                STATUS      EQU 3           ;STATUS REGISTER
0005                RA          EQU 5           ;I/O PORT A
0006                RB          EQU 6           ;I/O PORT B
0008                TIMER1      EQU 10          ;COUNTER FOR DELAY
000C                CSTAT       EQU 14          ;CO-ORDINATE STATUS
000D                BSTAT       EQU 15          ;BUTTON STATUS
000E                DATA0       EQU 16          ;
000F                DATA1       EQU 17          ;
0010                DATA2       EQU 20          ;5 BYTE RS232 DATA
0011                DATA3       EQU 21          ;
0012                DATA4       EQU 22          ;
0013                FLAGA       EQU 23          ;GENERAL PURPOSE FLAG
0014                XCOUNT      EQU 24          ;X-MOVEMENT COUNTER
0015                YCOUNT      EQU 25          ;Y-MOVEMENT COUNTER
0016                FLAGB       EQU 26          ;GENERAL PURPOSE FLAG
0018                COUNT       EQU 30          ;GENERAL PURPOSE COUNTER
0019                DATA_AREA   EQU 31          ;FOR TEMP. STORAGE
                    ;
                    ;_____________________
                    ;       BIT ASSIGNMENT
                    ;_____________________
                    ;
0000                YC          EQU 0           ;Y-CLOCK PIN
0001                YD          EQU 1           ;Y-DATA PIN
0001                UP          EQU 1           ;MOVING UP FLAG
0002                XC          EQU 2           ;X-CLOCK PIN
0003                XD          EQU 3           ;X-DATA PIN
0003                RI          EQU 3           ;MOVING RIGHT FLAG
0000                BU1         EQU 0           ;BUTTON #1 PIN
0002                BU2         EQU 2           ;BUTTON #2 PIN
0000                CA          EQU 0           ;CARRY FLAG
0007                RD          EQU 7           ;RECEIVED DATA PIN TO RS232
0002                ZERO_AREA   EQU 2           ;ZERO FLAG
0002                TR          EQU 2           ;TIGGER FLAG
                    ;
                    ;=========================================
                    ;       SUBROUTINES
                    ;=========================================
                    ;
                    ;*****************************************
                     ORG 0
                    ;*****************************************
                    ;
                    ;=========================================
                    ; DELAY A BIT TIME AND CHECK XC & YC STATUS
                    ;=========================================
```

```
                    BIT
0000 0745                BTFSS    RA,XC            ;XC = 1 ?
0001 0A0A                GOTO     BIT0
0002 064C                BTFSC    CSTAT,XC         ;(XC=1)
0003 0A11                GOTO     BITY             ;(XC ALWAYS = 1)
0004 02B4                INCF     XCOUNT           ;(XC —|__)
0005 0476                BCF      FLAGB,RI         ;DEFAULT LEFT
0006 0765                BTFSS    RA,XD            ;LEFT / RIGHT ?
0007 0A11                GOTO     BITY
0008 0576                BSF      FLAGB,RI
0009 0A11                GOTO     BITY
                    BIT0
000A 074C                BTFSS    CSTAT,XC         ;(XC=0)
000B 0A11                GOTO     BITY             ;(XC ALWAYS = 0)
000C 02B4                INCF     XCOUNT           ;(XC __|—)
000D 0476                BCF      FLAGB,RI         ;DEFAULT LEFT
000E 0665                BTFSC    RA,XD            ;LEFT / RIGHT ?
000F 0A11                GOTO     BITY
0010 0576                BSF      FLAGB,RI
                    BITY
0011 0705                BTFSS    RA,YC            ;YC = 1 ?
0012 0A1B                GOTO     BITY0
0013 060C                BTFSC    CSTAT,YC         ;(YC=1)
0014 0A22                GOTO     BITDY            ;(YC ALWAYS = 1)
0015 02B5                INCF     YCOUNT           ;(YC —|__)
0016 0436                BCF      FLAGB,UP         ;DEFAULT DOWN
0017 0725                BTFSS    RA,YD            ;DOWN / UP ?
0018 0A22                GOTO     BITDY
0019 0536                BSF      FLAGB,UP
001A 0A22                GOTO     BITDY
                    BITY0
001B 070C                BTFSS    CSTAT,YC         ;(YC=0)
001C 0A22                GOTO     BITDY            ;(YC ALWAYS = 0)
001D 02B5                INCF Y   COUNT            ;(YC __|—)
001E 0436                BCF      FLAGB,UP         ;DEFAULT DOWN
001F 0625                BTFSC    RA,YD            ;DOWN / UP ?
0020 0A22                GOTO     BITDY
0021 0536                BSF      FLAGB,UP
                    BITDY
0022 0205                MOVF     RA,W             ;SAVE COOR. STATUS
0023 002C                MOVWF    CSTAT
0024 0CC1                MOVLW    193D             ;0.833 MS DELAY
0025 0028                MOVWF    TIMER1
                    BITD0
0026 0000                NOP
0027 02E8                DECFSZ   TIMER1
0028 0A26                GOTO     BITD0
0029 0800                RETLW    0
                    ;
                    ;================================================
                    ;
                    ;************************************************
                    ;*      SUBROUTINE TO SEND A BYTE               *
                    ;*      AS RS232C FORMAT 8,N,1                  *
                    ;************************************************
                    ;
                    BYTE
002A 0078                CLRF     COUNT            ;RESET 8 BIT COUNT
002B 0753                BTFSS    FLAGA,TR         ;ANY TRIGGER
002C 0A2E                GOTO     BYTE0
002D 04E6                BCF      RB,RD            ;LOW RD FOR START BIT
                    BYTE0
002E 0900                CALL     BIT
                    BYTE1
002F 0753                BTFSS    FLAGA,TR         ;ANY TRIGGER ?
0030 0A37                GOTO     BYTE3
0031 0339                RRF      DATA_AREA        ;SHIFT DATA TO CARRY
0032 0703                BTFSS    STATUS,CA        ;0 / 1 ?
0033 0A36                GOTO     BYTE2
```

```
0034 05E6                BSF    RB,RD              ;SEND A 1
0035 0A37                GOTO   BYTE3
                    BYTE2
0036 04E6                BCF    RB,RD              ;SEND A 0
                    BYTE3
0037 0900                CALL   BIT
0038 02B8                INCF   COUNT
0039 0778                BTFSS  COUNT,3            ;COUNT = 8 ?
003A 0A2F                GOTO   BYTE1
003B 0753                BTFSS  FLAGA,TR           ;ANY TRIGGER ?
003C 0A42                GOTO   BYTE4
003D 04E6                BCF    RB,RD              ;SEND SENT BIT
003E 0900                CALL   BIT
003F 05E6                BSF    RB,RD
0040 0900                CALL   BIT
0041 0A44                GOTO   BYTE5
                    BYTE4
0042 0900                CALL   BIT
0043 0900                CALL   BIT
                    BYTE5
0044 0800                RETLW  0
                    ;
                    ;==========================================
                    ;       RESET ENTRY
                    ;==========================================
                    ;
                    INIT
0045 0CC1                MOVLW  B'11000001'        ;DISABLE WATCH DOG
0046 0002                OPTION
0047 0C0F                MOVLW   B'00001111'       ;INIT RB0~3 BE INPUTS
0048 0006                TRIS RB                    ;RB4~7 BE OUTPUTS
0049 0CFF                MOVLW  B'11111111'        ;INIT RA0~3 BE INPUTS
004A 0005                TRIS   RA
004B 05E6                BSF    RB,RD              ;HIGH RD PIN
004C 0246                COMF   RB,W               ;GET INIT BUTTON INPUTS
004D 0E05                ANDLW  B'00000101'
004E 0D80                IORLW  B'10000000'
004F 002D                MOVWF  BSTAT
0050 002E                MOVWF  DATA0
0051 0205                MOVF   RA,W
0052 002C                MOVWF  CSTAT
0053 0073                CLRF   FLAGA              ;CLEAR TR FLAG
0054 0074                CLRF   XCOUNT             ;RESET XCOUNT & YCOUNT
0055 0075                CLRF   YCOUNT
                    SCAN
0056 006F                CLRF   DATA1              ;UPDATE X,Y MOVEMENT DATA
0057 0070                CLRF   DATA2
0058 0071                CLRF   DATA3
0059 0072                CLRF   DATA4
005A 0214                MOVF   XCOUNT,W           ;XCOUNT = 0 ?
005B 0743                BTFSS  STATUS,ZERO_AREA
005C 0A80                GOTO   WRITX
                    SCANA
005D 0215                MOVF Y COUNT,W            ;YCOUNT = 0 ?
005E 0743                BTFSS  STATUS,ZERO_AREA
005F 0A92                GOTO   WRITY
                    SCANB
0060 0246                COMF   RB,W               ;BUTTON STATUS CHANGE ?
0061 0E05                ANDLW  B'00000101'
0062 0D80                IORLW  B'10000000'
0063 00AD                SUBWF  BSTAT
0064 0643                BTFSC  STATUS,ZERO_AREA   ;IF CHANGE THEN TRIGGER
0065 0A6B                GOTO   SCANC              ;(NO CHANGE)
0066 0553                BSF    FLAGA,TR           ;(CHANGE) SET TRIGGER FLAG
0067 0246                COMF   RB,W               ;FORMAT BUTTON STATUS DATA
0068 0E05                ANDLW  B'00000101'
0069 0D80                IORLW  B'10000000'
```

```
006A 002E                MOVWF   DATA0
                    SCANC
006B 0246                COMF    RB,W
006C 0E05                ANDLW   B'00000101'
006D 0D80                IORLW   B'10000000'
006E 002D                MOVWF   BSTAT
006F 020E                MOVF    DATA0,W          ;SEND DATA0,1,2,3,4 TO HOST
0070 0039                MOVWF   DATA_AREA
0071 092A                CALL    BYTE
0072 020F                MOVF    DATA1,W
0073 0039                MOVWF   DATA_AREA
0074 092A                CALL    BYTE
0075 0210                MOVF    DATA2,W
0076 0039                MOVWF   DATA_AREA
0077 092A                CALL    BYTE
0078 0211                MOVF    DATA3,W
0079 0039                MOVWF   DATA_AREA
007A 092A                CALL    BYTE
007B 0212                MOVF    DATA4,W
007C 0039                MOVWF   DATA_AREA
007D 092A                CALL    BYTE
007E 0453                BCF     FLAGA,TR         ;CLEAR TRIGGER FLAG
007F 0A56                GOTO    SCAN
                    ;
                    WRITX
0080 0553                BSF     FLAGA,TR         ;SET TRIGGER FLAG
0081 0C40                MOVLW   40H              ;IF XCOUNT > 64 THEN XCOUNT <-64
0082 0094                SUBWF   XCOUNT,W
0083 0603                BTFSC   STATUS,CA
0084 0A8D                GOTO    WRITR
                    WRITS
0085 0776                BTFSS   FLAGB,RI         ;LEFT / RIGHT ?
0086 0A90                GOTO    WRITL
0087 0274                COMF    XCOUNT           ;(RIGHT) NEG XCOUNT
0088 0294                INCF    XCOUNT,W
                    WRITA
0089 002F                MOVWF   DATA1
008A 0031                MOVWF   DATA3
008B 0074                CLRF    XCOUNT           ;RESET XCOUNT
008C 0A5D                GOTO    SCANA
                    ;
                    WRITR
008D 0C40                MOVLW   40H              ;XCOUNT <- 64
008E 0034                MOVWF   XCOUNT
008F 0A85                GOTO    WRITS
                    ;
                    WRITL
0090 0214                MOVF    XCOUNT,W         ;(LEFT)
0091 0A89                GOTO    WRITA
                    ;
                    WRITY
0092 0553                BSF     FLAGA,TR         ;SET TRIGGER FLAG
0093 0C40                MOVLW   40H              ;IF YCOUNT > 64 THEN YCOUNT <-64
0094 0095                SUBWF   YCOUNT,W
0095 0603                BTFSC   STATUS,CA
0096 0A9F                GOTO    WRITV
                    WRITW
0097 0736                BTFSS   FLAGB,UP         ;DOWN / UP ?
0098 0AA2                GOTO    WRITD
0099 0275                COMF    YCOUNT           ;(UP) NEG YCOUNT
009A 0295                INCF    YCOUNT,W
                    WRITB
009B 0030                MOVWF   DATA2
009C 0032                MOVWF   DATA4
009D 0075                CLRF    YCOUNT           ;RESET YCOUNT
009E 0A60                GOTO    SCANB
                    ;
```

```
                    WRITV
009F 0C40                MOVLW   40H                ;YCOUNT <- 64
00A0 0035                MOVWF   YCOUNT
00A1 0A97                GOTO    WRITW
                    ;
                    WRITD
00A2 0215                MOVF    YCOUNT,W           ;(DOWN)
00A3 0A9B                GOTO    WRITB
                    ;
                    ;=========================================
                    ;       RESET ENTRY
                    ;=========================================
                    ;
                     ORG 777
01FF 0A45                GOTO INIT                  ;JUMP TO PROGRAM STARTING
                    ;
                    END
                    ;
                    ;*****************************************


Errors   :    0
Warnings :    0
```

NOTES:

AN531

Intelligent Remote Positioner (Motor Control)

Author: Steven Frank - Vesta Technology Inc.

2

INTRODUCTION

The excellent cost/performance ratio of the PIC16C5X are well suited for a low-cost proportional D.C. actuator controller. This application note depicts a design of a remote intelligent positioning system using a D.C. motor (up to 1/3 hp) run from 12 to 24 V. The position accuracy is one in eight bits or 0.4%. The PIC16C5X receives its command and control information via a MICROWIRE™ serial bus. However, any serial communication method is applicable.

IMPLEMENTATION

The PIC16C5X based controller receives commands from a host, compares them to the actual position, calculates the desired motor drive level and then pulses a full H-bridge (Figure 2). In this way it serves as a remote intelligent positioner, driving the load until it has reached the commanded position. It can be used to control any proportional D.C. actuator i.e. D.C. motor or proportional valve.

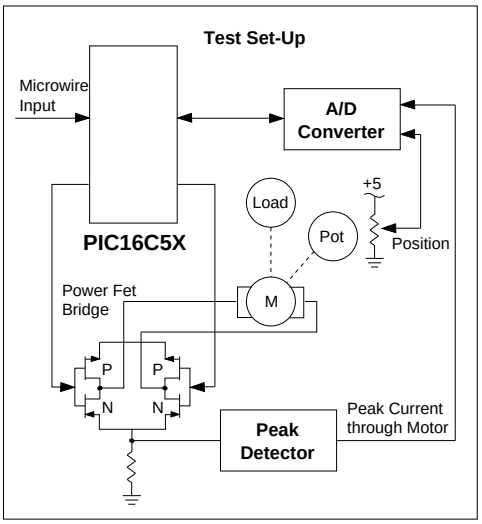
This system is ideally suited to remotely position valves and machinery. It can be used with D.C. motors to easily automate manual equipment. Because of the 5-wire serial interface, the positioner can be installed near its power supply and load. The remote intelligent positioner can then be linked to the central control processor by a small diameter easily routed cable. Since the positioner is running its own closed-loop PID algorithm (Figure 3), the host central processor need only send position commands and is therefore free to service the user interface, main application software and command many remote positioners.

The limit switch inputs provide a safety net this keeps the system from destroying itself in the event that the feedback device is lost. The optional current sense input can be used to determine if the load has jammed and prevent overheating of the actuator and drive electronics.

The commanded positions are presented to the PIC16C5X via a microwire type protocol at bit-rates of up to 50kb/s for the 4 MHz part. As currently implemented in this application note, the position request is the only communication. There are several variable locations available and they could be utilized to allow down-loading of the loop gain parameters, reading positioner information, or for setting a current limit. The host that is sending the position request must set the chip select low, and wait for the PIC16C5X to raise the "busy" (DO) line high. At this point, eight data bits can be clocked into the PIC16C5X. The requested position is sent most significant bit first and can be any 8-bit value. Values 1 through 255 represent valid positions with 0 being reserved for drive disable.

The PIC16C5X acquires data by way of a microwire A/D converter. This part was chosen for low cost yet it provides adequate performance. The second channel of the A/D is shown hooked up to a peak current detector. If the user desired, the PIC16C5X could monitor and protect the motor from overcurrent by monitoring this information.

FIGURE 1 - BLOCK DIAGRAM



MICROWIRE™ is a trademark of National Semiconductor Corporation.

The H-bridge power amplifier will deliver 10 or more amps at up to 24 volts when properly heat-sinked. It is wired for a modified 4-quadrant mode of operation. One leg of the bridge is used to control direction and the other leg pulses the low FET and the high FET alternately to generate the desired duty-cycle. In this way the system will operate well to produce a desired "speed" without the use of a separate speed control loop. This allows use of the PIC16C5X to control the PID algorithm for position directly while having reasonable speed control. The capacitance at the gates of the FETs combined with the impedance of the drive circuits provides for turn-off of the upper FET before the lower FET turns on... an important criteria.

The PID algorithm itself is where most of the meat of this application note is located so let's look at it more closely. The Algorithm is formed by summing the contribution of three basic components. The first calculation is the error for that is what the other terms are based on.

The error is the requested position minus the actual position. It is a signed number whose magnitude can be 255. In order not to lose resolution, the error is stored as an 8-bit magnitude with the sign stored separately in the FLAGS register under ER_SGN. This allows us to resolve a full signed 8-bit error with 8-bit math.

The proportional term is merely the algebraic difference of the requested position minus the actual position. It is scaled by a gain term (Kp) called the "proportional gain". The sign of this term is important for it tells the system which direction it must drive to correct the error. The proportional term is limited to $\pm$ 100. Increasing the proportional gain term will improve the dynamic and static accuracy of the system. Increasing it too much will cause oscillations.

The next term that gets calculated is the Integral term. This term is traditionally formed by integrating the error over time. In this application it is done by integrating the Ki term over time. When the error is zero, no integration is performed. This is a more practical way to handle a potentially large number in 8-bit math. By increasing the Ki term the D.C. or static gain of the system is improved. Increasing the integral gain too much can lead to low frequency oscillations.

The differential term (Kd) is a stabilizing term that helps keep the integral and proportional terms from overdriving the system through the desired position and thus creating oscillations. As you use more proportional and integral gain you will need more differential gain as well. The differential gain is calculated by looking at the rate of change of the positional error with respect to time. It is actually formed as "delta error/delta time" with the delta time being a program cycle.

The three terms are summed algebraically and scaled to produce a percentage speed request between 0 and 100%. The sign of the sum is used to control the H-bridge direction. The loop calculations run approximately 20 times per second on a 4 MHz part. This yields sufficient gain-bandwidth for most positioning applications. If higher system performance is desired, the number of pulses can be reduced to 20 and a 16 MHz PIC16C5X can be used. Your Loop gains (Kp, Ki, Kd) will have to be recalculated, but the system sample rate will be increased to 400 Hz. This should be sufficient to control a system that has a response time of 20 milliseconds or more.

The key to using the PIC16C5X series parts for PID control and PWM generation is to separate the two into separate tasks. There is simply not the hardware support or the processing speed to accurately do both concurrently. It is fortunate therefore that it is not necessary to do both concurrently. The systems that are generally controlled can be stabilized with a much lower information update rate than the PWM frequency. This supports the approach of calculating the desired percentage, outputting the PWM for a period of time and then recalculating the new desired percentage. Utilizing this technique the inexpensive PIC16C5X can implement PID control, PWM generation and still have processing time left over for monitor or communication functions.

About the Author:

Steven Frank has been designing analog and digital control systems for ten years. His background is in medical and consumer electronics. He has received numerous patents in control systems and instrumentation. At Vesta Technology Inc., Mr. Frank works with a number of engineers on custom embedded control systems designs. Vesta Technology Inc. is a provider of embedded control systems from an array of standard products and designs. Vesta offers custom design services and handles projects from concept to manufacturing.

FIGURE 2 - PROGRAM FLOW CHART

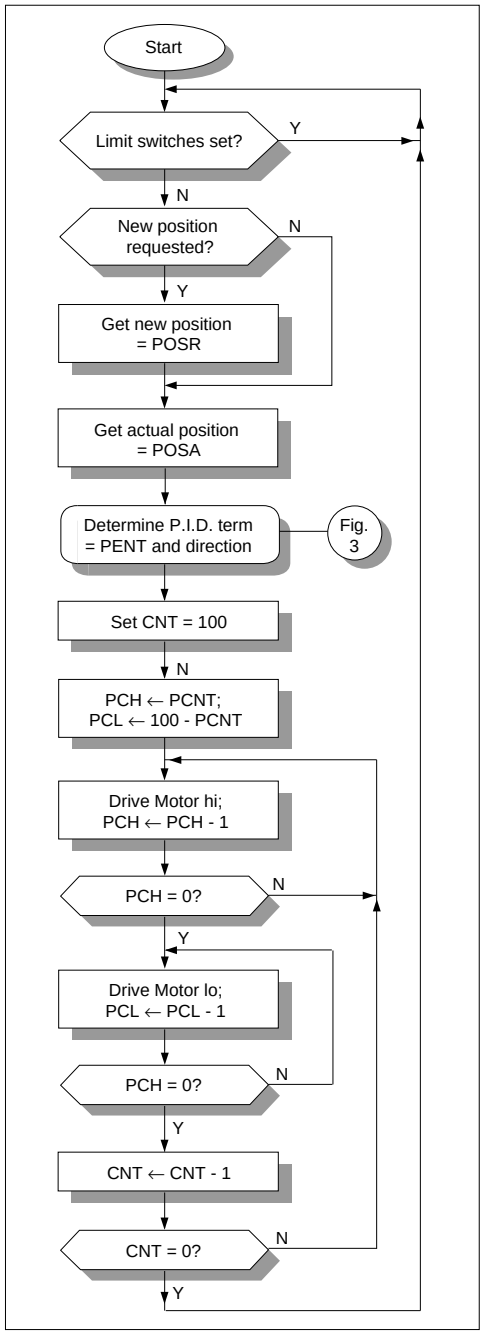


FIGURE 3 - P.I.D. ALGORITHM FLOW CHART

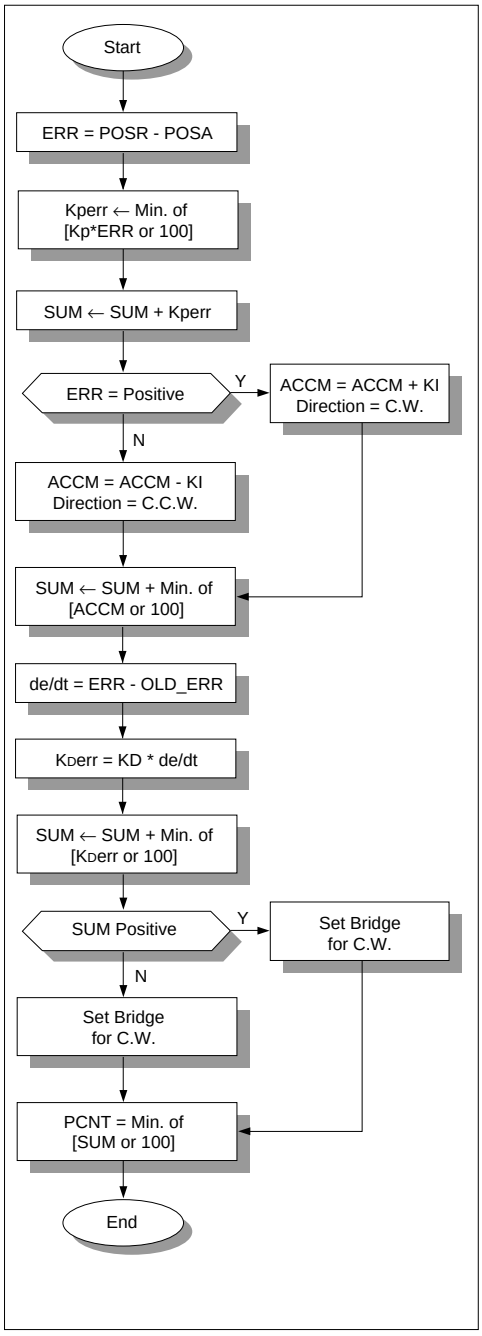
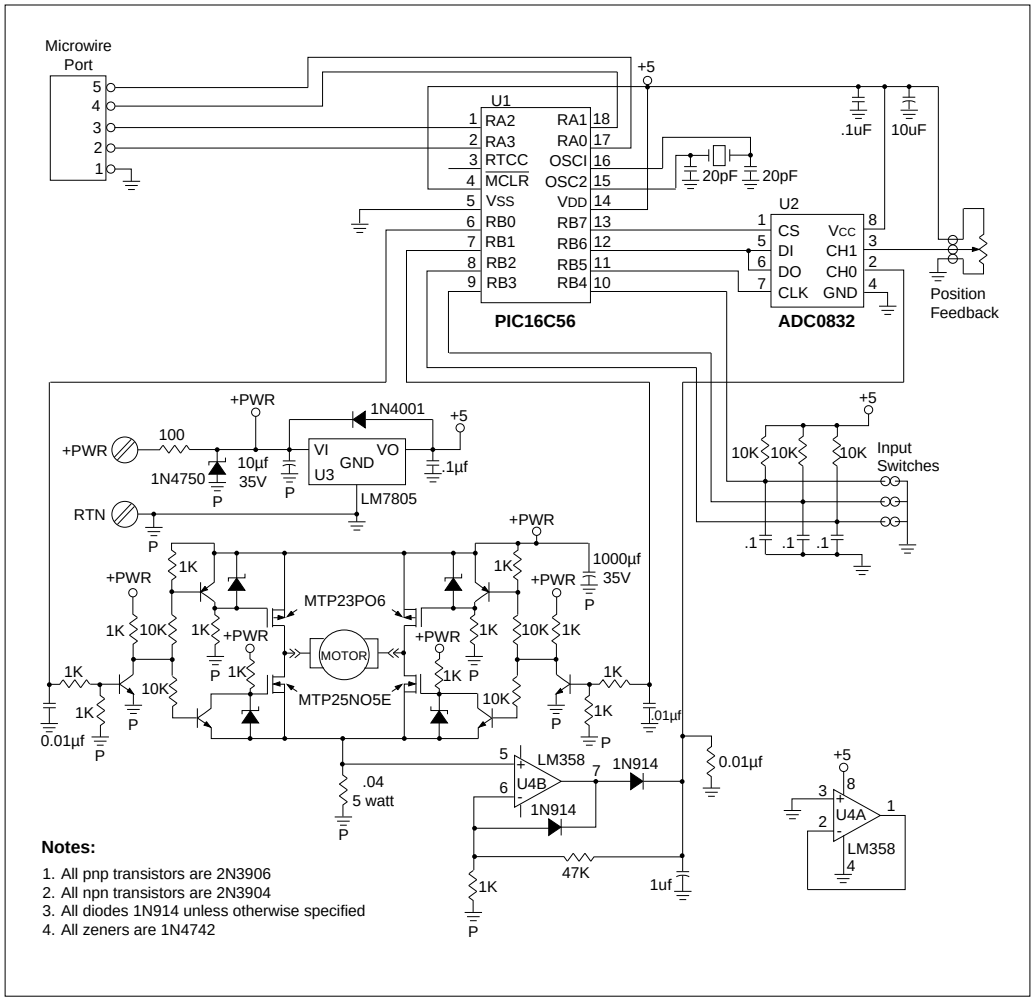


FIGURE 4 - SCHEMATIC



```
MPASM B0.54                                                    PAGE  1


                ;
                ;       mw8pos.asm
                ;
                        LIST P=16C56
                ;*************************************************************
                ; REV. A        Original release  1/10/92 srf
                ;
                ; *************************************************************
                ;
                ;REGISTER EQUATES
                ;
0000             W       EQU     0
0000             PNTR    EQU     00H             ; CONTENTS OF POINTER
0019             FLAGS   EQU     19H             ; USE THIS VARIABLE LOCATION AS FLAGS
                                                 ; 0 BIT IS SIGN OF ERROR 1 IS NEGATIVE
                                                 ; 1 BIT IS SIGN OF ERROR ACCUMULATOR
                                                 ; 2 BIT IS SIGN OF THE DE/DE TERM
                                                 ; 3 BIT IS DIRECTION 0 IS CW
                                                 ; 4 BIT IS SIGN OF THE OLD ERROR
0003             STATUS  EQU     03H
0003             SWR     EQU     03H             ; STATUS WORD REGISTER
                                                 ; 0 = CARRY
                                                 ; 1 = DC
                                                 ; 2 = Z, SET IF RESULT IS ZERO
0004             FSR     EQU     04H             ; FILE SELECT REGISTER
0005             PORTA   EQU     05H             ; I/O REG (A0-A3), (A4-A7 DEF=0)
0006             PORTB   EQU     06H             ; I/O REGISTER(B0-B7)
0007             HI      EQU     07H             ; NUMBER OF HIGH MICROSECONDS
0008             LO      EQU     08H             ; NUMBER OF LOW MICROSECONDS
0009             PCNT    EQU     09H             ; PERCENT DUTYCYCLE REQUEST
000A             HI_T    EQU     0AH             ; COUNTER FOR USECONDS LEFT/PULSE HI
000B             LO_T    EQU     0BH             ; COUNTER FOR USECONDS LEFT/PULSE LO
000C             ERROR   EQU     0CH             ; HOLDER FOR THE POSITIONAL ERROR
                                                 ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                                                 ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
000D             SUMLO   EQU     0DH             ; PROGRESSIVE SUM OF THE PID TERMS
000E             ACCUM   EQU     0EH             ; ERROR ACCUMULATOR
000F             ERR_O   EQU     0FH             ; ERROR HISTORY USED FOR de/dt
                                                 ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                                                 ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
0010             POSR    EQU     10H             ; POSITIONAL REQUEST
0011             POSA    EQU     11H             ; ACTUAL POSITION
0012             CYCLES  EQU     12H             ; COUNTER FOR CYCLES OUT

0013             mulcnd  equ     13H             ; 8 bit multiplicand
0013             ACCaLO  EQU     13H             ; same location used for the add routine
0014             mulplr  equ     14H             ; 8 bit multiplier
0014             ACCbLO  EQU     14H             ; same location used for the add routine
0015             H_byte  equ     15H             ; High byte of the 16 bit result
0015             ACCaHI  EQU     15H             ; same location used for the add routine
0016             L_byte  equ     16H             ; Low byte of the 16 bit result
0016             ACCbHI  EQU     16H             ; same location used for the add routine
0017             count   equ     17H             ; loop counter
0018             SUMHI   EQU     18H             ; HIGH BYTE OF THE LOOP SUM


                ; PORT ASSIGNMENTS AND CONSTANTS

0000             PWMCW   EQU     0               ; CLOCKWISE PWM OUTPUT BIT
0001             PWMCCW  EQU     1               ; COUNTERCLOCKWISE PWM OUTPUT BIT
0000             CARRY   EQU     0               ; CARRY BIT IN THE STATUS REGISTER
0002             Z       EQU     2               ; THE ZERO BIT OF THE STATUS REGISTER
0001             Same    equ     1               ;
```

```
0000                ER_SGN  EQU     0               ; SIGN BIT FOR THE ERROR IN FLAG REGISTER
0001                AC_SGN  EQU     1               ; SIGN BIT FOR THE ERROR ACCUMULATOR
0002                DE_SGN  EQU     2               ; SIGN BIT FOR DE/DT
0004                OER_SGN EQU     4               ; SIGN BIT FOR THE OLD ERROR
0030                KP      EQU     30              ; PROPORTIONAL GAIN
0002                KI      EQU     2               ; INTEGRAL GAIN
0020                KD      EQU     20              ; DIFFERENTIAL GAIN
0003                DIR     EQU     3               ; THE DIRECTION FLAG
0007                CSN     EQU     7               ; CHIP SELECT NOT ON A/D
0006                BV      EQU     6               ; DATA LINE FOR THE A/D
0005                CK      EQU     5               ; CLOCK LINE FOR THE A/D
0002                MWDO    EQU     2               ; MICROWIRE DATA OUT FROM POSITIONER
0001                MWDI    EQU     1               ; MICROWIRE DATA IN TO POSITIONER
0000                MWCS    EQU     0               ; MICROWIRE CHIP SELECT TO POSITIONER
0003                MWCK    EQU     3               ; MICROWIRE CLOCK IN TO POSITIONER

                    ;***** MACROS ************************************************
                    ;
                    CLKUP   MACRO                   ; clock up macro for the microwire
                      BSF     PORTB,CK              ; data acquisition from the a/d
                      NOP
                      ENDM

                    CLKDN   MACRO                   ; clock down macro for the microwire
                      BCF     PORTB,CK              ; data acquisition from the a/d
                      NOP
                      ENDM

                    GET_BIT MACRO                   ; ** FOR RECEIVING A/D DATA **
                      BCF     SWR,CARRY
                      BSF     PORTB,CK              ; SET CLOCK BIT HIGH
                      BTFSC   PORTB,BV              ; LOOK AT DATA COMMING IN
                      BSF     SWR,CARRY             ; SET THE CARRY FOR A 1
                      RLF     POSA                  ; ROTATE THE W REG LEFT
                      BCF     PORTB,CK              ; SET THE CLOCK LOW
                      NOP                           ; DELAY
                      ENDM

0000 0B88                     GOTO    CLRREG

                    ;***** MATH ROUTINES *************************************
                    ;
                    ; **** 8 BIT MULTIPLY ********
                    ; ****************************         Begin Multiplier Routine
0001 0075           mpy_S     clrf    H_byte
0002 0076                     clrf    L_byte
0003 0C08                     movlw   8
0004 0037                     movwf   count
0005 0213                     movf    mulcnd,w
0006 0403                     bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
0007 0334           loop      rrf     mulplr
0008 0603                     btfsc   STATUS,CARRY
0009 01F5                     addwf   H_byte,Same
000A 0335                     rrf     H_byte,Same
000B 0336                     rrf     L_byte,Same
000C 02F7                     decfsz  count
000D 0A07                     goto    loop
000E 0800                     retlw   0

                    ; ****************************
                    ; DOUBLE PRECISION ADD AND SUBTRACT ( ACCb-ACCa->ACCb )

000F 0917           D_sub     call    neg_A           ; At first negate ACCa, then add

                    ;***************
                    ; Double Precision Addition ( ACCb+ACCa->ACCb )

0010 0213           D_add     movf    ACCaLO,W
0011 01F4                     addwf   ACCbLO          ; add lsb
```

```
0012 0603                   btfsc   STATUS,CARRY    ; add in carry
0013 02B6                   incf    ACCbHI
0014 0215                   movf    ACCaHI,W
0015 01F6                   addwf   ACCbHI          ; add msb
0016 0800                   retlw   00
                    ;
                    ;
0017 0273           neg_A   comf    ACCaLO          ; negate ACCa
0018 02B3                   incf    ACCaLO
0019 0643                   btfsc   STATUS,Z
001A 00F5                   decf    ACCaHI
001B 0275                   comf    ACCaHI
001C 0800                   retlw   00

                    ; ****************************************
                    ; divide by 16 and limit to 100 Decimal

                    SHIFT   MACRO
                      BCF     SWR,CARRY
                      RRF     L_byte
                      BCF     SWR,CARRY
                      RRF     H_byte
                      BTFSC   SWR,CARRY
                      BSF     L_byte,7
                      ENDM
                    DIV_LMT
                      SHIFT
001D 0403                   BCF     SWR,CARRY
001E 0336                   RRF     L_byte
001F 0403                   BCF     SWR,CARRY
0020 0335                   RRF     H_byte
0021 0603                   BTFSC   SWR,CARRY
0022 05F6                   BSF     L_byte,7

                      SHIFT
0023 0403                   BCF     SWR,CARRY
0024 0336                   RRF     L_byte
0025 0403                   BCF     SWR,CARRY
0026 0335                   RRF     H_byte
0027 0603                   BTFSC   SWR,CARRY
0028 05F6                   BSF     L_byte,7

                      SHIFT
0029 0403                   BCF     SWR,CARRY
002A 0336                   RRF     L_byte
002B 0403                   BCF     SWR,CARRY
002C 0335                   RRF     H_byte
002D 0603                   BTFSC   SWR,CARRY
002E 05F6                   BSF     L_byte,7

                      SHIFT
002F 0403                   BCF     SWR,CARRY
0030 0336                   RRF     L_byte
0031 0403                   BCF     SWR,CARRY
0032 0335                   RRF     H_byte
0033 0603                   BTFSC   SWR,CARRY
0034 05F6                   BSF     L_byte,7


                    LMT100
0035 0C01                   MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0036 0095                   SUBWF   H_byte,0        ; IF THERE IS ANYTHING THERE, IF NOT,
0037 0703                   BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0038 0A3C                   GOTO    L8_E            ; OTHERWISE GIVE THE LOW BYTE A FULL
0039 0C64                   MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
003A 0036                   MOVWF   L_byte          ; TO 100
003B 0A42                   GOTO    LMT_EXIT
                    L8_E
003C 0C64                   MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
```

```
003D 0096                   SUBWF   L_byte,0        ; 100 DECIMAL
003E 0703                   BTFSS   SWR,CARRY
003F 0A42                   GOTO    LMT_EXIT
0040 0C64                   MOVLW   64H
0041 0036                   MOVWF   L_byte
                    LMT_EXIT
0042 0800                   RETLW   00
                    ;
                    ;THE ROUTINE CALCTIMES DOES THE FOLLOWING: PCNT = DUTY CYCLE IN %
                    ; 100 - PCNT —> LO  AND PCNT —> HI. ZERO VALUES IN EITHER LO OR HI
                    ;ARE FORCED TO 1.
                    CALCTIMES
0043 0209                   MOVF    PCNT,W          ; PUT REQUESTED % INTO W REGISTER
0044 0027                   MOVWF   HI              ; COPY ON MICROSECONDS IN TO HI TIME
0045 0C64                   MOVLW   64H
0046 0028                   MOVWF   LO
0047 0209                   MOVF    PCNT,0
0048 00A8                   SUBWF   LO,1            ; LEAVE 100-HI TIME IN LO TIME
0049 0207                   MOVF    HI,0            ; INSPECT THE HIGH TIME
004A 0643                   BTFSC   SWR,2           ; IF ITS IS ZERO
004B 02A7                   INCF    HI,1            ; INCREMENT IT
004C 0208                   MOVF    LO,0            ; INSPECT THE LO TIME
004D 0643                   BTFSC   SWR,2           ; IF ITS ZERO
004E 02A8                   INCF    LO,1            ; INCREMENT IT
004F 0800                   RETLW   00


                    ;*****************************************************************
                    BEGIN
0050 0000                   NOP                     ; STUBBED BEGINNING


                    ;****CHECKING THE LIMIT SWITCHES AND CHECKING FOR MW***************
                    ; This will check the switch inputs for closure and will terminate
                    ; pulsing is one is closed.  It doesn’t distinguish between the switches
                    ; so they are not dedicated to cw end and ccw end.

                    SW_TRAP
0051 0004                   CLRWDT
0052 0746                   BTFSS   PORTB,2         ; THIS WILL TEST ALL THREE OF THE
0053 0A51                   GOTO    SW_TRAP         ; SWITCH INPUTS.  IF ANY ONE IS
0054 0766                   BTFSS   PORTB,3         ; SET THEN EXECUTION OF THE CODE
0055 0A51                   GOTO    SW_TRAP         ; WILL BE LIMITED TO LOOKING FOR
0056 0786                   BTFSS   PORTB,4         ; IT TO BE CLEARED
0057 0A51                   GOTO    SW_TRAP


                    ;****RECEIVING THE POSITIONAL REQUEST*****************************
                    ; The host system that wishes to send positional requests to the positioner
                    ; servo makes its desire known by setting the chip select to the positioner
                    ; low.  It then monitors the busy (Data Out) line from the positioner.  When
                    ; the positioner sets the busy line high, the host may begin sending its 8
                    ; request.  The data bits should be valid on the rising edge of the clock.
                    ; After 8 bits have been received by the positioner it will begin operation
                    ; to send the system to the received position.  It can be interrupted at any
                    ; point during the positioning process by the host sending a new command.
                    ; opportunity to update the command is issued every 100 pwm pulses (every 50
                    ; milliseconds).
                    ; If the host sends a zero positional command the positioner will stop the
                    ; system and remain inactive.
                    ; If the host does not successfully complete a microwire transmission of 8
                    ; data bits the watchdog timer will trip and reset the system to an inactive
                    ; “stopped” state.


                    REC_MW
0058 0C0B                   MOVLW   0BH             ; RESET THE PORT FOR THREE INPUTS
```

```
0059 0005                TRIS    PORTA           ; AND ONE OUTPUT
005A 0445                BCF     PORTA,MWDO      ; SET THE DATA OUT LOW FOR BUSY
005B 0C20                MOVLW   20H
005C 0037                MOVWF   count
                  WATCH_CS
005D 0705                BTFSS   PORTA,MWCS      ; CHECK FOR INCOMMING REQUESTS
005E 0A62                GOTO    REC_CMD         ; RECEIVE A NEW POSITION REQUEST
005F 02F7                DECFSZ  count,1
0060 0A5D                GOTO    WATCH_CS
0061 0A71                GOTO    REC_EXIT        ; NO REQUEST WAS MADE IN THE TIME ALLOTED
                  REC_CMD
0062 0545                BSF     PORTA,MWDO      ; SET THE DATA OUT HIGH FOR “OK TO SEND”
0063 0C08                MOVLW   8H              ; SET TO RECEIVE 8 BITS
0064 0037                MOVWF   count
                  WAIT_UP
0065 0765                BTFSS   PORTA,MWCK      ; WAIT FOR A RISING EDGE
0066 0A65                GOTO    WAIT_UP
0067 0403                BCF     SWR,CARRY       ; RESET THE CARRY TO A DEFAULT ZERO
0068 0625                BTFSC   PORTA,MWDI      ; READ THE DATA IN
0069 0503                BSF     SWR,CARRY       ; SET THE CARRY FOR A ONE
006A 0370                RLF     POSR,1          ; ROTATE THE BIT INTO THE POSITION REQ.
006B 02F7                DECFSZ  count,1         ; DECREMENT THE BIT COUNTER
006C 0A6E                GOTO    WAIT_DN         ; WAIT FOR THE FALLING EDGE
006D 0A71                GOTO    REC_EXIT        ; LAST BIT RECEIVED
                  WAIT_DN
006E 0665                BTFSC   PORTA,MWCK      ; CHECK THE INCOMMING CLOCK
006F 0A6E                GOTO    WAIT_DN         ; IF IT IS STILL HIGH WAIT FOR IT TO GO LOW
0070 0A65                GOTO    WAIT_UP         ; IF IT GOES LOW GO BACK TO RECEIVE NEXT BIT

                  REC_EXIT
0071 0445                BCF     PORTA,MWDO      ; SET THE BUSY FLAG


                  ;********** CHECK FOR THE DISABLE REQUEST ************************
                  ; Position 0 is considered a request to not drive the system.  In this way
                  ; the positioner will come up from a reset in a safe state and will not
                  ; try to move the system to some arbitrary location.

                  MOVE?
0072 0210                MOVF    POSR,W          ; CHECK THE REQUESTED POSTION
0073 0643                BTFSC   SWR,Z           ; IF IT IS ZERO THEN WAIT FOR A NON-ZERO
0074 0A50                GOTO    BEGIN           ; REQUEST BY BRANCHING BACK TO THE BEGINNING

                  ;****READING THE A/D VALUES*************************************
                  ;
                  ; Read the positional a/d channel (1) and store the value in the actual
                  ; position variable (POSA).
                  ; This is written in line to minimize the use of variables

                  READ_POS
0075 0071                CLRF    POSA            ; CLEAN THE POSITION ACTUAL HOLDER
0076 04E6                BCF     PORTB,CSN       ; SET THE CHIP SELECT LOW TO A/D
0077 0C1C                MOVLW   1CH             ; SET THE DATA LINE TO OUTPUT
0078 0006                TRIS    PORTB           ; FOR SENDING SET-UP BITS
0079 05C6                BSF     PORTB,BV        ; SET FOR “START” BIT
007A 0000                NOP
                    CLKUP                        ; CLOCK IN THE START BIT
007B 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
007C 0000                NOP

                    CLKDN                        ; “
007D 04A6                BCF     PORTB,CK        ; data acquisition from the a/d
007E 0000                NOP

                    CLKUP                        ; CLOCK IN SINGLE-ENDED
007F 05A6                BSF     PORTB,CK        ; data acquisition from the a/d
0080 0000                NOP

                    CLKDN                        ; “
```

```
0081 04A6                     BCF     PORTB,CK        ; data acquisition from the a/d
0082 0000                     NOP

                        CLKUP                         ; CLOCK IN CHANNEL 1
0083 05A6                     BSF     PORTB,CK        ; data acquisition from the a/d
0084 0000                     NOP

                        CLKDN                         ; TO THE MUX
0085 04A6                     BCF     PORTB,CK        ; data acquisition from the a/d
0086 0000                     NOP

0087 0C5C                     MOVLW   5CH             ; SET THE DATA LINE TO INPUT
0088 0006                     TRIS    PORTB           ; TO RECEIVE DATA BITS FROM A/D
                        CLKUP                         ; CLOCK UP TO LET MUX SETTLE
0089 05A6                     BSF     PORTB,CK        ; data acquisition from the a/d
008A 0000                     NOP

                        CLKDN                         ; CLOCK DN TO LET MUX SETTLE
008B 04A6                     BCF     PORTB,CK        ; data acquisition from the a/d
008C 0000                     NOP

                        GET_BIT                       ; GET BIT 7
008D 0403                     BCF     SWR,CARRY
008E 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
008F 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0090 0503                     BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0091 0371                     RLF     POSA            ; ROTATE THE W REG LEFT
0092 04A6                     BCF     PORTB,CK        ; SET THE CLOCK LOW
0093 0000                     NOP                     ; DELAY

                        GET_BIT                       ; BIT 6
0094 0403                     BCF     SWR,CARRY
0095 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
0096 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0097 0503                     BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0098 0371                     RLF     POSA            ; ROTATE THE W REG LEFT
0099 04A6                     BCF     PORTB,CK        ; SET THE CLOCK LOW
009A 0000                     NOP                     ; DELAY
                        GET_BIT                       ; BIT 5
009B 0403                     BCF     SWR,CARRY
009C 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
009D 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
009E 0503                     BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
009F 0371                     RLF     POSA            ; ROTATE THE W REG LEFT
00A0 04A6                     BCF     PORTB,CK        ; SET THE CLOCK LOW
00A1 0000                     NOP                     ; DELAY

                        GET_BIT                       ; BIT 4
00A2 0403                     BCF     SWR,CARRY
00A3 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00A4 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00A5 0503                     BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00A6 0371                     RLF     POSA            ; ROTATE THE W REG LEFT
00A7 04A6                     BCF     PORTB,CK        ; SET THE CLOCK LOW
00A8 0000                     NOP                     ; DELAY

                        GET_BIT                       ; BIT 3
00A9 0403                     BCF     SWR,CARRY
00AA 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00AB 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00AC 0503                     BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00AD 0371                     RLF     POSA            ; ROTATE THE W REG LEFT
00AE 04A6                     BCF     PORTB,CK        ; SET THE CLOCK LOW
00AF 0000                     NOP                     ; DELAY

                         GET_BIT                      ; BIT 2
00B0 0403                     BCF     SWR,CARRY
00B1 05A6                     BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00B2 06C6                     BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
```

```
00B3 0503                   BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00B4 0371                   RLF     POSA            ; ROTATE THE W REG LEFT
00B5 04A6                   BCF     PORTB,CK        ; SET THE CLOCK LOW
00B6 0000                   NOP                     ; DELAY

                      GET_BIT                       ; BIT 1
00B7 0403                   BCF     SWR,CARRY
00B8 05A6                   BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00B9 06C6                   BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00BA 0503                   BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00BB 0371                   RLF     POSA            ; ROTATE THE W REG LEFT
00BC 04A6                   BCF     PORTB,CK        ; SET THE CLOCK LOW
00BD 0000                   NOP                     ; DELAY

                      GET_BIT                        ; BIT 0
00BE 0403                   BCF     SWR,CARRY
00BF 05A6                   BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00C0 06C6                   BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00C1 0503                   BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00C2 0371                   RLF     POSA            ; ROTATE THE W REG LEFT
00C3 04A6                   BCF     PORTB,CK        ; SET THE CLOCK LOW
00C4 0000                   NOP                     ; DELAY

00C5 05E6                   BSF     PORTB,CSN       ; DESELECT THE CHIP


                    ;****************** CALCULATING THE PID TERMS **********************

                    ;****CALCULATE THE ERROR*******
                    ; The error is very simply the signed difference between where the
                    ; system is and where it is supposed to be at a particular instant
                    ; in time.  It is formed by subtracting the actual position from the
                    ; requested position (Position requested - Position actual).  This
                    ; difference is then used to determine the proportional,integral and
                    ; differential term contributions to the output.

                    C_ERR
00C6 0211                   MOVF    POSA,0          ; LOAD THE ACTUAL POSITION INTO W
00C7 0090                   SUBWF   POSR,0          ; SUBTRACT IT FROM THE REQUESTED POSITION
00C8 0603                   BTFSC   SWR,CARRY       ; CHECK THE CARRY BIT TO DETERMINE THE SIGN
00C9 0ACB                   GOTO    PLS_ER          ; ITS POSATIVE  (POSR>POSA)
00CA 0ACE                   GOTO    MNS_ER          ; ITS NEGATIVE  (POSA>POSR)

                    PLS_ER
00CB 002C                   MOVWF   ERROR           ; SAVE THE DIFFERENCE IN “ERROR”
00CC 0419                   BCF     FLAGS,ER_SGN    ; SET THE SIGN FLAG TO INDICATE POSATIVE
00CD 0AD2                   GOTO    CE_EXIT

                    MNS_ER
00CE 0210                   MOVF    POSR,0          ; RE-DO THE SUBTRACTION
00CF 0091                   SUBWF   POSA,0          ; ACTUAL - REQUESTED
00D0 002C                   MOVWF   ERROR           ; STORE THE DIFFERENCE IN “ERROR”
00D1 0519                   BSF     FLAGS,ER_SGN    ; SET THE SIGN FLAG FOR NEGATIVE

                    CE_EXIT
00D2 006D                   CLRF    SUMLO           ; CLEAN OLD VALUES OUT TO PREPARE
00D3 0078                   CLRF    SUMHI           ; FOR THIS CYCLES SUMMATION

                    ;****CALCULATE THE PROPORTIONAL TERM******
                    ; The proportional term is the error times the proportional gain term.
                    ; This term simply gives you more output drive the farther away you are
                    ; from where you want to be (error)*Kp.
                    ; The proportional gain term is a signed term between -100 and 100  The
                    ; more proportional gain you have the lower your system following error
                    ; will be.  The higher your proportional gain, the more integral and
                    ; differential term gains you will have to add to make the system stable.
                    ; The sum is being carried as a 16 bit signed value.

                    C_PROP
```

```
00D4 020C                   MOVF    ERROR,0         ; LOAD THE ERROR TERM INTO W
00D5 0033                   MOVWF   mulcnd          ; MULTIPLY IT BY THE PROPORTIONAL GAIN
00D6 0C30                   MOVLW   KP              ; KP AND THEN SCALE IT DOWN BY DIVIVING
00D7 0034                   MOVWF   mulplr          ; IT DOWN BY 16. IF IT IS STILL OVER
00D8 0901                   CALL    mpy_S           ; 255 THEN LIMIT IT TO 255
00D9 091D                   CALL    DIV_LMT

                    RESTORE_SGN
00DA 0719                   BTFSS   FLAGS,ER_SGN    ; IF THE ERROR SIGN IS NEGATIVE THEN
00DB 0ADE                   GOTO    ADDPROP         ; PUT THE SIGN INTO THE LOW BYTE
00DC 0276                   COMF    L_byte,1
00DD 02B6                   INCF    L_byte,1

                    ADDPROP
00DE 0216                   MOVF    L_byte,W        ; SAVE THE PROPORTIONAL PART
00DF 01ED                   ADDWF   SUMLO,1         ; IN THE SUM
00E0 0603                   BTFSC   SWR,CARRY       ; IF THE ADDITION CARRIED OUT THEN
00E1 02B8                   INCF    SUMHI,1         ; INCREMENT THE HIGH BYTE
00E2 0C00                   MOVLW   0               ; THEN
00E3 06ED                   BTFSC   SUMLO,7         ; SIGN EXTEND TO THE UPPER
00E4 0CFF                   MOVLW   0FF             ; BYTE
00E5 01F8                   ADDWF   SUMHI,1



                    ;****CALCULATE THE INTEGRAL TERM******
                    ; The integral term is an accumulation of the error thus far.  Its purpose
                    ; is to allow even a small error to effect a large change.  It does this
                    ; by adding a small number into an accumulator each cycle through the pro
                    ; Thusly even a small error that exist for a while will build up to a large
                    ; enough number to effect an output sufficient to move the system.  The
                    ; that this integral accumulator has is modulated by the integral gain term
                    ; The integral of the error over time is multiplied be KI and the result is
                    ; contribution to the final summation for determining the output value.  This
                    ; term helps to insure the long-term accuracy of the system is good.  A
                    ; amount is necessary for this purpose but too much will cause oscillations.
                    ; The integral is bounded in magnitude for two purposes.  The first is so
                    ; it never rolls over and changes sign.  The second is that it may saturate
                    ; long moves forcing an excessively large overshoot to “de-integrate” the
                    ; accumulated during the first of the move

                    C_INT
00E6 020C                   MOVF    ERROR,W         ; MOVE THE ERROR INTO THE W REG
00E7 0643                   BTFSC   SWR,Z           ; AND CHECK TO SEE IF IT IS ZERO
00E8 0AFF                   GOTO    ADDINT          ; IF SO THEN DONT CHANGE THE ACCUMULATOR
00E9 0619                   BTFSC   FLAGS,ER_SGN    ; TEST THE FLAGS TO FIND THE POLARITY
00EA 0AEE                   GOTO    MNS_1           ; OF THE ERROR .. 0 POSATIVE 1 NEGATIVE
                    PLS_1
00EB 0C02                   MOVLW   KI              ; IF POSATIVE ADD ONE TO
00EC 01EE                   ADDWF   ACCUM,1         ; THE ERROR ACCUMULATOR
00ED 0AF0                   GOTO    LMTACM          ; THEN LIMIT IT TO +/-100
                    MNS_1
00EE 0C02                   MOVLW   KI              ; IF NEGATIVE THEN SUBTRACT ONE
00EF 00AE                   SUBWF   ACCUM,1         ; FROM THE ERROR ACCUMULATOR
                    LMTACM
00F0 06EE                   BTFSC   ACCUM,7         ; CHECK THE SIGN BIT OF THE ERROR ACCUMULA-
TOR
00F1 0AF9                   GOTO    M_LMT           ; AND DO A POSATIVE OR NEGATIVE LIMIT
                    P_LMT
00F2 0C9C                   MOVLW   9CH             ; FOR THE POSATIVE LIMIT ADD 156 TO THE
00F3 01CE                   ADDWF   ACCUM,0         ; NUMBER AND SEE IF YOU GENERATE A CARRY
00F4 0703                   BTFSS   SWR,CARRY       ; BY CHECKING THE CARRY FLAG
00F5 0AFF                   GOTO    ADDINT          ; IF NOT THEN ITS O.K.
00F6 0C64                   MOVLW   64H             ; IF SO THEN FORCE THE ACCUMULATOR TO
00F7 002E                   MOVWF   ACCUM           ; 100 DECIMAL
00F8 0AFF                   GOTO    ADDINT
                    M_LMT
00F9 0C9C                   MOVLW   9CH             ; FOR THE NEGATIVE LIMIT SUBTRACT 156 FROM
00FA 008E                   SUBWF   ACCUM,0         ; THE NUMBER AND SEE IF YOU GENERATE A
```

```
00FB 0603                   BTFSC   SWR,CARRY       ; NON-CARRY CONDITION INDICATING A ROLL-OVER
00FC 0AFF                   GOTO    ADDINT          ; IF NOT THEN LEAVE THE ACCUMULATOR ALONE
00FD 0C9C                   MOVLW   9CH             ; IF SO THEN LIMIT IT TO -100 BY
00FE 002E                   MOVWF   ACCUM           ; FORCING THAT VALUE IN THE ACCUMULATOR

                    ADDINT
00FF 020E                   MOVF    ACCUM,W         ; ADD THE INTEGRAL ACCUMULATOR TO
0100 01ED                   ADDWF   SUMLO,1         ; THE LOW BYTE OF THE SUM
0101 0603                   BTFSC   SWR,CARRY       ; TEST FOR OVERFLOW, IF SO THEN
0102 02B8                   INCF    SUMHI,1         ; INCREMENT THE HI BYTE
0103 0C00                   MOVLW   0               ; LOAD 0 INTO THE W REGISTER
0104 06EE                   BTFSC   ACCUM,7         ; IF THE INTEGRAL ACCUMULATOR WAS NEGATIVE
0105 0240                   COMF    W,W             ; COMPLEMENT THE 0 TO GET SIGN FOR HIGH BYTE
0106 01F8                   ADDWF   SUMHI,1         ; ADD INTO THE HIGH BYTE OF THE SUM

                    U_DEXIT                        ; EXIT POINT FOR THE UP/DOWN CONTROL OF ACCUM


                    ;****CALCULATING THE DIFFERENTIAL TERM*************************
                    ; The differential term examines the error and determines how much
                    ; it has changed since the last cycle.  It does this by subtracting the
                    ; old error from the new error.  Since the cycle time is relatively fixed
                    ; we can use it as the “dt” of the desired “de/dt”.  This derivative of the
                    ; error is then multiplied by the differential gain term KD and becomes the
                    ; differential term contribution for the final summation.

                    ; First, create the “de” term by doing a signed subtaction of new error
                    ; minus the old error. (new_error - old_error)

                    C_DIFF
0107 020C                   MOVF    ERROR,W         ; LOAD THE NEW ERROR INTO REGISTER
0108 0719                   BTFSS   FLAGS,ER_SGN
0109 0B0D                   GOTO    LO_BYTE
010A 026C                   COMF    ERROR,1         ; CORRECT THE VALUE TO BE 16 BIT
010B 028C                   INCF    ERROR,W
010C 026C                   COMF    ERROR,1         ; RESTORE IT FOR FUTURE USE TO 8 BIT MAGNI-
TUDE
                    LO_BYTE
010D 0034                   MOVWF   ACCbLO          ; FOR SUBTRACTION
010E 0C00                   MOVLW   00
010F 0619                   BTFSC   FLAGS,ER_SGN    ; SIGN EXTEND THE UPPER BYTE
0110 0CFF                   MOVLW   0FF
0111 0036                   MOVWF   ACCbHI
0112 020F                   MOVF    ERR_O,W         ; LOAD THE OLD ERROR INTO OTHER REGISTER
0113 0799                   BTFSS   FLAGS,OER_SGN
0114 0B17                   GOTO    LO_BYTEO
0115 026F                   COMF    ERR_O,1         ; CORRECT THE VALUE TO BE 16 BIT
0116 028F                   INCF    ERR_O,W
                    LO_BYTEO
0117 0033                   MOVWF   ACCaLO          ; FOR SUBTRACTION
0118 0C00                   MOVLW   00
0119 0699                   BTFSC   FLAGS,OER_SGN   ; SIGN EXTEND THE UPPER BYTE
011A 0CFF                   MOVLW   0FF
011B 0035                   MOVWF   ACCaHI
011C 090F                   CALL    D_sub           ; PERFORM THE SUBTRACTION

                    STRIP_SGN
011D 06F6                   BTFSC   ACCbHI,7        ; TEST THE SIGN OF THE RESULT
011E 0B20                   GOTO    NEG_ABS
011F 0B25                   GOTO    POS_ABS
                    NEG_ABS
0120 0559                   BSF     FLAGS,DE_SGN    ; ITS NEGATIVE SO SET THE FLAG AND
0121 0274                   COMF    ACCbLO,1        ; COMPLEMENT THE VALUE
0122 0294                   INCF    ACCbLO,W
0123 002F                   MOVWF   ERR_O
0124 0B28                   GOTO    MULT_KD
                    POS_ABS
0125 0459                   BCF     FLAGS,DE_SGN    ; ITS POSATIVE SO SET RESET THE FLAG
```

```
0126 0214                   MOVF    ACCbLO,W        ; AND SAVE THE VALUE
0127 002F                   MOVWF   ERR_O

                    ; Then multiply by Kd

                    MULT_KD
0128 020F                   MOVF    ERR_O,W
0129 0033                   MOVWF   mulcnd          ; MOVE THE DE/DT TERM INTO THE MULCND REG.
012A 0C20                   MOVLW   KD              ; MOVE THE DIFFERENTIAL GAIN TERM INTO
012B 0034                   MOVWF   mulplr          ; MULPLR TO MULTIPLY THE DE/DT
012C 0901                   CALL    mpy_S           ; DO THE MULTIPLICATION
012D 091D                   CALL    DIV_LMT         ; SCALE AND LIMIT TO 100

                    RE_SGN
012E 0759                   BTFSS   FLAGS,DE_SGN    ; IF THE DE SIGN IS NEGATIVE THEN
012F 0B32                   GOTO    SAVE_DIFF       ; PUT THE SIGN INTO THE LOW BYTE
0130 0276                   COMF    L_byte,1
0131 02B6                   INCF    L_byte,1
                    SAVE_DIFF
0132 0216                   MOVF    L_byte,W
0133 0643                   BTFSC   SWR,Z
0134 0B45                   GOTO    ROLL_ER
0135 002F                   MOVWF   ERR_O

                    ; ADD THE DIFF TERM INTO THE SUMM ***************
                    ADDDIF
0136 0C00                   MOVLW   00
0137 0659                   BTFSC   FLAGS,DE_SGN    ; PUT THE KD*(DE/DT) TERM INTO THE
0138 0CFF                   MOVLW   0FF             ; REGISTERS TO ADD. AND
0139 0036                   MOVWF   ACCbHI          ; SIGN EXTEND THE UPPER BYTE
013A 020F                   MOVF    ERR_O,W
013B 0034                   MOVWF   ACCbLO
013C 020D                   MOVF    SUMLO,W         ; LOAD THE CURRENT SUM INTO THE
013D 0033                   MOVWF   ACCaLO          ; REGISTERS TO ADD
013E 0218                   MOVF    SUMHI,W
013F 0035                   MOVWF   ACCaHI
0140 0910                   CALL    D_add           ; ADD IN THE DIFFERENTIAL TERM
0141 0214                   MOVF    ACCbLO,W        ; SAVE THE RESULTS BACK
0142 002D                   MOVWF   SUMLO           ; INTO SUMLO AND HI
0143 0216                   MOVF    ACCbHI,W
0144 0038                   MOVWF   SUMHI

                    ROLL_ER
0145 020C                   MOVF    ERROR,W         ; TAKE THE CURRENT ERROR
0146 002F                   MOVWF   ERR_O           ; AND PUT IT IN THE ERROR HISTORY
0147 0499                   BCF     FLAGS,OER_SGN   ; SAVE THE CURRENT ERROR SIGN
0148 0619                   BTFSC   FLAGS,ER_SGN    ; IN THE OLD ERROR SIGN FOR
0149 0599                   BSF     FLAGS,OER_SGN   ; NEXT TIME THROUGH


                    ;****SET UP THE DIRECTION FOR THE BRIDGE*********************
                    ;
                    ; After the sum of all the components has been made, the sign of the
                    ; sum will determine which way the bridge should be powered.
                    ; If the sum is negative the bridge needs to be set to drive ccw; if the
                    ; sum is posative then the bridge needs to be set to drive cw.  This
                    ; is purely a convention and depends upon the polarity the motor and feedback
                    ; element are hooked up in.

                    SET_DIR
014A 0479                   BCF     FLAGS,DIR       ; SET FOR DEFAULT CLOCKWISE
014B 06F8                   BTFSC   SUMHI,7         ; LOOK AT THE SIGN BIT, IF IT IS SET
014C 0579                   BSF     FLAGS,DIR       ; THEN SET FOR CCW BRIDGE DRIVE


                    ;**** SCALE THE NUMBER TO BETWEEN 0 AND 100% **********************
                    ; After the direction is set the request for duty cycle is limited to between
                    ; 0 and 100 percent inclusive.  This value is passed to the dutycycle setting
                    ; routine by loading it in the variable “PCNT”.
```

```
                        L_SUMM
014D 07F8                       BTFSS   SUMHI,7         ; CHECK TO SEE IF IT IS NEGATIVE
014E 0B52                       GOTO    POS_LM
014F 0278                       COMF    SUMHI,1
0150 026D                       COMF    SUMLO,1
0151 02AD                       INCF    SUMLO,1

                        POS_LM
0152 0C01                       MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0153 0098                       SUBWF   SUMHI,0         ; IF THERE IS ANYTHING THERE, IF NOT,
0154 0703                       BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0155 0B59                       GOTO    LB_L            ; OTHERWISE GIVE THE LOW BYTE A FULL
0156 0C64                       MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
0157 002D                       MOVWF   SUMLO           ; TO 100
0158 0B5F                       GOTO    LP_EXIT         ; GOTO LIMIT PERCENT EXIT
                        LB_L
0159 0C64                       MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
015A 008D                       SUBWF   SUMLO,0         ; 100 DECIMAL
015B 0703                       BTFSS   SWR,CARRY
015C 0B5F                       GOTO    LP_EXIT
015D 0C64                       MOVLW   64H
015E 002D                       MOVWF   SUMLO

                        LP_EXIT
015F 020D                       MOVF    SUMLO,W         ; STORE THE LIMITED VALUE IN
0160 0029                       MOVWF   PCNT            ; THE PERCENT DUTYCYCLE REQUEST


                        ;*********************************************************
                        ; PWM GENERATING ROUTINE
                        ;
                        ; The important thing here is not to have to do too many decisions or
                        ; calculations while you are generating the 100 or so pulses.  These will
                        ; take time and limit the minimum or maximum duty cycle.

                        WHICH_DIR
0161 0679                       BTFSC   FLAGS,DIR       ; CHECK THE DIRECTION FLAG
0162 0B76                       GOTO    GOCCW           ; DO CCW PULSES FOR 1
0163 0B64                       GOTO    GOCW            ; DO CW PULSES FOR 0


                        GOCW
0164 0426                       BCF     PORTB,PWMCCW    ; SET THE BRIDGE FOR CW MOVE
0165 0C64                       MOVLW   64H             ;
0166 0032                       MOVWF   CYCLES          ; SET UP CYCLES COUNTER FOR 100 PULSES
0167 0943                       CALL    CALCTIMES       ; CALCULATE THE HI AND LO TIMES

                        RLDCW
0168 0207                       MOVF    HI,0            ; RE LOAD THE HI TIMER
0169 002A                       MOVWF   HI_T            ; WITH THE CALCULATED TIME
016A 0208                       MOVF    LO,0            ; RE LOAD THE LO TIMER
016B 002B                       MOVWF   LO_T            ; WITH THE CALCULATED TIME
016C 0004                       CLRWDT                  ; TAG THE WATCHDOG TIMER

                        CWHI
016D 0506                       BSF     PORTB,PWMCW     ; SET THE CLOCKWISE PWMBIT HIGH
016E 02EA                       DECFSZ  HI_T,1          ; DECREMENT THE HI USEC. COUNTER
016F 0B6D                       GOTO    CWHI            ; DO ANOTHER LOOP
                        CWLO
0170 0406                       BCF     PORTB,PWMCW     ; SET THE CLOCKWISE PWM BIT LOW
0171 02EB                       DECFSZ  LO_T,1          ; DECREMENT THE LO USEC. COUNTER
0172 0B70                       GOTO    CWLO            ; DO ANOTHER LOOP
0173 02F2                       DECFSZ  CYCLES,1        ; DECREMENT THE NUMBER OF CYCLES LEFT
0174 0B68                       GOTO    RLDCW           ; DO ANOTHER PULSE
0175 0A50                       GOTO    BEGIN           ; DO ANOTHER MAIN SYSTEM CYCLE
```

```
                        GOCCW
0176 0406                       BCF     PORTB,PWMCW     ; SET THE BRIDGE FOR CCW MOVE
0177 0C64                       MOVLW   64H             ;
0178 0032                       MOVWF   CYCLES          ; SET UP CYCLE COUNTER FOR 100 PULSES
0179 0943                       CALL    CALCTIMES       ; CALCULATE THE HI AND LO TIMES
                        RLDCCW
017A 0207                       MOVF    HI,0            ; RE LOAD THE HI TIMER
017B 002A                       MOVWF   HI_T            ; WITH THE CALCULATED TIME
017C 0208                       MOVF    LO,0            ; RE LOAD THE LO TIMER
017D 002B                       MOVWF   LO_T            ; WITH THE CALCULATED TIME
017E 0004                       CLRWDT                  ; TAG THE WATCHDOG

                        CCWHI
017F 0526                       BSF     PORTB,PWMCCW    ; SET THE COUNTERCLOCKWISE PWM BIT HIGH
0180 02EA                       DECFSZ  HI_T,1          ; DECREMENT THE HI USEC. COUNTER
0181 0B7F                       GOTO    CCWHI           ; DO ANOTHER LOOP
                        CCWLO
0182 0426                       BCF     PORTB,PWMCCW    ; SET THE COUNTERCLOCKWISE PWM BIT LOW
0183 02EB                       DECFSZ  LO_T,1          ; DECREMENT THE LO USEC. COUNTER
0184 0B82                       GOTO    CCWLO           ; DO ANOTHER LOOP
0185 02F2                       DECFSZ  CYCLES,1        ; DECREMENT THE NUMBER OF CYCLES LEFT
0186 0B7A                       GOTO    RLDCCW          ; DO ANOTHER PULSE
0187 0A50                       GOTO    BEGIN           ; DO ANOTHER MAIN SYSTEM CYCLE



                          ;************* START VECTOR ********************

                        CLRREG                          ;INITIALIZE REGISTERS

0188 0C0B                       MOVLW   0BH             ; SET PORT A FOR 3 INPUTS AND
0189 0005                       TRIS    PORTA           ; AN OUTPUT
018A 0C1C                       MOVLW   1CH             ; SET PORT B FOR INPUTS AND OUTPUTS
018B 0006                       TRIS    PORTB           ; THIS SETTING FOR SENDING TO A/D
018C 0040                       CLRW                    ; CLEAR THE W REGISTER
018D 0002                       OPTION                  ; STORE THE W REG IN THE OPTION REG
018E 0C08                       MOVLW   08H             ; STARTING REGISTER TO ZERO
018F 0024                       MOVWF   FSR             ;
                        GCLR
0190 0060                       CLRF    00              ;
0191 03E4                       INCFSZ  FSR             ; SKIP AFTER ALL REGISTERS
0192 0B90                       GOTO    GCLR            ; HAVE BEEN INITIALIZED
0193 0A50                       GOTO    BEGIN           ; START AT THE BEGINING OF THE PROGRAM

                                ORG     01FF            ;
01FF 0B88                       GOTO    CLRREG          ; START VECTOR


                              END


Errors   :    0
Warnings :    0
```

AN590

A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs

INTRODUCTION

The purpose of this application note is to design a clock while multiplexing the features as much as possible, allowing the circuit to use the 18-pin PIC16C54. Other devices in the Microchip Technology Inc. line expand on this part, making it a good starting point for learning the basics. This design is useful because it utilizes every pin for output and switches some of them to inputs briefly to read the keys. For a more extensive clock design, consult application note AN529.

THE DESIGN

This design is a simple time of day clock incorporating four seven-segment LED displays and three input switches. There is also an additional reset switch that would not normally be incorporated into the final design. The schematic is illustrated in Figure 1.

CONNECTIONS

The individual segments of each display are connected together, A-A-A-A, B-B-B-B, etc. The displays are numbered from the right, or least significant digit. The second display from the right is flipped upside down to align its decimal with the third display, creating the center clock colon. Therefore the segments are not tied together evenly straight across on the board, but must compensate for the change in one display's orientation. The common cathode for each display is turned on with transistors connected to the four I/O lines of Port A. The connections are RA0-CC4/Digit4, RA1-CC3/Digit3, RA2-CC2/Digit2, RA3-CC1/Digit1. A low output turns on the PNP transistor for the selected display. The Port B pins activate the LED segments. For this design only the center colon decimal points were connected. The connections are RB0-dp, RB1-A, RB2-B, RB3-C...RB7-G.

The switches are also connected to Port B I/O pins. Port B pins RB1, RB2, and RB3 are pulled low with 10K ohm resistors. This value is high enough to not draw current away from the LEDs when they are being driven on. Inputs are detected by pulling the pins high with a switch to V_{DD} through 820 ohm resistors. This value is low enough to pull the pin high quickly when the outputs have been turned off, and to create a 90% of V_{DD} high input.

OPERATION

Switches

When no buttons are pressed, the circuit will display the current time, starting at 12:00 on reset. Pressing SW1 will cause seconds to be displayed. The time is set by pressing SW2 to advance minutes, and SW3 to advance hours. Since each of the segments are tied together across all displays, only one display should be turned on at a time, or all displays turned on would display duplicate data. The displays are turned on right to left, with each display's value being output its turn. This is done fast enough so that there is no perceived flicker. The switches are read between display cycles.

Timing

The PIC16CXX prescaler is assigned to the RTCC as a 1:16 divide. The RTCC pin is tied low since it is not used. The OPTION Register is loaded with 03h to initialize this prescaler set up. The software is written with timing based on a 4.000mhz crystal. The instruction clock is 1.000 MHz after the internal divide by four. The 8-bit RTCC register rolls over every 256 cycles, for a final frequency of 244.1406 Hz. (exactly a 4.096 ms period) A variable named sec_nth is used to count 244 roll-overs of the RTCC for one second. The benefit of keeping time with a nth variable is that it can be written to as needed to adjust time in "nths" of a second, allowing almost any odd crystal frequency to be used. Simply determine the best prescale and "nth" divider, and compute the "nth" adjustment needed for each minute, hour, twelve hour roll-over. Time can be kept accurately to two "nths" a day (an "nth" is 1/244 of a second in this case). In this circuit, 9 "nths" are subtracted each minute, 34 "nths" are added each hour, and 6 "nths" subtracted every twelve hour roll-over. This leaves a computed error of 1.5 seconds/year except for crystal frequency drift. Another possible solution is to initialize the RTCC to some value that causes a roll-over at a predetermined time interval. Writing to the RTCC causes two clock cycles to be missed while clock edges realign, which would have to be accounted for. This is described in the *Microchip Data Book*.

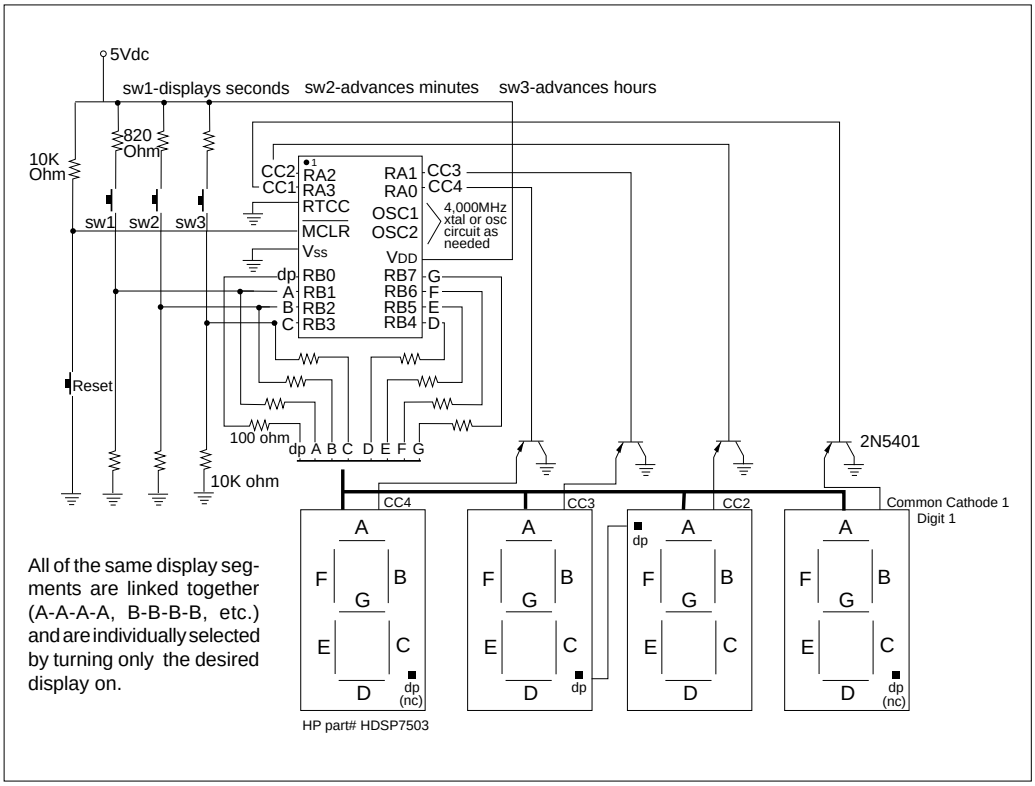
Displays

The program contains portions of code that act as a display driver. A variable exists for each of the four displays. A hex value from 0 to 9 can be written to these variables and they will be converted to display code and output to the displays. Only one display is actually on at a time, and its code is output into it in its turn. Another section of code takes the seconds, minutes, or hours value and separates it into the two digits needed for each display. In other words, 48 seconds would be separated into a "4" and an "8" and written to the appropriate display variable. The displays used were common cathode and turned on with transistors to avoid trying to sink too much current into the PIC16CXX. A display is enabled with a zero at the appropriate pin. 100 ohm resistors were used in series with the segments to obtain the desired brightness. Different values may be required if different displays are used. Since the displays are each on less than one fourth of the time, the resistor value must be low enough to compensate for the needed forward current.

CONCLUSION

The instruction execution speed of the PIC16C54 (and the rest of the PIC16/17 series) allows many functions to be implemented on a few pins by multiplexing them in software. User inputs, Real Time Clock Counter, and multiple LED displays are all accommodated with little or no sacrifice in functionality.

FIGURE 1: TIME OF DAY CLOCK USING PIC16C54



Author: Dan Matthews
Corporate Applications Manager

```
-LOC   LINE SOURCE TEXT

       0001  list P = 16C54
       0002 ;
       0003 ;*****************************************************************
       0004 ;                       Clock
       0005 ;*****************************************************************
       0006 ;
       0007 ;                PROGRAM DESCRIPTION
       0008 ;
       0009 ; This program runs on a PIC16C54.
       0010 ;
       0011 ;                       Hardware Description
       0012 ;
       0013 ;  DISPLAYS
       0014 ; Four 7 segment displays are multiplexed.  The segments are tied together,
       0015 ; with the common cathode pins broken out separately.  The display appears
       0016 ; as a clock with a center semicolon ( 88:88 ).  The segments are assigned
       0017 ; to Port B, with the semicolon being RB0, and segments A through F
       0018 ; assigned as RB1 to RB7 respectively.
       0019 ;  The four common cathodes are activated by the four Port A pins through
       0020 ; transistors. RA0 for Digit4, RA1/Digit3, RA2/Digit2... through Digit4,
       0021 ; with Digit1 being in the rightmost position. The center semicolon is
       0022 ; made from the decimals of LED 2 and 3.
       0023 ;  Digit2 is turned upside down to put its decimal into position,
       0024 ; but it is wired with a corrected A-F assignment to compensate.  Both
       0025 ; decimals are tied together at RB0, but the display cathodes are still
       0026 ; separate.  Activating the decimal for digit2 AND 3 will turn on the
       0027 ; center colon.
       0028 ;
       0029 ;  SWITCHES
       0030 ; Because all twelve I/O pins are already used for the muxed displays,
       0031 ; eight for segments and four for digit selection, the three switches must
       0032 ; be read alternatingly through software.  The switches lie
       0033 ; across Port B pins, which are changed to inputs momentarily during read
       0034 ; and changed back to outputs during display.  Enough series resistance
       0035 ; must be used to prevent turning on or shorting segments during display
       0036 ; cycles if a switch is pressed.
       0037 ;
       0038 ; SW1-displays seconds, SW2-advances minutes,
       0039 ; SW3-advances hours, (none)-displays time
       0040 ;
       0041 ;*************************   Header ************************
       0042 ;
       0043 ;
01FF   0044 PIC54     equ    H'01FF' ; start address if used in a PIC16C54
03FF   0045 PIC56     equ    H'03FF' ;   "     "     "    "   " " PIC16C56
       0046                         ;
0000   0047 POINTER   equ    H'00'   ; address location f0 is an indirect address pointer
0001   0048 RTCC      equ    H'01'   ; address of RTCC clock value
0002   0049 PC        equ    H'02'   ; program counter
0003   0050 STATUS    equ    H'03'   ; F3 Reg is STATUS Reg.
0004   0051 FSR       equ    H'04'   ; F4 is File Select Register, address POINTER will direct to.
       0052                         ;
0005   0053 PORT_A    equ    H'05'   ; 7 segment Display Common Cathodes
0006   0054 PORT_B    equ    H'06'   ; Muxed Display Segments (Switches when inputs)
       0055                         ;
       0056                         ; STATUS REG. Bits
0000   0057 CARRY     equ    0       ; Carry Bit is Bit.0 of F3
0000   0058 C         equ    0
0001   0059 DCARRY    equ    1
0001   0060 DC        equ    1
0002   0061 Z_bit     equ    2       ; Bit 2 of F3 is Zero Bit
0002   0062 Z         equ    2
0003   0063 P_DOWN    equ    3
0003   0064 PD        equ    3
0004   0065 T_OUT     equ    4
0004   0066 TO        equ    4
0005   0067 PA0       equ    5       ;16C5X Status bits
```

```
0006    0068 PA1       equ     6            ;16C5X Status bits
0007    0069 PA2       equ     7            ;16C5X Status bits
        0070                                ;
007E    0071 ZERO      equ     H'7E'
000C    0072 ONE       equ     H'0C'
00B6    0073 TWO       equ     H'B6'
009E    0074 THREE     equ     H'9E'
00CC    0075 FOUR      equ     H'CC'
00DA    0076 FIVE      equ     H'DA'
00FA    0077 SIX       equ     H'FA'        ; coding of segments for display (PORT_B)
000E    0078 SEVEN     equ     H'0E'
00FE    0079 EIGHT     equ     H'FE'
00CE    0080 NINE      equ     H'CE'
0000    0081 BLANK     equ     H'00'
        0082                                ;
        0083                                ; timer variables start at a number that allows
        0084                                ; rollover in sync with time rollover, i.e. seconds
        0085                                ; starts at decimal 196 so that sixty 1-second
                                            ;   increments causes 0.
000C    0086 MAXNTHS   equ     D'12'        ;  initialization constants for timer count up
00C4    0087 MAXSECS   equ     D'196'       ;     see variable explanations for more info
00C4    0088 MAXMINS   equ     D'196'       ;
00F4    0089 MAXHRS    equ     D'244'
00F3    0090 MINHRS    equ     D'243'
0009    0091 ADJMIN    equ     D'9'         ; number of nths to be subtracted each minute for
                                            ;   accuracy
0022    0092 ADJHR     equ     D'34'        ; nths added each hour for accurate time
0006    0093 ADJDAY    equ     D'6'         ; nths subtracted each 1/2 day rollover
        0094 ;
00FE    0095 DISP4     equ     B'11111110'
00FD    0096 DISP3     equ     B'11111101'  ; Mapping of Active Display Selection (PORT_A)
00FB    0097 DISP2     equ     B'11111011'  ;  displays are active low
00F7    0098 DISP1     equ     B'11110111'
00FF    0099 DISPOFF   equ     H'FF'        ; turns all displays off when written to PORT_A
000E    0100 SWITCH    equ     B'00001110'  ; Used in tris B to set RB1-3 for switch inputs
        0101                                ;
        0102                                ;  Flag bit assignments
0000    0103 SEC       equ     H'0'         ; update time display values for sec, min, or hours
0001    0104 MIN       equ     H'1'
0002    0105 HRS       equ     H'2'
0003    0106 CHG       equ     H'3'         ; a change has occurred on a switch or a display
                                            ;   value
0004    0107 SW1       equ     H'4'         ; Flag bit assignments - switches that are on = 1
0005    0108 SW2       equ     H'5'         ;  SW1 is Seconds-minutes, SW2-hours, SW3-mode
0006    0109 SW3       equ     H'6'
0007    0110 SW_ON     equ     H'7'         ; indicates a switch has been pressed
        0111                                ;
        0112                                ; RAM VARIABLES
        0113 ;         equ     H'08'        ; not used
0009    0114 flags     equ     H'09'        ; bits:0-SEC,1-MIN,2-HRS,3-CHG,4-SW1,5-SW2,6-SW3,7-SW_ON
000B    0115 display   equ     H'0B'        ; SW_ON variable location - which display to update
000C    0116 digit1    equ     H'0C'        ; Rightmost display value
000D    0117 digit2    equ     H'0D'        ; Second display from right
000E    0118 digit3    equ     H'0E'        ; Third    "       "    "
000F    0119 digit4    equ     H'0F'        ; Fourth (and Leftmost)
0010    0120 sec_nth   equ     H'10'        ; seconds, fractional place
0011    0121 seconds   equ     H'11'        ; seconds
0012    0122 minutes   equ     H'12'        ; minutes
0013    0123 hours     equ     H'13'        ; hours
0014    0124 var       equ     H'14'        ; variable for misc math computations
0015    025  count     equ     H'15'        ; loop counter variable
        0126                                ;
        0127                                ;*****************************************
        0128                                ;
        0129                                ;  Initialize Ports all outputs, blank display
        030                                 ;
```

```
     OBJECT
-LOC  CODE  LINE SOURCE TEXT
0000  0C03  0131 START   movlw   H'03'        ; set option register, transition on clock,
0001  0002  0132         option                ; Prescale RTCC, 1:16
                        0133                       ;
0002  0C00  0134         movlw   0
0003  0005  0135         tris    PORT_A        ; Set all port pins as outputs
0004  0006  0136         tris    PORT_B
0005  0C00  0137         movlw   BLANK
0006  0026  0138         movwf   PORT_B        ; Blank the display
0007  04C3  0139         bcf     STATUS,PA1    ; page zero in case this is a higher PIC version
0008  04A3  0140         bcf     STATUS,PA0
            0141                               ;
            0142                               ;  initialize variables
0009  0C01  0143         movlw   H'01'
000A  0021  0144         movwf   RTCC          ; set RTCC above zero so initial wait period occurs
000B  0CF7  0145         movlw   DISP1
000C  002B  0146         movwf   display       ; initializes display selected to first display.
000D  0C00  0147         movlw   BLANK         ; put all displays to blank, no visible segments
000E  002C  0148         movwf   digit1
000F  002D  0149         movwf   digit2
0010  002E  0150         movwf   digit3
0011  002F  0151         movwf   digit4
0012  0C0C  0152         movlw   MAXNTHS       ; set timer variables to initial values
0013  0030  0153         movwf   sec_nth
0014  0CC4  0154         movlw   MAXSECS
0015  0031  0155         movwf   seconds
0016  0CC4  0156         movlw   MAXMINS
0017  0032  0157         movwf   minutes
0018  0CFF  0158         movlw   H'FF'         ; hours start at 12 which is max at FF
0019  0033  0159         movwf   hours
001A  0C00  0160         movlw   H'00'
001B  0029  0161         movwf   flags         ; clear the flags variable
001C  0004  0162         clrwdt                ; clear WatchDog Timer, must be within every 18ms
            0163                               ;
            0164                               ;
            0165 MAIN
            0166                               ;
            0167                               ;  wait for RTCC to roll-over
            0168 RTCC_FILL
001D  0201  0169         movf    RTCC,0
001E  0743  0170         btfss   STATUS,Z      ; note, RTCC is left free running
001F  0A1D  0171         goto    RTCC_FILL
            0172                               ;
0020  03F0  0173         incfsz  sec_nth,1     ;add 1 to nths, n X nths = 1 sec, n is based on prescaler
0021  0A54  0174         goto    TIME_DONE
0022  0004  0175         clrwdt
0023  0C0C  0176         movlw   MAXNTHS
0024  0030  0177         movwf   sec_nth       ; restore sec_nths variable for next round
            0178                               ;
            0179 CHECK_SW
0025  07E9  0180         btfss   flags,SW_ON   ; if no switches pressed, bypass this
0026  0A3C  0181         goto    SET_TIME
0027  0689  0182         btfsc  flags,SW1
0028  0A3C  0183         goto   SET_TIME       ; if seconds display is pressed, do not change time
0029  0CC4  0184         movlw  MAXSECS
002A  0031  0185         movwf  seconds        ; reset seconds to zero when setting clock
002B  0C7F  0186         movlw  H'7F'
002C  0030  0187         movwf  sec_nth        ; advance second timer 1/2 second to speed setting
002D  07A9  0188         btfss  flags,SW2
002E  0A35  0189         goto   HOURSET        ; if minutes do not need changing, check hours
002F  0CAF  0190         movlw  H'AF'
0030  0030  0191         movwf  sec_nth        ; advances timer faster when setting minutes
0031  03F2  0192         incfsz minutes,1      ; advances minutes 1
0032  0A35  0193         goto   HOURSET
0033  0CC4  0194         movlw  MAXMINS
0034   0032 0195         movwf  minutes        ; if minutes roll over to zero, reinitialize
                                                ;    minutes
            0196                               ;
```

```
0035  06A9  0197 HOURSET btfsc   flags,SW2
0036  0A60  0198         goto    CHECK_TIME     ; if not changing hours (changing minutes)
                                                ; skip this
0037  03F3  0199         incfsz  hours,1        ; advance hours 1
0038  0A60  0200         goto    CHECK_TIME
0039  0CF4  0201         movlw   MAXHRS         ; if hours rolls over to zero, reinitialize
003A  0033  0202         movwf   hours
003B  0A60  0203         goto    CHECK_TIME     ;  skip time keeping, go to display changes
            0204                                ;
            0205 SET_TIME
003C  0509  0206         bsf     flags,SEC      ; indicates seconds, if displayed, should be
                                                ;   updated
003D  0569  0207         bsf     flags,CHG      ; inicates a flag change was made.
003E  03F1  0208         incfsz  seconds,1      ;  add 1 to seconds
003F  0A54  0209         goto    TIME_DONE
0040  0CC4  0210         movlw   MAXSECS
0041  0031  0211         movwf   seconds        ; restore seconds variable for next round
            0212                                ;
0042  0529  0213         bsf     flags,MIN      ; minutes, if displayed, should be updated
0043  0569  0214         bsf     flags,CHG      ; indicates a flag change was made
0044  0C09  0215         movlw   ADJMIN
0045  00B0  0216         subwf   sec_nth,1      ; accuracy adjustment, do not go below 0
0046  03F2  0217         incfsz  minutes,1      ; add 1 to minutes
0047  0A54  0218         goto    TIME_DONE
0048  0CC4  0219         movlw   MAXMINS
0049  0032  0220         movwf   minutes        ; restore minutes variable for next hour
                                                ;   countdown
            0221                                ;
004A  0549  0222         bsf     flags,HRS      ; hours, if displayed, should be updated
004B  0569  0223         bsf     flags,CHG      ; indicates a flag change was made
004C  0C22  0224         movlw   ADJHR
004D  01F0  0225         addwf   sec_nth,1      ; add needed adjustment to nths for each hour
004E  03F3  0226         incfsz  hours,1        ; add 1 to hours
004F  0A54  0227         goto    TIME_DONE
0050  0CF4  0228         movlw   MAXHRS
0051  0033  0229         movwf   hours          ; restore hours variable for next round
0052  0C06  0230         movlw   ADJDAY
0053  00B0  0231         subwf   sec_nth,1      ; subtraction adjustment for each 1/2 day rollover
            0232                                ;
            0233 TIME_DONE
0054  0769  0234         btfss   flags,CHG      ; if no switches or potentially dislayed
                                                ;   numbers
0055  0A91  0235         goto    CYCLE          ;  were changed, then leave the display same
            0236                                ;
            0237                                ;
            0238 CHECK_SECONDS
            0239                                ;  if seconds button was pushed display
                                                ;   seconds
0056  0789  0240         btfss   flags,SW1
0057  0A60  0241         goto    CHECK_TIME     ; if seconds button not pressed, skip this
0058  0C00  0242         movlw   H'00'          ; zero time display variables except seconds
                                                ;   (digit1)
0059  002D  0243         movwf   digit2         ; digit1 used to temporarily hold hex seconds
                                                ;   or minutes
005A  002E  0244         movwf   digit3
005B  002F  0245         movwf   digit4
005C  0CC4  0246         movlw   MAXSECS
005D  0091  0247         subwf   seconds,0      ; subtract initialized preset to get actual
                                                ;   seconds
005E  002C  0248         movwf   digit1         ; 1st digit variable temporarily holds hex
                                                ;   value seconds
005F  0A69  0249         goto    SPLIT_HEX      ; done updating display variables in hex
            0250                                ;
            0251 CHECK_TIME
0060  0C00  0252         movlw   H'00'
0061  002F  0253         movwf   digit4         ; zero out tens places in case there is no tens
                                                ;   increment
0062  002D  0254         movwf   digit2
```

```
0063  0CF3   0255          movlw   MINHRS
0064  0093   0256          subwf   hours,0          ; subtract initialized preset to get actual
                                                    ;   hours
0065  002E   0257          movwf   digit3           ; 3rd digit variable temporarily holds hex
                                                    ;   value for hours
0066  0CC4   0258          movlw   MAXMINS
0067  0092   0259          subwf   minutes,0        ; subtract initialized preset to get actual
                                                    ;   minutes
0068  002C   0260          movwf   digit1           ; 1st digit temporarily holds hex value for
                                                    ;   minutes
              0261                                  ;
              0262                                  ; note digit variables are used for temp
                                                    ;   variables and final display variables
              0263                                  ;
              0264 SPLIT_HEX                        ;  split into two hex display variables and
                                                    ;   write
              0265                                  ;
0069  0C02   0266          movlw   H'02'
006A  0035   0267          movwf   count            ; convert each number - seconds - or minutes
                                                    ;   and hours
              0268
              0269                                  ;1st time through, FSR = digit1, 2nd time FSR =
                                                    ;   digit3
006B  0C0C   0270          movlw   digit1
006C  0024   0271          movwf   FSR              ; address of digit1 into File Select Register
                                                    ;   prepares POINTER
006D  0A70   0272          goto    LOOP             ; this loop is used to modify the minutes/
                                                    ;   seconds place
              0273                                  ;
006E  0C0E   0274 LOOP2    movlw   digit3
006F  0024   0275          movwf   FSR              ; this loop is used to modify the hours place
              0276                                  ;
              0277 LOOP
0070  0C0A   0278          movlw   D'10'
0071  00A0   0279          subwf   POINTER,1        ; find out how many tens in number,
0072  0603   0280          btfsc   STATUS,C         ; was a borrow needed?
0073  0A76   0281          goto    INCREMENT_10S    ;  if not, add 1 to tens position
0074  01E0   0282          addwf   POINTER,1        ;  if so, do not increment tens place, add ten
                                                    ;   back on
0075  0A7A   0283          goto    NEXT_DIGIT
              0284                                  ;
              0285 INCREMENT_10S
0076  02A4   0286          incf    FSR,1            ; bump address pointed to from 1s positoion to
                                                    ;   10s
0077  02A0   0287          incf    POINTER,1        ; add 1 to 10s position as determined by
                                                    ;   previous subtract
0078  00E4   0288          decf    FSR,1            ; put POINTER value back to 1s place for next
                                                    ;   subtraction
0079  0A70   0289          goto    LOOP             ; go back and keep subtracting until finished
              0290                                  ;
              0291 NEXT_DIGIT
007A  02F5   0292          decfsz  count,1
007B  0A6E   0293          goto    LOOP2            ; after splitting minutes into two places, go
                                                    ;   split hours
              0294                                  ;
              0295 CONVERT_HEX_TO_DISPLAY           ; converts digit variables to decimal display
                                                    ;   code
007C  0C0C   0296          movlw   digit1
007D  0024   0297          movwf   FSR              ; put the address of the digit1 into the FSR to
                                                    ;   enable POINTER
007E  0C04   0298          movlw   H'04'
007F  0035   0299          movwf   count            ; prepare count variable to loop for all four
                                                    ;   displays
              0300 NEXT_HEX
0080  0200   0301          movf    POINTER,0        ; get the hex value of the current digit vari
                                                    ;   able
0081  09C3   0302          call    RETURN_CODE      ; call for the hex to segment display code
                                                    ;   conversion
0082  0020   0303          movwf   POINTER          ; put the returned display code into the digit
```

```
                                                    ;   variable
0083  02A4   0304         incf    FSR,1             ; increment the pointer to the next digit
                                                    ;   address
0084  02F5   0305         decfsz  count,1           ; allow only count(4) times through loop
0085  0A80   0306         goto    NEXT_HEX
             0307                                   ;
             0308 FIX_DISPLAY
0086  0C7E   0309         movlw   ZERO
0087  008F   0310         subwf   digit4,0          ; check to see if left digit is a zero, if so
                                                    ;   blank it out
0088  0743   0311         btfss   STATUS,Z
0089  0A8C   0312         goto    FIX_SEC
008A  0C00   0313         movlw   BLANK
008B  002F   0314         movwf   digit4
             0315
008C  0789   0316 FIX_SEC btfss   flags,SW1         ; if seconds are displayed, blank the third
                                                    ;   display too
008D  0A8F   0317         goto    CLEAR_FLAGS
008E  002E   0318         movwf   digit3
             0319                                   ;
             0320 CLEAR_FLAGS
008F  0CF0   0321         movlw   H'F0'
0090  0169   0322         andwf   flags,1           ; clear the lower 4 flag bits to show updated
                                                    ;   time status
             0323                                   ;
             0324 CYCLE
0091  0CFF   0325         movlw   DISPOFF
0092  0025   0326         movwf   PORT_A            ; Turn off LED Displays
0093  0C0E   0327         movlw   SWITCH
0094  0006   0328         tris    PORT_B            ; Set some port B pins as switch inputs
0095  0C0F   0329         movlw   H'0F'
0096  0169   0330         andwf   flags,1           ; reset switch flags to zero
0097  0000   0331         nop                       ; nop may not be needed, allows old outputs to
0098  0000   0332         nop                       ;   bleedoff through 10k R before reading port
                                                    ;   pins
0099  0000   0333         nop
009A  0206   0334         movf    PORT_B,0          ; read PORT_B for switch status
009B  0034   0335         movwf   var               ;  write switch status to temporary variable
                                                    ;   "var"
009C  0734   0336         btfss   var,1
009D  0AA1   0337         goto    SWITCH2           ; indicate which switches are pressed in the
                                                    ;   flags variable
009E  0569   0338         bsf     flags,CHG
009F  0589   0339         bsf     flags,SW1
00A0  05E9   0340         bsf     flags,SW_ON
00A1  0754   0341 SWITCH2 btfss   var,2
00A2  0AA6   0342         goto    SWITCH3
00A3  0569   0343         bsf     flags,CHG
00A4  05A9   0344         bsf     flags,SW2
00A5  05E9   0345         bsf     flags,SW_ON
00A6  0774   0346 SWITCH3 btfss   var,3
00A7  0AAB   0347         goto    SETPORT
00A8  0569   0348         bsf     flags,CHG
00A9  05C9   0349         bsf     flags,SW3
00AA  05E9   0350         bsf     flags,SW_ON
             0351                                   ;
00AB  0C00   0352 SETPORT movlw   H'00'             ; restore PORT_B as all outputs to displays
00AC  0006   0353         tris    PORT_B
00AD  0C00   0354            movlw   BLANK          ; blank display in preparation for next digit
                                                    ;   cycle
00AE  0026   0355            movwf   PORT_B
             0356                                   ;
             0357                                   ; determine which display needs updating and
                                                    ;   cycle it on
00AF  070B   0358            btfss   display,0      ; if 1st display, get 1st digit value into w
00B0  020F   0359            movf    digit4,0
00B1  072B   0360            btfss   display,1      ; if 2nd display, get 2nd digit
00B2  020E   0361            movf    digit3,0
00B3  074B   0362            btfss   display,2      ; if 3rd display, get 3rd digit
```

```
00B4  020D   0363            movf    digit2,0
00B5  076B   0364            btfss   display,3       ; if 4th display, get 4th digit
00B6  020C   0365            movf    digit1,0
00B7  0026   0366            movwf   PORT_B          ; put the number in w out to display
00B8  06F0   0367            btfsc   sec_nth,7
00B9  0506   0368            bsf     PORT_B,0        ; sets colon decimal on %50 duty using highest
                                                     ;   bit
00BA  020B   0369            movf    display,0       ; get display needing cycle on
00BB  0025   0370            movwf   PORT_A          ; enables proper display
00BC  002B   0371            movwf   display         ; enables display selected in last pass of
                                                     ;   CYCLE
00BD  036B   0372            rlf     display,1       ; rotate display “on” bit to next position
00BE  050B   0373            bsf     display,0       ; assures a 1 on lowest position since rotated
                                                     ;   in carry is zero
00BF  078B   0374            btfss   display,4       ; check if last display was already updated
00C0  040B   0375            bcf     display,0       ; if it was, set display back to 1st (bit 0
                                                     ;   cleared)
00C1  0004   0376            clrwdt                  ; this program pass completed normally, reset
                                                     ;   watch dog
             0377                                    ;
             0378                                    ;
             0379                                    ;
00C2  0A1D   0380            goto    MAIN
             0381                                    ;
             0382 RETURN_CODE
             0383                                    ;
00C3  01E2   0384            addwf   PC,1            ; the hex value in the display variable is
00C4  087E   0385            retlw   ZERO            ;   added to PC which causes a jump to return
                                                     ;   its display code
00C5  080C   0386            retlw   ONE
00C6  08B6   0387            retlw   TWO
00C7  089E   0388            retlw   THREE
00C8  08CC   0389            retlw   FOUR
00C9  08DA   0390            retlw   FIVE
00CA  08FA   0391            retlw   SIX
00CB  080E   0392            retlw   SEVEN
00CC  08FE   0393            retlw   EIGHT
00CD  08CE   0394            retlw   NINE
             0395                                    ;
             0396                                    ;
             0397            org     PIC54           ; reset location for this processor
Warning: Crossing page boundary — ensure page bits are set
01FF  0A00   0398            goto    START           ; begin program execution at START label
             0399                                    ;
             0400     END
             0401

MEMORY USAGE MAP (‘X’ = Used,  ‘-’ = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXX-- --------- --------- ---------

0180 : --------- --------- --------- ---------
01C0 : --------- --------- --------- --------X

All other memory blocks unused.

Errors   :    0
Warnings :    1
```

NOTES:

AN529

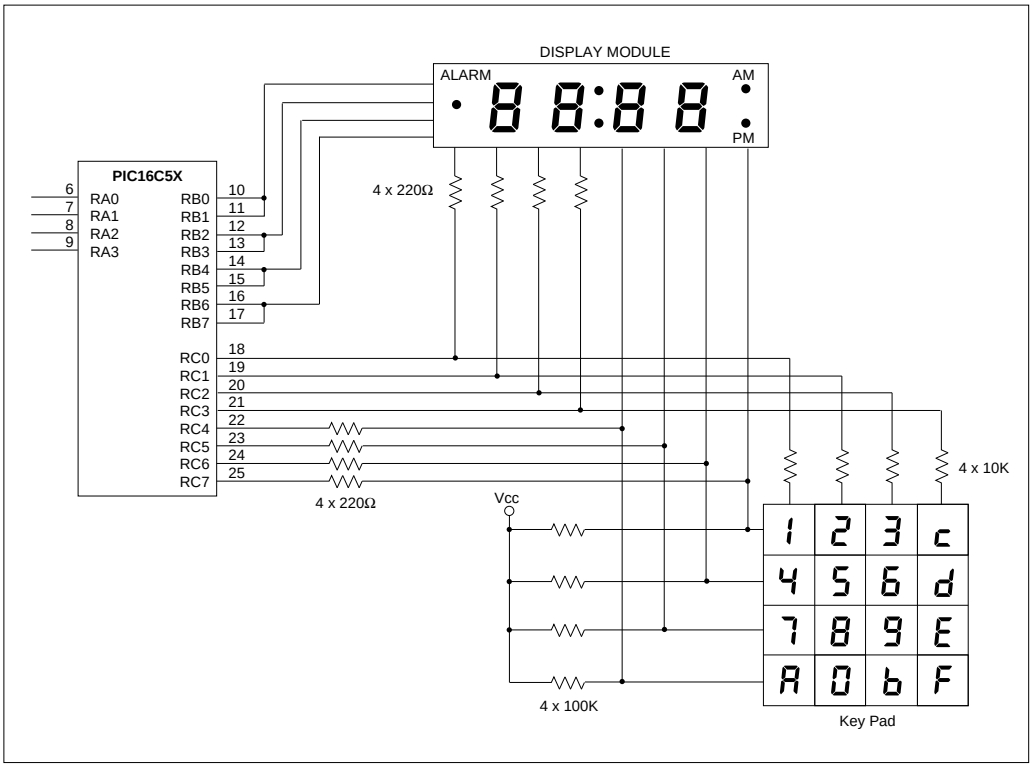
Multiplexing LED Drive and a 4x4 Keypad Sampling

INTRODUCTION

Many applications require driving LEDs along with an interface to a keypad. Implementing such designs usually involves using up significant amounts of the processors I/O lines. This application note describes a method which uses only 16 I/O pins of a PIC16C5X microcontroller to sample a 4x4 keypad matrix, and directly drive four 7 segment LEDs (see Figure 1). Direct drive of the LEDs is possible, because of the high sink and source capabilities of the PIC16C5X microcontroller, thus eliminating the use of external drive transistor, and resulting in reduced cost and complexity of the overall circuit.

Typically applications having LEDs and keypads also keep track of real time, in order to synchronize certain key events. An Industrial Clock/Timer example has been used in this application note as a demonstration of this technique. The software overhead to keep track of real time is minimal and the user can modify the code to significantly expand the functionality of this circuit.

FIGURE 1 - PIC16C5X INTERFACE TO A SEGMENT DISPLAY AND 4X4 KEYPAD



Multiplexing LED Drive and a 4x4 Keypad Sampling

PART A: 4X4 KEY MATRIX SAMPLING

Implementation

The 4x4 Key Matrix is connected to port C of the PIC16C5X (Figure 2a). The four columns are connected to RC0-RC3 and the four rows are connected to RC4-RC7. Each digit is refreshed every 20 ms. with a 5 ms pulse. The keypad is sampled every 20 ms with four 3 µs pulses (Figure 3).

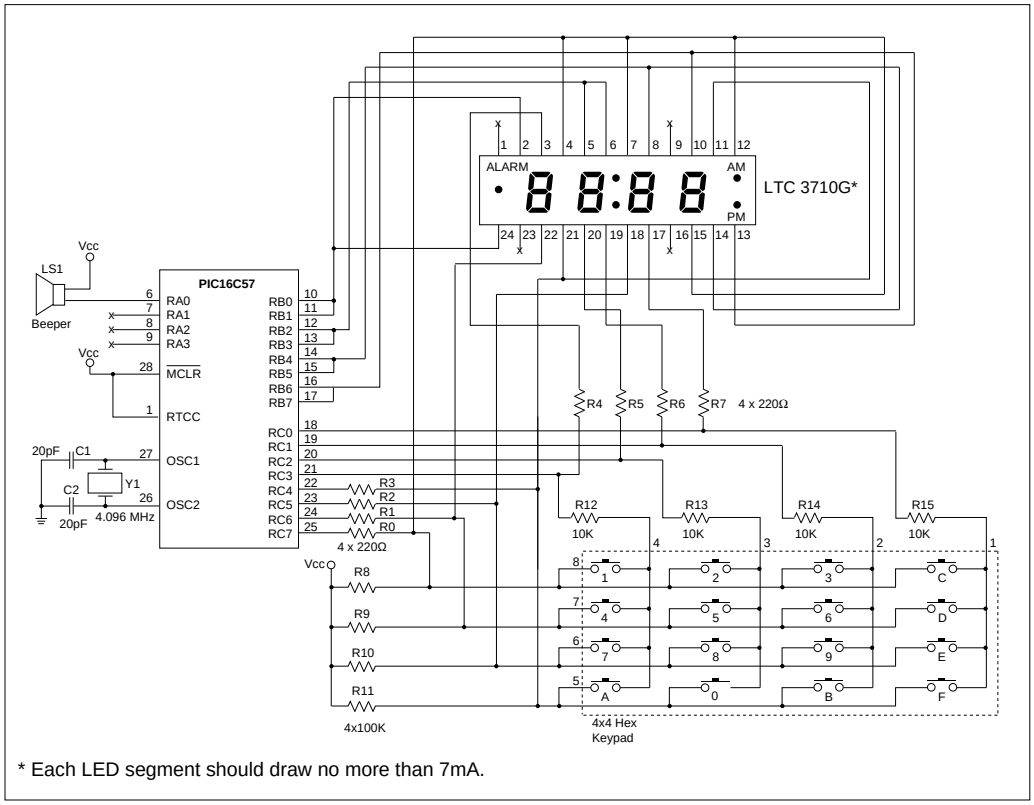
The keypad sampling is as follows:

1. The columns are connected to output pins, and the rows are connected to input pins.
2. Each column is sequentially driven to a low voltage while at the same instance the four rows are sampled. Since the rows are all held high with pull-up resistors, all four inputs will normally be high. If a key is pressed in a column which is at a low level, that low level will be conducted to the input pin through the closed key and the corresponding row will be sensed as a low.
3. Before a new column is brought low, care should be taken to discharge the input pins (see code section for details).
4. A 50 ms key debounce technique has been implemented in the software, in order to eliminate multiple key strokes.

Notes:

1. Resistors R8-R11 and R12-R14 have been selected such that their ratio is 1:10. This will insure a 0.5 Volt level at the input, when a key is pressed. Also R8-R14 should have a value such that their current contribution to the LEDs segments is negligible.
2. In circuits where there is substantial interference between the key matrix and the LED drive circuit, the alternative circuit (Figure 2b) should be utilized. Diodes in the path of all pins connected to the keypad insure that there is minimal interference from the keypad, when it is not being sampled.

FIGURE 2A - PIC16C5X INDUSTRIAL CLOCK/TIMER SCHEMATIC



* Each LED segment should draw no more than 7mA.

PART B: INDUSTRIAL CLOCK/TIMER

Clock Selection

The 4.096 MHz crystal oscillator is the time base. The PIC16C5X internally divides the clock by 4 to give an internal clock of 1.024 MHz. This clock is further divided by 32 (by the prescaler in the OPTIONS register) to give a clock of 32 KHz which is used to increment the RTCC in the PIC16C5X. If the RTCC is initialized to 96, it would overflow to 0 in 5 ms.

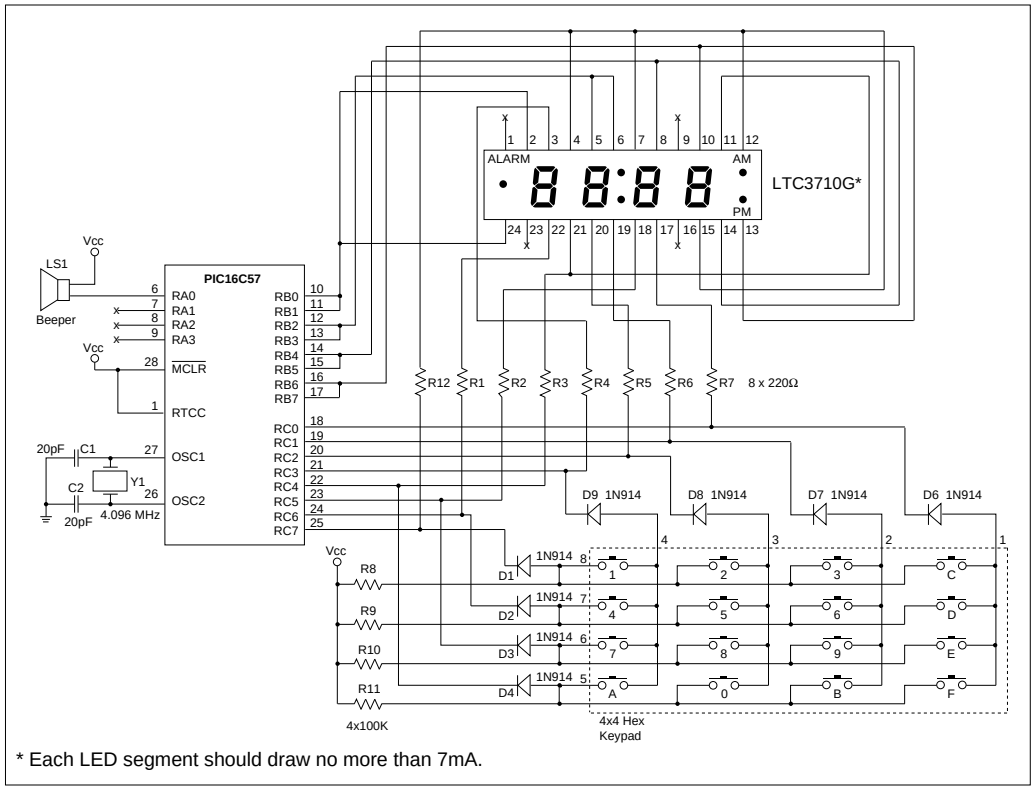
(256-96) x (1/32000) = 5.000 ms

This 5 ms is used to count the seconds, minutes and hours in the clock/timer. It is also used as a time base to update the display digits and sample the keyboard. The clock speed being 4.096 MHz, each instruction will execute in 1 µs. Therefore in 5 ms, approximately 5000 instructions can be executed. This gives sufficient time to execute a large section of code and not miss the overflow in the RTCC.

Using a 3.579545 MHz color burst crystal oscillator as a time base

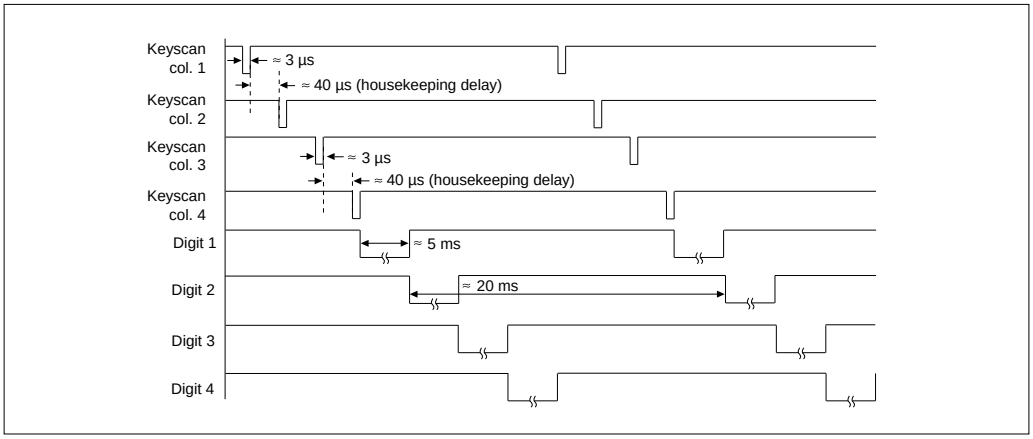
Some users may want to use a color burst crystal oscillator as a time base, because of its low cost. If a 3.579545 MHz crystal is used, then the internal clock will be 1.117 µs. If this is prescaled by 32, the RTCC will be incremented every 35.758 µs. Initializing the RTCC with 116 will cause it to overflow to 0 in 5.006 ms. giving an error of 0.12%. This error can be corrected in software by making time adjustments every minute and/or every hour.

FIGURE 2B - PIC16C5X ALARM CLOCK SCHEMATIC (USING DIODES)



* Each LED segment should draw no more than 7mA.

FIGURE 3 - KEY SCAN AND LED DIGIT SELECT TIMING



FEATURES

The Flow Chart (Figure 4) shows the sequence of events in the clock/timer software. The clock has the following features:

1. 12 hour clock with a.m./p.m.
2. 12 hour alarm with a.m./p.m.
3. Full function Hex keypad (Figure 5).
4. AA audible alarm for 1 minute.
5. 10 minute alarm disable.

SETTING CLOCK/TIMER FUNCTIONS

Function	Key Sequence to Activate Function
Set Real Time	Set → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
View Alarm Time	Alarm (alarm time is displayed for 5 seconds)
Set Alarm Time	Alarm → Set (must be pressed when alarm LED is flashing) → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
Enable/Disable Alarm	Alarm → Alarm (toggles alarm status)
Disable AA alarm	Disable Alarm (disable audible beep for 10 minutes)
Clear Alarm	Clear Alarm (clears audible alarm)
Abort Entry	Clear Entry (aborts data entry mode when setting real and alarm time)

Notes: 1. Valid key strokes will be acknowledged with a beep.

2. Hours and minutes used above correspond to digits 0 - 9 on the keypad.

FIGURE 4 - TIMER/CLOCK FLOW CHART

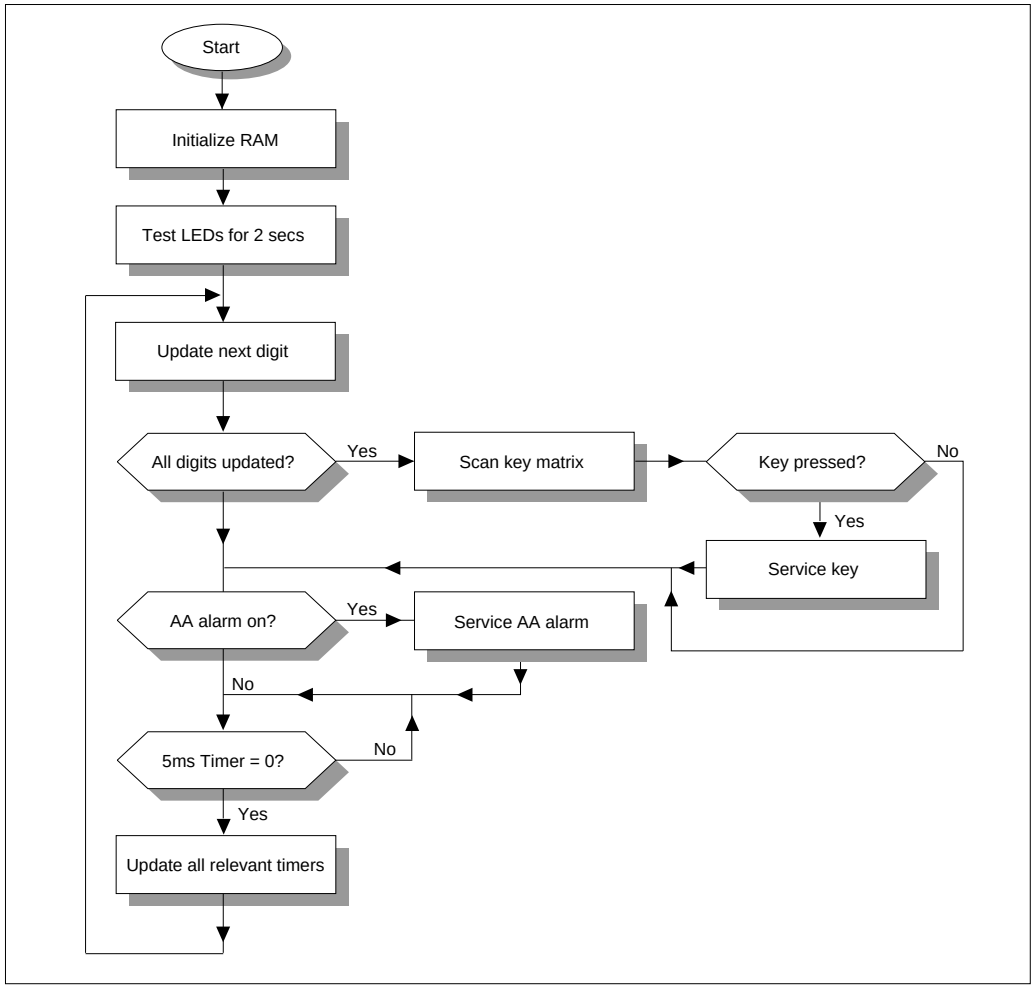
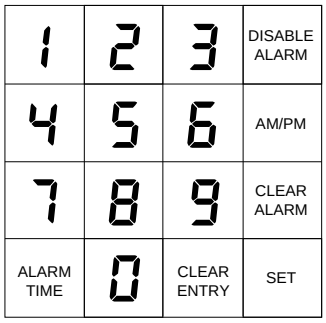
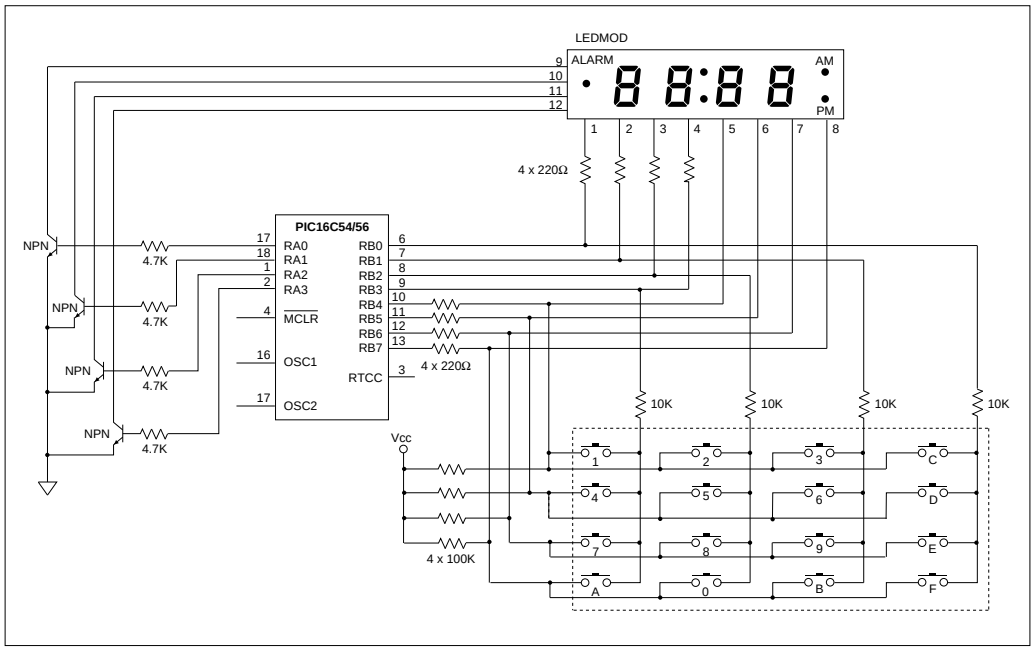


FIGURE 5 - KEYPAD



Multiplexing LED Drive and a 4x4 Keypad Sampling

FIGURE 6 - INTERFACE TO PIC16C54/56



SUMMARY

This Application Note demonstrates a simple method of interfacing the PIC16C5X to 7-segment LEDs and a keypad. The key features of the PIC16C5X which made this possible are:

1. High sink/source of the I/O ports.
2. Fast instruction cycle for quick key-scan.
3. RISC processor allowing minimal overhead for real time clock maintenance.
4. Reconfigurable I/O ports, enabling dual functionality of ports.

Figure 6 depicts a block diagram connecting a PIC16C54/56 to a 4-digit, 7-segment LED display and a 4x4 hex keypad. Since only 12 I/O pins are available in the PIC16C54/56, external npn transistor will have to be utilized to sink the current from each digit.

CODE SIZE

Key scan → 97 bytes

Display update → 113 bytes

Author: Stan D'Souza
Logic Products Division

APPENDIX A: CODE LISTING

```
MPASM 1.00 Released       CLK.ASM   7-15-1994  13:15:10                PAGE  1
Alarm Clock

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001                 TITLE        “Alarm Clock”
                     0002         LIST  P = 16C57,f=inhx8m
                     0003 ;
                     0004 ;Define Equates:
                     0005 ;
07FF                 0006         PIC57   EQU     7FFH
                     0007
;*****************************************************************
                     0008 ;External Ossc. used = 4.096Mhz. Prescaler of 32 used, which gives a
                     0009 ;31.25 microSec increment of the RTCC. If RTCC is intially loaded with
96,
                     0010 ;it would overflow to 0 in 5.000 milliSecs. Giving a 0.00% error.
0060                 0011         MSEC5   EQU     D’96'
                     0012
;*****************************************************************
0000                 0013         C       EQU     0
0000                 0014         BEP     EQU     0
0000                 0015         RTATS   EQU     0
0001                 0016         DC      EQU     1
0001                 0017         HR10    EQU     1
0002                 0018         Z       EQU     2
0002                 0019         HR      EQU     2
0003                 0020         MIN10   EQU     3
0004                 0021         MIN     EQU     4
0004                 0022         FLASH   EQU     4
0005                 0023         PA0     EQU     5
0005                 0024         KEY_BEEP EQU    5
0005                 0025         AMPM    EQU     5
0006                 0026         PA1     EQU     6
0000                 0027         F0      EQU     0
0006                 0028         KEY_HIT EQU     6
0006                 0029         ALED    EQU     6
0007                 0030         AM_PM   EQU     7
0003                 0031         COLON   EQU     3
0002                 0032         ALRMLED EQU     2
0007                 0033         SERVICED EQU    7
0000                 0034         ALONOF  EQU     0
0001                 0035         INAL    EQU     1
0002                 0036         SILNC   EQU     2
0003                 0037         INAA    EQU     3
0005                 0038         INKEYBEP EQU    5
                     0039 ;
                     0040 ;DEFINE RAM LOCATIONS:
0001                 0041         RTCC    EQU     1
0002                 0042         PC      EQU     2
0003                 0043         STATUS  EQU     3
0004                 0044         FSR     EQU     4
0005                 0045         PORT_A  EQU     5
0006                 0046         PORT_B  EQU     6
0007                 0047         PORT_C  EQU     7
                     0048 ;DEFINE REAL TIME MODE REGS (RTM)
0008                 0049         MSTMR   EQU     8       ;MILLI SEC. TIMER
0009                 0050         STMR    EQU     9       ;SEC. TIMER
                     0051 ;*************************************************
                     0052 ;DO NOT CHANGE RELATIVE POSITION OF NEXT 6 BYTES
000A                 0053         MTMR    EQU     0A      ;MIN. TIMER
000B                 0054         HTMR    EQU     0B      ;HOUR TIMER
                     0055 ;DEFINE ALARM TIME MODE REGS (ATM)
000C                 0056         MALARM  EQU     0C      ;MIN. ALARM
000D                 0057         HALARM  EQU     0D      ;HOUR ALARM
                     0058 ;DEFINE DATA ENTRY MODE REGS (DEM)
```



```
000E                0059         MENTRY  EQU     0E      ;MIN. ENTRY
000F                0060         HENTRY  EQU     0F      ;HOUR ENTRY
                    0061 ;*************************************************
                    0063 ;
                    0064 ;DEFINE FLAG REG AND FUNCTION:
0010                0065         FLAG    EQU     10
                    0066 ;       BIT # 7|6|5|4|3|2|1|0|
                    0067 ;————————|-|-|-|-|-|-|-|
                    0068 ;              X|X|X|X|X|X|0|0| —> REAL TIME MODE (RTM)
                    0069 ;              X|X|X|X|X|X|0|1| —> ALARM TIME MODE(ATM)
                    0070 ;              X|X|X|X|X|X|1|0| —> DATA ENTRY M0DE(DEM)
                    0071 ;              X|X|X|X|X|X|1|1| —> TEST MODE (TM)
                    0072 ;              X|X|X|X|X|Y|X|X| —> ALRMLED ON/OFF
                    0073 ;              X|X|X|X|Y|X|X|X| —> COLON LED ON/OFF
                    0074 ;              X|X|X|Y|X|X|X|X| —> FLASH DISPLAY
                    0075 ;              X|X|Y|X|X|X|X|X| —> KEY_BEEP
                    0076 ;              X|Y|X|X|X|X|X|X| —> KEY_HIT (0/1)
                    0077 ;              Y|X|X|X|X|X|X|X| —> SERVICED
                    0078 ; X = DEFINED ELSEWHERE IN TABLE
                    0079 ; Y = DEFINED AS SHOWN (0/1)
                    0080 ;
0011                0081         TEMP    EQU     11
0012                0082         DIGIT   EQU     12
0013                0083         NEW_KEY EQU     13
0014                0084         KEY_NIBL EQU    14
0015                0085         DEBOUNCE EQU    15
0016                0086         MIN_SEC EQU     16      ;MIN/SECONDS TIMER
0017                0087         ENTFLG  EQU     17
                    0088 ;flag dedicated to the key entry mode
                    0089 ;       BIT # 7|6|5|4|3|2|1|0|
                    0090 ;————————|-|-|-|-|-|-|-|
                    0091 ;              X|X|X|X|X|X|X|Y| —> REAL/ALARM TIME STATUS
                    0092 ;              X|X|X|X|X|X|Y|X| —> HR10 DONE
                    0093 ;              X|X|X|X|X|Y|X|X| —> HR DONE
                    0094 ;              X|X|X|X|Y|X|X|X| —> MIN10 DONE
                    0095 ;              X|X|X|Y|X|X|X|X| —> MIN DONE
                    0096 ;              X|X|Y|X|X|X|X|X| —> INKEYBEP
                    0097 ;              X|Y|X|Y|X|X|X|X| —> NOT USED
                    0098 ;              Y|X|X|X|X|X|X|X| —> NOT USED
                    0099 ;
                    0100 ;
0018                0101         ALFLAG  EQU     18
                    0102 ;flag dedicated to the alarm
                    0103 ;       BIT # 7|6|5|4|3|2|1|0|
                    0104 ;————————|-|-|-|-|-|-|-|
                    0105 ;              X|X|X|X|X|X|X|Y| —> ALONOF
                    0106 ;              X|X|X|X|X|X|Y|X| —> INAL
                    0107 ;              X|X|X|X|X|Y|X|X| —> SILNC
                    0108 ;              X|X|X|X|Y|X|X|X| —> INAA
                    0109 ;              X|X|X|Y|X|X|X|X| —> NOT USED
                    0110 ;              X|X|Y|X|X|X|X|X| —> NOT USED
                    0111 ;              X|Y|X|Y|X|X|X|X| —> NOT USED
                    0112 ;              Y|X|X|X|X|X|X|X| —> NOT USED
                    0113 ;
0019                0114         AAFLAG  EQU     19
                    0115 ;flag dedicated to the AA alarm
001A                0116         AATMR   EQU     1A
                    0117 ;
                    0118 ;Port pin definitions:
                    0119 ;
                    0120 ;PORT_A:
                    0121 ;       BIT 0   —> BEEPER (ACTIVE LOW) OUTPUT
                    0122 ;       BIT 1-3 —> unused I/O
                    0123 ;
                    0124 ;PORT_B: ALL OUTPUTS
                    0125 ;       BIT 0&4 —> MSB DIGIT COMMON CATHODE & ALARM
                    0126 ;       BIT 1&5 —> 2ND DIGIT COMMOM CATHODE & COLON
                    0127 ;       BIT 2&6 —> 3RD DIGIT COMMON CATHODE & PM
                    0128 ;       BIT 3&7 —> LSB DIGIT COMMON CATHODE & AM
```

```
                    0129 ;
                    0130 ;PORT_C:
                    0131 ;IN DISPLAY MODE ALL SEG/ANNN SET AS OUTPUTS
                    0132 ;IN KEY SCAN MODE COLS ARE OUTPUTS ROWS ARE INPUTS
                    0133 ;       BIT 0   -> SEGMENT A & COL 4
                    0134 ;       BIT 1   -> SEGMENT B & COL 3
                    0135 ;       BIT 2   -> SEGMENT C & COL 2
                    0136 ;       BIT 3   -> SEGMENT D & COL 1
                    0137 ;       BIT 4   -> SEGMENT E & ROW 4
                    0138 ;       BIT 5   -> SEGMENT F & ROW 3
                    0139 ;       BIT 6   -> SEGMENT G & ROW 2
                    0140 ;       BIT 7   -> CA OF ALL ANNUNCIATORS & ROW 1
                    0141 ;
                    0142 ;
                    0144 ;
                    0145         ORG     0
                    0146 START
0000 0AFC           0147         GOTO    INIT_CLK        ;INITIALIZE CLOCK
                    0148 ;THIS ROUTINE RUNS A TEST ON THE LEDS.
                    0149 ;ALL THE RELEVENT LEDS ARE LIT UP FOR 2 SECS.
                    0150 ;
                    0151 TEST_HARDWARE
0001 0C02           0152         MOVLW   d'02'           ;DISPLAY FOR 2 SECS
0002 0036           0153         MOVWF   MIN_SEC         ;       /
                    0154 ;
                    0155 ;
                    0156 TEST_LOOP
0003 0216           0157         MOVF    MIN_SEC,W       ;GET MIN/SEC
0004 0643           0158         BTFSC   STATUS,Z        ;NOT 0 THEN SKIP
0005 0A0B           0159         GOTO    NORM_TIME       ;ELSE NORMAL TIME
0006 0925           0160         CALL    UPDATE_DISPLAY  ;UPDATE DISPLAY
0007 05A3           0161         BSF     STATUS,PA0      ;GOTO PAGE 1
0008 0900           0162         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE
0009 04A3           0163         BCF     STATUS,PA0      ;RESET PAGE MARKER
000A 0A03           0164         GOTO    TEST_LOOP       ;LOOP BACK
                    0165 NORM_TIME
000B 0410           0166         BCF     FLAG,0          ;PUT IN REAL TIME
000C 0430           0167         BCF     FLAG,1
                    0168 TIME_LOOP
000D 0925           0169         CALL    UPDATE_DISPLAY
000E 05C3           0170         BSF     STATUS,PA1      ;GOTO PAGE 2
000F 0900           0171         CALL    SERVICE_KEYS
0010 05A3           0172         BSF     STATUS,PA0      ;GOTO PAGE 3
0011 0900           0173         CALL    SOUND_AA        ;CHECK ALARM
0012 04C3           0174         BCF     STATUS,PA1      ;GOTO PAGE 1
0013 0900           0175         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE TIMERS
0014 04A3           0176         BCF     STATUS,PA0      ;RESET PAGE MARKER
0015 04C3           0177         BCF     STATUS,PA1      ;       /
0016 0210           0178         MOVF    FLAG,W          ;SEE IF IN ATM
0017 0E03           0179         ANDLW   B'00000011'     ;       /
0018 0F01           0180         XORLW   B'00000001'     ;      /
0019 0643           0181         BTFSC   STATUS,Z        ;SKIP IF NOT
001A 091C           0182         CALL    RESET_ATM
001B 0A0D           0183         GOTO    TIME_LOOP
                    0184 ;
                    0185 RESET_ATM
001C 0216           0186         MOVF    MIN_SEC,W       ;GET MIN/SEC
001D 0E0F           0187         ANDLW   B'00001111'     ;       /
001E 0743           0188         BTFSS   STATUS,Z        ;Z THEN SKIP
001F 0800           0189         RETLW   0               ;ELSE RETURN
0020 0410           0190         BCF     FLAG,0          ;SET TO RTM
0021 0450           0191         BCF     FLAG,ALRMLED    ;CLEAR LED
0022 0618           0192         BTFSC   ALFLAG,ALONOF   ;TEST STAT
0023 0550           0193         BSF     FLAG,ALRMLED    ;SET LED
0024 0800           0194         RETLW   0               ;RETURN
                    0196 ;
                    0197 ;
                    0198 UPDATE_DISPLAY
0025 0C00           0199         MOVLW   B'00000000'     ;CLEAR SEG DRIVE
```

```
0026 0027           0200         MOVWF   PORT_C          ;       /
0027 0C3F           0201         MOVLW   B'00111111'     ;SEE IF LAST DIGIT
0028 0186           0202         XORWF   PORT_B,0        ;       /
0029 0643           0203         BTFSC   STATUS,Z        ;NO THEN SKIP
002A 0A6F           0204         GOTO    SCAN_KP         ;ELSE SCAN KEYPAD
                    0205 UP_DSP_1
                    0206 ;SELECT DIGIT TO BE DISPLAYED
002B 0246           0207         COMF    PORT_B,0        ;GET COMPL. PORT B IN W
002C 0643           0208         BTFSC   STATUS,Z        ;NO DIGIT SELECTED?
002D 0CC0           0209         MOVLW   B'11000000'     ;THEN SELECT DEFAULT
002E 0031           0210         MOVWF   TEMP            ;SAVE IN TEMP
002F 0271           0211         COMF    TEMP            ;COMPLEMENT VALUE
0030 0503           0212         BSF     STATUS,C        ;SET CARRY
0031 0371           0213         RLF     TEMP            ;SHIFT LEFT
0032 0703           0214         BTFSS   STATUS,C        ;IF C=1 THEN SKIP
0033 0371           0215         RLF     TEMP            ;ELSE 3 TIMES...
0034 0371           0216         RLF     TEMP            ;THRU CARRY
0035 0211           0217         MOVF    TEMP,0          ;GET IN W
0036 0026           0218         MOVWF   PORT_B          ;OUTPUT TO PORT
                    0219 ;NOW THAT DIGIT IS SELECTED, SELECT SEG VALUES FOR THAT DIGIT
                    0220 ;FIRST FIND MODE OF OPERATION.
0037 0C0A           0221         MOVLW   MTMR            ;LOAD FSR WITH MTMR
0038 0024           0222         MOVWF   FSR             ;       /
0039 0210           0223         MOVF    FLAG,0          ;GET FLAG IN W
003A 0E03           0224         ANDLW   B'00000011'     ;MASK OTHER BITS
003B 0031           0225         MOVWF   TEMP            ;SAVE IN TEMP
003C 0F03           0226         XORLW   B'00000011'     ;IN TEST MODE
003D 0643           0227         BTFSC   STATUS,Z        ;NO THEN SKIP
003E 0A4B           0228         GOTO    DO_TM           ;ELSE TEST MODE
003F 0403           0229         BCF     STATUS,C        ;CLEAR CARRY
0040 0371           0230         RLF     TEMP            ;LEFT SHIFT TEMP
0041 0211           0231         MOVF    TEMP,0          ;GET IN W
0042 01E4           0232         ADDWF   FSR             ;CHANGE INDIRECT POINTER
0043 0954           0233         CALL    GET_7_SEG       ;GET 7 SEG DATA IN W
0044 0032           0234         MOVWF   DIGIT           ;SAVE IN DIGIT LOC.
0045 09D1           0235         CALL    MASK_ANNC       ;MASK ANNC TO DIGIT
0046 0690           0236         BTFSC   FLAG,FLASH      ;NO FLASH THEN SKIP
0047 094E           0237         CALL    CHK_HALF_SEC    ;ELSE CHK. IF ON
0048 0212           0238         MOVF    DIGIT,0         ;GET BACK DIGIT
0049 0027           0239         MOVWF   PORT_C          ;OUTPUT TO PORT
004A 0800           0240         RETLW   0               ;RETURN
                    0241 ;
                    0242 DO_TM
004B 0CFF           0243         MOVLW   B'11111111'     ;LIGHT ALL SEGMENTS
004C 0027           0244         MOVWF   PORT_C          ;       /
004D 0800           0245         RETLW   0               ;RETURN FROM UPDATE DISPLAY
                    0246 ;
                    0247 CHK_HALF_SEC
004E 0770           0248         BTFSS   FLAG,COLON      ;IF COLON ON THEN DO
004F 0A51           0249         GOTO    BLANK_DSP       ;ELSE BLANK DISPLAY
0050 0800           0250         RETLW   0
                    0251 BLANK_DSP
0051 0C00           0252         MOVLW   B'00000000'     ;MAKE PORT C LOW
0052 0032           0253         MOVWF   DIGIT
0053 0800           0254         RETLW   0
                    0255 ;
                    0257 ;
                    0258 ;ON ENTRY FSR POINTS TO THE REAL TIME MODE'S MINUTES REGISTER.
                    0259 ;ON RETURN FSR POINTS TO THE TIMER REGISTER TO BE DISPLAYED.
                    0260 ;W REG. CONTAINS THE DECODED 7 SEG. INFO OF THE DIGIT
                    0261 ;TO BE DISPLAYED
                    0262 ;
                    0263 GET_7_SEG
0054 0246           0264         COMF    PORT_B,0        ;COMPLEMENT B -> W
0055 0EF0           0265         ANDLW   B'11110000'     ;MASK LO NIBBLE
0056 0643           0266         BTFSC   STATUS,Z        ;NZ THEN SKIP
0057 02A4           0267         INCF    FSR             ;INC POINTER
0058 0200           0268         MOVF    F0,0            ;MOVE INDIRECT TO W
0059 0031           0269         MOVWF   TEMP            ;GET INTO TEMP
```

```
005A 0246           0270         COMF    PORT_B,0        ;COMPL.B —> W
005B 0EF0           0271         ANDLW   B'11110000'     ;MASK LO NIBBLE
005C 0643           0272         BTFSC   STATUS,Z        ;IF D1/2 THEN
005D 04F1           0273         BCF     TEMP,AM_PM      ;CLEAR AM/PM BIT
005E 0246           0274         COMF    PORT_B,0        ;GET PORT B AGAIN
005F 0ECC           0275         ANDLW   B'11001100'     ;SEE IF D2 OR D4
0060 0643           0276         BTFSC   STATUS,Z        ;YES THEN SKIP
0061 03B1           0277         SWAPF   TEMP            ;SWAP TEMP
0062 0C0F           0278         MOVLW   B'00001111'     ;MASK HI NIBBLE
0063 0151           0279         ANDWF   TEMP,0
0064 01E2           0280         ADDWF   PC              ;ADD TO PC
0065 083F           0281         RETLW   B'00111111'     ;CODE FOR 0
0066 0806           0282         RETLW   B'00000110'     ;CODE FOR 1
0067 085B           0283         RETLW   B'01011011'     ;CODE FOR 2
0068 084F           0284         RETLW   B'01001111'     ;CODE FOR 3
0069 0866           0285         RETLW   B'01100110'     ;CODE FOR 4
006A 086D           0286         RETLW   B'01101101'     ;CODE FOR 5
006B 087D           0287         RETLW   B'01111101'     ;CODE FOR 6
006C 0807           0288         RETLW   B'00000111'     ;CODE FOR 7
006D 087F           0289         RETLW   B'01111111'     ;CODE FOR 8
006E 0867           0290         RETLW   B'01100111'     ;CODE FOR 9
                    0291 ;
                    0292 ;This routine scans the 4x4 hex key pad for a key hit.
                    0293 ;If key is pressed, KEY_HIT flag is set and the value of
                    0294 ;the hex key is returned in reg NEW_KEY
                    0295 ;If no key is detected, then a 0xff value is returned in
                    0296 ;register NEW_KEY and the flag KEY_HIT is reset.
                    0297 ;
                    0298 SCAN_KP
006F 06D0           0299         BTFSC   FLAG,KEY_HIT    ;KEY UNDER SERVICE?
0070 0A2B           0300         GOTO    UP_DSP_1        ;YES SKIP ROUTINE
0071 0CFF           0301         MOVLW   B'11111111'     ;SET DIGIT SINKS ...
0072 0026           0302         MOVWF   PORT_B          ;TO HIGH
0073 0CF7           0303         MOVLW   B'11110111'     ;SET KEY COL LOW
0074 0031           0304         MOVWF   TEMP            ;SAVE IN TEMP
                    0305 SKP1
0075 0C00           0306         MOVLW   B'00000000'     ;SET PORT C AS OUTPUTS
0076 0007           0307         TRIS    PORT_C          ;       /
0077 0211           0308         MOVF    TEMP,W
0078 0E0F           0309         ANDLW   B'00001111'     ;DISCHARGE PINS
0079 0027           0310         MOVWF   PORT_C          ;       /
007A 0CF0           0311         MOVLW   B'11110000'     ;SET AS I/O
007B 0007           0312         TRIS    PORT_C          ;   /
007C 0211           0313         MOVF    TEMP,W          ;GET OLD VALUE
007D 0027           0314         MOVWF   PORT_C          ;OUTPUT TO PORT
007E 0207           0315         MOVF    PORT_C,W        ;INPUT PORT VALUE
007F 0EF0           0316         ANDLW   B'11110000'     ;MASK LO BYTE
0080 0FF0           0317         XORLW   B'11110000'     ;SEE IF KEY HIT
0081 0743           0318         BTFSS   STATUS,Z        ;NO KEY THEN SKIP
0082 0A8D           0319         GOTO    DET_KEY         ;LOAD KEY VALUE
                    0320 SKP3
0083 0503           0321         BSF     STATUS,C        ;SET CARRY
0084 0331           0322         RRF     TEMP            ;MAKE NEXT COL. LOW
0085 0603           0323         BTFSC   STATUS,C        ;ALL DONE THEN SKIP
0086 0A75           0324         GOTO    SKP1
0087 0073           0325         CLRF    NEW_KEY         ;SET NEW_KEY = FF
0088 00F3           0326         DECF    NEW_KEY         ;       /
                    0327 SKP2
0089 0067           0328         CLRF    PORT_C          ;SETPORT C AS ...
008A 0C00           0329         MOVLW   B'00000000'     ;OUTPUTS
008B 0007           0330         TRIS    PORT_C          ;       /
008C 0A2B           0331         GOTO    UP_DSP_1        ;RETURN
                    0332 DET_KEY
                    0333 ;key is detected
008D 0293           0334         INCF    NEW_KEY,W       ;CHK IF KEY ...
008E 0743           0335         BTFSS   STATUS,Z        ;WAS RELEASED
008F 0A89           0336         GOTO    SKP2            ;NO THEN RETURN
0090 0207           0337         MOVF    PORT_C,W        ;GET RAW KEY...
0091 0D0F           0338         IORLW   B'00001111'     ;VALUE.
```

```
0092 0151           0339         ANDWF   TEMP,W          ;   /
0093 0033           0340         MOVWF   NEW_KEY         ;SAVE IN NEW_KEY
0094 0998           0341         CALL    GET_KEY_VAL     ;GET ACTUAL KEY ...
0095 0033           0342         MOVWF   NEW_KEY         ;VALUE
0096 05D0           0343         BSF     FLAG,KEY_HIT    ;SET KEY HIT FLAG
0097 0A89           0344         GOTO    SKP2            ;RETURN
                    0345 ;
                    0347 ;This routine decodes the hex value from the “raw” data got
                    0348 ;from scanning the rows and cols.
                    0349 ;   actual key value          raw hex value
                    0350 ;       ONE             EQU     77
                    0351 ;       TWO             EQU     7B
                    0352 ;       THREE           EQU     7D
                    0353 ;       C               EQU     7E
                    0354 ;       FOUR            EQU     0B7
                    0355 ;       FIVE            EQU     0BB
                    0356 ;       SIX             EQU     0BD
                    0357 ;       D               EQU     0BE
                    0358 ;       SEVEN           EQU     0D7
                    0359 ;       EIGHT           EQU     0DB
                    0360 ;       NINE            EQU     0DD
                    0361 ;       E               EQU     0DE
                    0362 ;       A               EQU     0E7
                    0363 ;       ZERO            EQU     0EB
                    0364 ;       B               EQU     0ED
                    0365 ;       F               EQU     0EE
                    0366 ;
                    0367 ;
                    0368 GET_KEY_VAL
0098 0E0F           0369         ANDLW   B’00001111'     ;GET LO NIBBLE
0099 0034           0370         MOVWF   KEY_NIBL        ;SAVE
009A 0C04           0371         MOVLW   4               ;SET COUNT TO 4
009B 0031           0372         MOVWF   TEMP            ;       /
                    0373 GKV1
009C 0503           0374         BSF     STATUS,C        ;SET CARRY
009D 0334           0375         RRF     KEY_NIBL        ;ROTATE NIBBLE
009E 0703           0376         BTFSS   STATUS,C        ;SKIP IF NOT Z
009F 0AA5           0377         GOTO    GET_HI_KEY      ;GOTO NEXT PART
00A0 02F1           0378         DECFSZ  TEMP            ;DEC COUNT
00A1 0A9C           0379         GOTO    GKV1            ;LOOP
                    0380 GO_RESET
00A2 05A3           0381         BSF     STATUS,PA0      ;SET MSB
00A3 05C3           0382         BSF     STATUS,PA1      ;       /
00A4 0BFF           0383         GOTO    SYS_RESET       ;ELSE BIG ERROR
                    0384 GET_HI_KEY
00A5 00F1           0385         DECF    TEMP            ;REDUCE BY 1
00A6 0393           0386         SWAPF   NEW_KEY,W       ;GET HI NIBBLE
00A7 0E0F           0387         ANDLW   B’00001111'     ;       /
00A8 0034           0388         MOVWF   KEY_NIBL        ;SAVE
00A9 0211           0389         MOVF    TEMP,W          ;GET OFFSET TO TBL
00AA 01E2           0390         ADDWF   PC              ;LOAD IN PC
00AB 0AAF           0391         GOTO    GET147A         ;JUMP TO NEXT PART
00AC 0AB8           0392         GOTO    GET2580         ;       /
00AD 0ABA           0393         GOTO    GET369B         ;     /
00AE 0ABC           0394         GOTO    GETCDEF         ;   /
                    0395 ;
                    0396 GET147A
00AF 0C04           0397         MOVLW   4               ;SET COUNT TO 4
                    0398 GETCOM
00B0 0031           0399         MOVWF   TEMP            ;
                    0400 GETCOM1
00B1 0503           0401         BSF     STATUS,C        ;SET CARRY
00B2 0334           0402         RRF     KEY_NIBL        ;ROTATE RIGHT
00B3 0703           0403         BTFSS   STATUS,C        ;CHECK IF DONE
00B4 0ABE           0404         GOTO    KEY_TBL         ;JUMP TO TABLE
00B5 02F1           0405         DECFSZ  TEMP            ;DEC COUNT
00B6 0AB1           0406         GOTO    GETCOM1         ;LOOP
00B7 0AA2           0407         GOTO    GO_RESET        ;ELSE ERROR
                    0408 ;
```

```
                    0409 GET2580
00B8 0C08           0410         MOVLW   8               ;SET COUNT TO 8
00B9 0AB0           0411         GOTO    GETCOM
                    0412 ;
                    0413 GET369B
00BA 0C0C           0414         MOVLW   D'12'           ;SET COUNT TO 12
00BB 0AB0           0415         GOTO    GETCOM
                    0416 ;
                    0417 GETCDEF
00BC 0C10           0418         MOVLW   D'16'           ;SET COUNT TO 16
00BD 0AB0           0419         GOTO    GETCOM
                    0421 ;
                    0422 KEY_TBL
00BE 00F1           0423         DECF    TEMP            ;REDUCE BY 1
00BF 0211           0424         MOVF    TEMP,W          ;GET IN W
00C0 01E2           0425         ADDWF   PC              ;JUMP TO TABLE
00C1 0801           0426         RETLW   1               ;KEY 1
00C2 0804           0427         RETLW   4               ;KEY 4
00C3 0807           0428         RETLW   7               ;KEY 7
00C4 080A           0429         RETLW   0A              ;KEY A
00C5 0802           0430         RETLW   2               ;KEY 2
00C6 0805           0431         RETLW   5               ;KEY 5
00C7 0808           0432         RETLW   8               ;KEY 8
00C8 0800           0433         RETLW   0               ;KEY 0
00C9 0803           0434         RETLW   3               ;KEY 3
00CA 0806           0435         RETLW   6               ;KEY 6
00CB 0809           0436         RETLW   9               ;KEY 9
00CC 080B           0437         RETLW   0B              ;KEY B
00CD 080C           0438         RETLW   0C              ;KEY C
00CE 080D           0439         RETLW   0D              ;KEY D
00CF 080E           0440         RETLW   0E              ;KEY E
00D0 080F           0441         RETLW   0F              ;KEY F
                    0442 ;
                    0444 ;
                    0445 MASK_ANNC
00D1 0CFC           0446         MOVLW   B'11111100'     ;CHK IF DIGIT 1
00D2 0186           0447         XORWF   PORT_B,0        ;       /
00D3 0643           0448         BTFSC   STATUS,Z        ;NO THEN SKIP
00D4 0AE5           0449         GOTO    MASK_ALARM      ;ELSE MASK ALARM
00D5 0CF3           0450         MOVLW   B'11110011'     ;CHK IF DIGIT 2
00D6 0186           0451         XORWF   PORT_B,0        ;       /
00D7 0643           0452         BTFSC   STATUS,Z        ;NO THEN SKIP
00D8 0AE8           0453         GOTO    MASK_COLON      ;ELSE MASK COLON
00D9 0CCF           0454         MOVLW   B'11001111'     ;CHK IF DIGIT 3
00DA 0186           0455         XORWF   PORT_B,0        ;       /
00DB 0643           0456         BTFSC   STATUS,Z        ;NO THEN SKIP
00DC 0AE1           0457         GOTO    MASK_PM         ;ELSE MASK PM
                    0458 MASK_AM
00DD 02A4           0459         INCF    FSR             ;INC FSR
00DE 07E0           0460         BTFSS   F0,AM_PM        ;IF 0 THEN AM
00DF 05F2           0461         BSF     DIGIT,7         ;SET MSB
00E0 0AEB           0462         GOTO    BLNK_LEAD_0     ;NEXT
                    0463 MASK_PM
00E1 02A4           0464         INCF    FSR             ;INC FSR
00E2 06E0           0465         BTFSC   F0,AM_PM        ;IF 1 THEN PM
00E3 05F2           0466         BSF     DIGIT,7         ;SET MSB
00E4 0AEB           0467         GOTO    BLNK_LEAD_0     ;NEXT
                    0468 MASK_ALARM
00E5 0650           0469         BTFSC   FLAG,ALRMLED    ;1 THEN LIGHT LED
00E6 05F2           0470         BSF     DIGIT,7         ;      /
00E7 0AEB           0471         GOTO    BLNK_LEAD_0     ;NEXT
                    0472 MASK_COLON
00E8 0670           0473         BTFSC   FLAG,COLON      ;1 THEN LIGHT LED
00E9 05F2           0474         BSF     DIGIT,7         ;       /
00EA 0AEB           0475         GOTO    BLNK_LEAD_0     ;NEXT
                    0476 ;
                    0477 BLNK_LEAD_0
00EB 0210           0478         MOVF    FLAG,W          ;GET IN W
00EC 0E03           0479         ANDLW   B'00000011'     ;SEE IF IN DEM
```

```
00ED 0F02           0480         XORLW   B'00000010'     ;CHECK
00EE 0643           0481         BTFSC   STATUS,Z        ;NO THEN DO
00EF 0800           0482         RETLW   0               ;ELSE RETURN
00F0 0CFC           0483         MOVLW   B'11111100'     ;SEE IF DIGIT 1
00F1 0186           0484         XORWF   PORT_B,0        ;       /
00F2 0743           0485         BTFSS   STATUS,Z        ;YES THEN SKIP
00F3 0800           0486         RETLW   0               ;RETURN
00F4 0C3F           0487         MOVLW   B'00111111'     ;ELSE MASK G AND ANUNC
00F5 0152           0488         ANDWF   DIGIT,0         ;GET IN W
00F6 0F3F           0489         XORLW   B'00111111'     ;SEE IF 0
00F7 0743           0490         BTFSS   STATUS,Z        ;YES THEN SKIP
00F8 0800           0491         RETLW   0               ;RETURN
00F9 0C80           0492         MOVLW   B'10000000'     ;ELSE BLANK D1
00FA 0172           0493         ANDWF   DIGIT           ;       /
00FB 0800           0494         RETLW   0               ;RETURN
                    0495 ;
                    0496 ;
                    0497 ;
                    0499 ;
                    0500 ;THIS ROUTINE SETS UP PORTS A,B,C AND THE INTERNAL
                    0501 ;REAL TIME CLOCK COUNTER.
                    0502 INIT_CLK
00FC 0C0F           0503         MOVLW   B'00001111'     ;MAKE ACTIVE HIGH
00FD 0025           0504         MOVWF   PORT_A          ;       /
00FE 0C00           0505         MOVLW   B'00000000'     ;SET PORT A AS OUTPUTS
00FF 0005           0506         TRIS    PORT_A
                    0507 ;
0100 0CFF           0508         MOVLW   B'11111111'     ;SET LEVELS HIGH
0101 0026           0509         MOVWF   PORT_B          ;       /
0102 0C00           0510         MOVLW   B'00000000'     ;SET PORT B AS OUTPUTS
0103 0006           0511         TRIS    PORT_B
                    0512 ;
0104 0C00           0513         MOVLW   B'00000000'     ;SET LEVELS LOW
0105 0027           0514         MOVWF   PORT_C          ;       /
0106 0C00           0515         MOVLW   B'00000000'     ;SET PORT C AS OUTPUTS
0107 0007           0516         TRIS    PORT_C          ;       /
                    0517 ;
0108 0C04           0518         MOVLW   B'00000100'     ;SET UP PRESCALER
0109 0002           0519         OPTION                  ;       /
                    0520 ;
010A 0C60           0521         MOVLW   MSEC5           ;RTCC = 5 mSEC
010B 0021           0522         MOVWF   RTCC            ;       /
010C 0068           0523         CLRF    MSTMR           ;CLEAR MSTMR
010D 0069           0524         CLRF    STMR            ; & SEC TMR
010E 006A           0525         CLRF    MTMR            ;& MINUTES
010F 0C12           0526         MOVLW   12H             ;MAKE HRS = 12
0110 002B           0527         MOVWF   HTMR            ;     /
0111 002D           0528         MOVWF   HALARM          ;MAKE HRS = 12
0112 006C           0529         CLRF    MALARM          ;       /
0113 0C03           0530         MOVLW   B'00000011'     ;SET TO TEST MODE
0114 0030           0531         MOVWF   FLAG            ;       /
0115 0078           0532         CLRF    ALFLAG          ;CLEAR ALL FLAG
0116 0079           0533         CLRF    AAFLAG          ;       /
0117 0077           0534         CLRF    ENTFLG          ;      /
0118 0A01           0535         GOTO    TEST_HARDWARE
                    0536 ;
                    0537 ;All routines related to timer updates are located at
                    0538 ;address 200 and above.
                    0540         ORG     0200
                    0541 ;
                    0542 UPDATE_TIMERS
0200 0201           0543         MOVF    RTCC,W          ;SEE IF RTCC = 0
0201 0743           0544         BTFSS   STATUS,Z        ;IF 0 THEN SKIP
0202 0A00           0545         GOTO    UPDATE_TIMERS   ;ELSE LOOP
0203 0C60           0546         MOVLW   MSEC5           ;RTCC = 5 mSEC
0204 0021           0547         MOVWF   RTCC            ;       /
0205 02A8           0548         INCF    MSTMR           ;INC 5 MILLI SEC
0206 06D0           0549         BTFSC   FLAG,KEY_HIT    ;NO KEY HIT THEN SKIP
0207 0A70           0550         GOTO    CHK_DE_BOUNCE   ;ELSE DEBOUNCE
```

```
                    0551 UP_TMR_1
0208 0210           0552         MOVF    FLAG,W          ;ALARM MODE?
0209 0E03           0553         ANDLW   B'00000011'     ;       /
020A 0F01           0554         XORLW   B'00000001'     ;       /
020B 0743           0555         BTFSS   STATUS,Z        ;SKIP IF YES
020C 0A14           0556         GOTO    UP_TMR_2        ;DO NEXT
020D 0550           0557         BSF     FLAG,ALRMLED    ;LIGHT LED
020E 0570           0558         BSF     FLAG,COLON      ;       /
020F 0C64           0559         MOVLW   D'100'          ;IF 1/2 SEC
0210 0088           0560         SUBWF   MSTMR,0         ; BLINK
0211 0703           0561         BTFSS   STATUS,C        ;  /
0212 0450           0562         BCF     FLAG,ALRMLED    ;ALARM LED
0213 0A19           0563         GOTO    UP_TMR_3        ;SKIP
                    0564 UP_TMR_2
0214 0570           0565         BSF     FLAG,COLON      ;TURN ON
0215 0C64           0566         MOVLW   D'100'          ;<100 BLINK COLON
0216 0088           0567         SUBWF   MSTMR,0         ;       /
0217 0703           0568         BTFSS   STATUS,C        ;YES THEN SKIP
0218 0470           0569         BCF     FLAG,COLON      ;ELSE TURN OFF
                    0570 UP_TMR_3
0219 0208           0571         MOVF    MSTMR,0         ;GET MSTMR IN W
021A 0FC8           0572         XORLW   D'200'          ;= 200 THEN SKIP
021B 0743           0573         BTFSS   STATUS,Z        ;       /
021C 0800           0574         RETLW   0
                    0575 ;INC SECONDS COUNT
021D 0068           0576         CLRF    MSTMR           ;CLEAR MS_TMR
021E 0216           0577         MOVF    MIN_SEC,W       ;GET MIN_SEC TIMER
021F 0E0F           0578         ANDLW   B'00001111'     ;MASK MINUTES
0220 0743           0579         BTFSS   STATUS,Z        ;ZERO THEN SKIP
0221 00F6           0580         DECF    MIN_SEC         ;REDUCE SECONDS
0222 0C09           0581         MOVLW   STMR            ;LOAD FSR WITH S_TMR
0223 0024           0582         MOVWF   FSR             ;       /
0224 0955           0583         CALL    INC_60          ;INC SECONDS
0225 0D00           0584         IORLW   0               ;DO AN OPERATION
0226 0743           0585         BTFSS   STATUS,Z        ;IF RETURN = 0 SKIP
0227 0A38           0586         GOTO    CHK_AL_TIM      ;CHK ALRM
                    0587 ;INC MINUTES COUNT
0228 03B6           0588         SWAPF   MIN_SEC         ;SWAP MIN SEC
0229 0216           0589         MOVF    MIN_SEC,W       ;GET MIN_SEC IN W
022A 0E0F           0590         ANDLW   B'00001111'     ;MASK SECONDS
022B 0743           0591         BTFSS   STATUS,Z        ;SKIP IF NOT SET
022C 00F6           0592         DECF    MIN_SEC         ;ELSE DEC
022D 03B6           0593         SWAPF   MIN_SEC         ;SWAP BACK
022E 0966           0594         CALL    CHK_SILNC_TIM   ;SILNCE ON?
022F 0C0A           0595         MOVLW   MTMR            ;INC MINUTES
0230 0024           0596         MOVWF   FSR             ;       /
0231 0955           0597         CALL    INC_60          ;       /
0232 0D00           0598         IORLW   0               ;DO AN OPERATION
0233 0743           0599         BTFSS   STATUS,Z        ;IF 0 THEN SKIP
0234 0A38           0600         GOTO    CHK_AL_TIM      ;CHECK ALRAM TIME
                    0601 ;INC HOUR COUNT
0235 0C0B           0602         MOVLW   HTMR            ;GET HTMR IN FSR
0236 0024           0603         MOVWF   FSR
0237 0989           0604         CALL    INC_HR          ;INC HOURS
                    0605 ;
                    0606 CHK_AL_TIM
0238 0718           0607         BTFSS   ALFLAG,ALONOF   ;IF OFF QUIT
0239 0800           0608         RETLW   0               ;       /
023A 0658           0609         BTFSC   ALFLAG,SILNC    ;RET IF IN SILENCE
023B 0800           0610         RETLW   0
023C 0638           0611         BTFSC   ALFLAG,INAL     ;ALREADY DONE
023D 0A4D           0612         GOTO    CHK_1_MIN       ;SEE IF 1 MIN UP
                    0613 ;        RETLW   0               ;YES THEN QUIT
023E 020D           0614         MOVF    HALARM,W        ;CHK HRS
023F 018B           0615         XORWF   HTMR,W          ;EQUAL?
0240 0743           0616         BTFSS   STATUS,Z        ;YES THEN SKIP
0241 0800           0617         RETLW   0               ;ELSE RET
0242 020C           0618         MOVF    MALARM,W        ;CHK MIN
0243 018A           0619         XORWF   MTMR,W          ;EQUAL?
```

```
0244 0743           0620         BTFSS   STATUS,Z        ;YES THEN SKIP
0245 0800           0621         RETLW   0               ;ELSE RET
0246 0209           0622         MOVF    STMR,W          ;SEE IF SEC=0
0247 0743           0623         BTFSS   STATUS,Z        ;YES THEN SKIP
0248 0800           0624         RETLW   0               ;NO THEN RET
0249 0538           0625         BSF     ALFLAG,INAL     ;SET IN ALARM FLAG
024A 0C10           0626         MOVLW   10              ;SET 1 MIN TIMER
024B 0036           0627         MOVWF   MIN_SEC         ;       /
024C 0800           0628         RETLW   0
                    0629 ;
                    0630 CHK_1_MIN
024D 0396           0631         SWAPF   MIN_SEC,W       ;SWAP IN W
024E 0E0F           0632         ANDLW   B'00001111'     ;CHK MINUTES
024F 0743           0633         BTFSS   STATUS,Z        ;0 THEN SKIP
0250 0800           0634         RETLW   0               ;ELSE RET
0251 0438           0635         BCF     ALFLAG,INAL     ;CLR IN ALARM
0252 0478           0636         BCF     ALFLAG,INAA     ;CLR IN AA
0253 0505           0637         BSF     PORT_A,BEP      ;STOP BEEPER
0254 0800           0638         RETLW   0
                    0639 ;
                    0640 INC_60
0255 02A0           0641         INCF    F0              ;INC AND GET IN W
0256 0200           0642         MOVF    F0,0            ;       /
0257 0E0F           0643         ANDLW   B'00001111'     ;MASK HI BITS
0258 0F0A           0644         XORLW   B'00001010'     ;= 10 THEN MAKE IT 0
0259 0743           0645         BTFSS   STATUS,Z        ;       /
025A 0801           0646         RETLW   1               ;ELSE RETURN NON ZERO
025B 0CF0           0647         MOVLW   B'11110000'     ;ZERO LSB
025C 0160           0648         ANDWF   F0              ;  /
025D 03A0           0649         SWAPF   F0              ;SWAP INDIRECT
025E 02A0           0650         INCF    F0              ;INC
025F 0200           0651         MOVF    F0,0            ;GET IN W
0260 03A0           0652         SWAPF   F0              ;SWAP F0 BACK
0261 0F06           0653         XORLW   D'6'            ;=6 THEN SKIP
0262 0743           0654         BTFSS   STATUS,Z        ;       /
0263 0801           0655         RETLW   1               ;ELSE RETURN NZ
0264 0060           0656         CLRF    F0              ;       /
0265 0800           0657         RETLW   0               ;RET 0
                    0658 ;
                    0660 ;
                    0661 CHK_SILNC_TIM
0266 0758           0662         BTFSS   ALFLAG,SILNC    ;CHK IF IN SILENCE
0267 0800           0663         RETLW   0               ;NO THEN SKIP
0268 0396           0664         SWAPF   MIN_SEC,W       ;GET MIN IN W
0269 0E0F           0665         ANDLW   B'00001111'     ;MASK SECS
026A 0743           0666         BTFSS   STATUS,Z        ;ZERO?
026B 0800           0667         RETLW   0               ;NO THEN RET
026C 0458           0668         BCF     ALFLAG,SILNC    ;RESET SILENCE
026D 0C10           0669         MOVLW   10              ;SET I MIN TIMER
026E 0036           0670         MOVWF   MIN_SEC         ;       /
026F 0800           0671         RETLW   0
                    0672 ;
                    0673 ;
                    0674 CHK_DE_BOUNCE
0270 06B7           0675         BTFSC   ENTFLG,INKEYBEP ;IN KEY BEEP?
0271 0A76           0676         GOTO    CHK_DEB_1       ;YES THEN DEC TIMER
0272 07B0           0677         BTFSS   FLAG,KEY_BEEP   ;KEY BEEP SET?
0273 0A7F           0678         GOTO    CHK_SERV        ;NO, SEE IF SERVICED
0274 0678           0679         BTFSC   ALFLAG,INAA     ;IN AA?
0275 0A86           0680         GOTO    CHK_BEP_ON      ;YES THEN SEE IF ON
                    0681 CHK_DEB_1
0276 05B7           0682         BSF     ENTFLG,INKEYBEP ;SET FLAG
0277 0215           0683         MOVF    DEBOUNCE,W      ;GET IN W
0278 0643           0684         BTFSC   STATUS,Z        ;NZ THEN SKIP
0279 0C14           0685         MOVLW   D'20'           ;ELSE DB 100 mSEC
027A 0035           0686         MOVWF   DEBOUNCE        ;       /
027B 0405           0687         BCF     PORT_A,BEP      ;TURN ON BEEPER
027C 02F5           0688         DECFSZ  DEBOUNCE        ;DEC AND CHK
027D 0A08           0689         GOTO    UP_TMR_1        ;GO BACK
```

```
027E 0505           0690         BSF     PORT_A,BEP      ;TURN OFF BEEPER
                    0691 CHK_SERV
                    0692 ;        CLRF    DEBOUNCE
                    0693 ;        BSF     PORT_A,BEP
027F 07F0           0694         BTFSS   FLAG,SERVICED   ;SERVICED THEN SKIP
0280 0A08           0695         GOTO    UP_TMR_1        ;GO BACK
0281 04F0           0696         BCF     FLAG,SERVICED   ;ELSE CLEAR FLAGS
0282 04D0           0697         BCF     FLAG,KEY_HIT    ;       /
0283 04B0           0698         BCF     FLAG,KEY_BEEP   ;RESET FLAG
0284 04B7           0699         BCF     ENTFLG,INKEYBEP ;    /
0285 0A08           0700         GOTO    UP_TMR_1        ;GO BACK
                    0701 ;
                    0702 CHK_BEP_ON
0286 0705           0703         BTFSS   PORT_A,BEP      ;IF OFF THEN SKIP
0287 0A08           0704         GOTO    UP_TMR_1        ;ELSE WAIT
0288 0A76           0705         GOTO    CHK_DEB_1       ;RETURN
                    0706 ;
                    0707 ;
                    0708 INC_HR
0289 02A0           0709         INCF    F0              ;INC HOUR TIMER
028A 0200           0710         MOVF    F0,W            ;GET HR TMR IN W
028B 0031           0711         MOVWF   TEMP            ;SAVE IN TEMP
028C 0E0F           0712         ANDLW   B'00001111'     ;CHK LO BYTE = 10
028D 0F0A           0713         XORLW   D'10'           ;       /
028E 0743           0714         BTFSS   STATUS,Z        ;YES THEN SKIP
028F 0A93           0715         GOTO    INC_AM_PM       ;ELSE CHK 12
0290 0C10           0716         MOVLW   B'00010000'     ;LOAD 1 IN MSB
0291 0020           0717         MOVWF   F0
0292 0AA3           0718         GOTO    RESTORE_AM_PM   ;RESTORE AM/PM
                    0719 INC_AM_PM
0293 04E0           0720         BCF     F0,AM_PM        ;CLEAR AM/PM
0294 0200           0721         MOVF    F0,W            ;GET IN W
0295 0F12           0722         XORLW   12H             ;SEE IF 12 HEX
0296 0743           0723         BTFSS   STATUS,Z        ;YES THEN SKIP
0297 0A9D           0724         GOTO    CHK_13          ;ELSE CHK 13
0298 07F1           0725         BTFSS   TEMP,AM_PM      ;IF SET, SKIP
0299 0A9C           0726         GOTO    SET_AM_PM       ;ELSE SET
029A 04E0           0727         BCF     F0,AM_PM        ;CLEAR FLAG
029B 0800           0728         RETLW   0               ;RETURN
                    0729 SET_AM_PM
029C 05E0           0730         BSF     F0,AM_PM        ;SET FLAG
                    0731 CHK_13
029D 0200           0732         MOVF    F0,W            ;GET IN W
029E 0F13           0733         XORLW   13H             ;SEE IF 13
029F 0743           0734         BTFSS   STATUS,Z        ;YES THEN SKIP
02A0 0AA3           0735         GOTO    RESTORE_AM_PM
                    0736 SET_1_HR
02A1 0C01           0737         MOVLW   B'00000001'     ;SET TO 1
02A2 0020           0738         MOVWF   F0
                    0739 RESTORE_AM_PM
02A3 06F1           0740         BTFSC   TEMP,AM_PM      ;SKIP IF AM
02A4 05E0           0741         BSF     F0,AM_PM        ;ELSE SET TO PM
02A5 0800           0742         RETLW   0
                    0743 ;
                    0744 ;
                    0745 ;
                    0747         ORG     400
                    0748 ;
                    0749 ;KEY DEFINITIONS
000A                0750         ALARM_KEY       EQU     0A
000B                0751         CE_KEY          EQU     0B
000C                0752         SNOOZE_KEY      EQU     0C
000D                0753         AM_PM_KEY       EQU     0D
000E                0754         CLR_ALARM_KEY   EQU     0E
000F                0755         SET_KEY         EQU     0F
                    0756 ;
                    0757 SERVICE_KEYS
0400 07D0           0758         BTFSS   FLAG,KEY_HIT    ;NO KEY HIT THEN ...
0401 0800           0759         RETLW   0               ;RETURN
```

```
0402 06F0           0760         BTFSC   FLAG,SERVICED   ;IF NOT SERVICED SKIP
0403 0800           0761         RETLW   0               ;ELSE RETURN
0404 05F0           0762         BSF     FLAG,SERVICED   ;SET SERVICED FLAG
0405 0210           0763         MOVF    FLAG,W          ;GET MODE OF OPERATION
0406 0E03           0764         ANDLW   B'00000011'     ;       /
0407 0643           0765         BTFSC   STATUS,Z        ;00 THEN RTM
0408 0A10           0766         GOTO    RTMKS           ;RTM KEY SERVICE
0409 0031           0767         MOVWF   TEMP            ;SAVE IN TEMP
040A 02F1           0768         DECFSZ  TEMP            ;REDUCE TEMP
040B 0A0D           0769         GOTO    SK1             ;SKIP
040C 0A1D           0770         GOTO    ATMKS           ;01, DO ALARM MODE
                    0771 SK1
040D 02F1           0772         DECFSZ  TEMP            ;REDUCE TEMP
040E 0800           0773         RETLW   0               ;11 THEN RETURN
040F 0A2A           0774         GOTO    DEMKS           ;10, DATA ENTRY MODE
                    0775 ;
                    0776 ;REAL TIME MODE KEY SERVICE
                    0777 RTMKS
0410 09BA           0778         CALL    CHK_AL_KEYS     ;CHK ALARM KEYS
0411 0D00           0779         IORLW   0               ;SEE IF NZ RET
0412 0643           0780         BTFSC   STATUS,Z        ;NZ THEN SKIP
0413 0800           0781         RETLW   0               ;ELSE RETURN
0414 0C0F           0782         MOVLW   SET_KEY         ;SEE IF SET KEY
0415 0193           0783         XORWF   NEW_KEY,W       ;       /
0416 0643           0784         BTFSC   STATUS,Z        ;NO THEN SKIP
0417 0A91           0785         GOTO    SERV_SET_RTM    ;SERVICE SET KEY
0418 0C0A           0786         MOVLW   ALARM_KEY       ;ALARM KEY?
0419 0193           0787         XORWF   NEW_KEY,W       ;    /
041A 0643           0788         BTFSC   STATUS,Z        ;NO THEN SKIP
041B 0AAB           0789         GOTO    SERV_ALARM_RTM  ;ELSE SERVICE ALARM
                    0790 IGNORE_KEY
041C 0800           0791         RETLW   0               ;ELSE RETURN
                    0792 ;
                    0793 ;ALARM TIME MODE KEY SERVICE
                    0794 ATMKS
041D 09BA           0795         CALL    CHK_AL_KEYS     ;CHECK ALRM KEYS
041E 0D00           0796         IORLW   0               ;CHECK IF 0
041F 0643           0797         BTFSC   STATUS,Z        ;NZ THEN SKIP
0420 0800           0798         RETLW   0               ;ELSE RETURN
0421 0C0F           0799         MOVLW   SET_KEY         ;SEE IF SET KEY
0422 0193           0800         XORWF   NEW_KEY,W       ;       /
0423 0643           0801         BTFSC   STATUS,Z        ;NO THEN SKIP
0424 0A9C           0802         GOTO    SERV_SET_ATM
0425 0C0A           0803         MOVLW   ALARM_KEY       ;GET ALARM KEY
0426 0193           0804         XORWF   NEW_KEY,W       ;SEE IF HIT
0427 0643           0805         BTFSC   STATUS,Z        ;NO THEN SKIP
0428 0AA2           0806         GOTO    SERV_ALARM_ATM  ;ELSE SERVICE
0429 0A1C           0807         GOTO    IGNORE_KEY
                    0808 ;
                    0809 ;DATA ENTRY MODE KEY SERVICE
                    0810 DEMKS
042A 09BA           0811         CALL    CHK_AL_KEYS     ;CHECK ALARM KEYS
042B 0D00           0812         IORLW   0               ;CHK IF 0
042C 0643           0813         BTFSC   STATUS,Z        ;NZ THEN SKIP
042D 0800           0814         RETLW   0               ;ELSE RETURN
042E 0C0F           0815         MOVLW   SET_KEY         ;IF SET KEY THEN END
042F 0193           0816         XORWF   NEW_KEY,W       ;       /
0430 0643           0817         BTFSC   STATUS,Z        ;NO THEN SKIP
0431 0A3F           0818         GOTO    DEMKS_END       ;GOTO END
0432 0C0B           0819         MOVLW   CE_KEY          ;IF CLEAR ENTRY
0433 0193           0820         XORWF   NEW_KEY,W       ;       /
0434 0643           0821         BTFSC   STATUS,Z        ;SKIP IF NO
0435 0A48           0822         GOTO    DEMKS_END_1     ;ABANDON ENTRY
0436 0737           0823         BTFSS   ENTFLG,HR10     ;10'S HRS DONE?
0437 0A54           0824         GOTO    ENT_HR_10       ;NO THEN GET
0438 0757           0825         BTFSS   ENTFLG,HR       ;HRS DONE?
0439 0A5F           0826         GOTO    ENT_HRS         ;NO THEN GET
043A 0777           0827         BTFSS   ENTFLG,MIN10    ;10'S MIN. DONE?
043B 0A72           0828         GOTO    ENT_MIN_10      ;NO THEN GET
```

```
043C 0797           0829         BTFSS   ENTFLG,MIN      ;MIN DONE?
043D 0A7F           0830         GOTO    ENT_MIN         ;NO THEN GET
043E 0A87           0831         GOTO    ENT_AM_PM       ;NO THEN GET
                    0832 DEMKS_END
043F 0717           0833         BTFSS   ENTFLG,RTATS    ;GET OLD STATUS
0440 0A4D           0834         GOTO    LD_RTM          ;LOAD IN TIME
0441 020E           0835         MOVF    MENTRY,W        ;LD IN ALARM
0442 002C           0836         MOVWF   MALARM          ;       /
0443 020F           0837         MOVF    HENTRY,W        ;     /
0444 002D           0838         MOVWF   HALARM          ;   /
0445 0450           0839         BCF     FLAG,ALRMLED    ;CLEAR FLAG
0446 0618           0840         BTFSC   ALFLAG,ALONOF   ;SEE IF ON-OFF
0447 0550           0841         BSF     FLAG,ALRMLED    ;ELSE SET
                    0842 DEMKS_END_1
0448 0410           0843         BCF     FLAG,0          ;RTM MODE
0449 0430           0844         BCF     FLAG,1          ;    /
044A 0490           0845         BCF     FLAG,FLASH      ;STOP FLASH
                    0846 SERV_COM_RET
044B 05B0           0847         BSF     FLAG,KEY_BEEP
044C 0800           0848         RETLW   0               ;RETURN
                    0849 ;
                    0850 LD_RTM
044D 020E           0851         MOVF    MENTRY,W        ;LD IN RTM
044E 002A           0852         MOVWF   MTMR            ;       /
044F 020F           0853         MOVF    HENTRY,W        ;     /
0450 002B           0854         MOVWF   HTMR            ;   /
0451 0068           0855         CLRF    MSTMR           ;CLR TIME
0452 0069           0856         CLRF    STMR            ;  /
0453 0A48           0857         GOTO    DEMKS_END_1     ;GO BACK
                    0858 ;
                    0859 ENT_HR_10
0454 0213           0860         MOVF    NEW_KEY,W       ;SEE IF 0
0455 0643           0861         BTFSC   STATUS,Z        ;NZ THEN SKIP
0456 0A5C           0862         GOTO    LD_HENTRY_0     ;LOAD 0
0457 02D3           0863         DECFSZ  NEW_KEY,0       ;1 THE SKIP
0458 0A1C           0864         GOTO    IGNORE_KEY      ;ELSE IGNORE KEY
0459 058F           0865         BSF     HENTRY,4        ;SET TO 1
045A 0537           0866         BSF     ENTFLG,HR10     ;SET FLAG
045B 0A4B           0867         GOTO    SERV_COM_RET    ;GO GET NEXT
                    0868 LD_HENTRY_0
045C 048F           0869         BCF     HENTRY,4        ;SET TO 0
045D 0537           0870         BSF     ENTFLG,HR10
045E 0A4B           0871         GOTO    SERV_COM_RET    ;
                    0872 ENT_HRS
045F 0C0F           0873         MOVLW   HENTRY          ;USE INDIRECT ADDR.
0460 0024           0874         MOVWF   FSR             ;       /
0461 068F           0875         BTFSC   HENTRY,4        ;SEE IF 0
0462 0A6D           0876         GOTO    ALLOW0_2        ;YES THEN 0,1&2
0463 0C0A           0877         MOVLW   D'10'           ;SEE IF 0 - 9
0464 0093           0878         SUBWF   NEW_KEY,W       ;       /
0465 0603           0879         BTFSC   STATUS,C        ;IF C THEN SKIP
0466 0A1C           0880         GOTO    IGNORE_KEY      ;ELSE IGNORE
                    0881 ENT_LO_COM1
0467 0557           0882         BSF     ENTFLG,HR       ;SET FLAG
                    0883 ENT_LO_COM
0468 0200           0884         MOVF    F0,W            ;LD HRS
0469 0EF0           0885         ANDLW   B'11110000'     ;MASK LO NIBL
046A 0113           0886         IORWF   NEW_KEY,W       ;OR NEW KEY
046B 0020           0887         MOVWF   F0              ;SAVE BACK
046C 0A4B           0888         GOTO    SERV_COM_RET    ;GET NEXT
                    0889 ALLOW0_2
046D 0C03           0890         MOVLW   D'3'            ;SEE IF 0 - 2
046E 0093           0891         SUBWF   NEW_KEY,W       ;       /
046F 0603           0892         BTFSC   STATUS,C        ;<3 THEN SKIP
0470 0A1C           0893         GOTO    IGNORE_KEY
0471 0A67           0894         GOTO    ENT_LO_COM1     ;       /
                    0895 ;
                    0896 ENT_MIN_10
0472 0C0E           0897         MOVLW   MENTRY          ;DO INDIRECT ADDR.
```

```
0473 0024           0898         MOVWF   FSR             ;       /
0474 0C06           0899         MOVLW   D'6'            ;ALLOW 0 - 5
0475 0093           0900         SUBWF   NEW_KEY,W       ;       /
0476 0603           0901         BTFSC   STATUS,C        ;IF C THEN SKIP
0477 0A1C           0902         GOTO    IGNORE_KEY      ;ELSE IGNORE
0478 0380           0903         SWAPF   F0,W            ;SWAP AND GET
0479 0EF0           0904         ANDLW   B'11110000'     ;MASK LO NIBL
047A 0113           0905         IORWF   NEW_KEY,W       ;OR NEW KEY
047B 0020           0906         MOVWF   F0              ;SAVE BACK
047C 03A0           0907         SWAPF   F0              ;SWAP BACK
047D 0577           0908         BSF     ENTFLG,MIN10
047E 0A4B           0909         GOTO    SERV_COM_RET    ;GET NEXT
                    0910 ;
                    0911 ENT_MIN
047F 0C0E           0912         MOVLW   MENTRY          ;DO INDIRECT
0480 0024           0913         MOVWF   FSR             ;       /
0481 0C0A           0914         MOVLW   D'10'           ;ALLOW 0 - 9
0482 0093           0915         SUBWF   NEW_KEY,W       ;SEE IF >
0483 0603           0916         BTFSC   STATUS,C        ;NO THEN SKIP
0484 0A1C           0917         GOTO    IGNORE_KEY      ;ELSE IGNORE
0485 0597           0918         BSF     ENTFLG,MIN      ;SET FLAG
0486 0A68           0919         GOTO    ENT_LO_COM      ;       /
                    0920 ;
                    0921 ENT_AM_PM
0487 0C0D           0922         MOVLW   AM_PM_KEY       ;AM/PM KEY?
0488 0193           0923         XORWF   NEW_KEY,W       ;    /
0489 0743           0924         BTFSS   STATUS,Z        ;YES THEN SKIP
048A 0A1C           0925         GOTO    IGNORE_KEY
048B 07EF           0926         BTFSS   HENTRY,AM_PM    ;TEST BIT
048C 0A8F           0927         GOTO    SETAMPM         ;ELSE SET
048D 04EF           0928         BCF     HENTRY,AM_PM    ;CLEAR FLAG
048E 0A4B           0929         GOTO    SERV_COM_RET    ;GOTO END
                    0930 SETAMPM
048F 05EF           0931         BSF     HENTRY,AM_PM    ;SET FLAG
0490 0A4B           0932         GOTO    SERV_COM_RET
                    0933 ;
                    0934 ;
                    0935 ;
                    0936 SERV_SET_RTM
0491 020A           0937         MOVF    MTMR,W          ;TRANSFER TIME
0492 002E           0938         MOVWF   MENTRY          ;TO DATA ENTRY
0493 020B           0939         MOVF    HTMR,W          ;       /
0494 002F           0940         MOVWF   HENTRY          ;     /
                    0941 SERV_COM
0495 0210           0942         MOVF    FLAG,W          ;SAVE IN W
0496 0E01           0943         ANDLW   B'00000001'     ;ATM OR RTM MODE?
0497 0037           0944         MOVWF   ENTFLG          ;SAVE IN ENTFLG
0498 0CF2           0945         MOVLW   B'11110010'     ;FORCE 1S
0499 0130           0946         IORWF   FLAG            ;    /
049A 0410           0947         BCF     FLAG,0          ;   /
049B 0800           0948         RETLW   0
                    0949 ;
                    0950 SERV_SET_ATM
049C 020C           0951         MOVF    MALARM,W        ;TRANSFER ALARM
049D 002E           0952         MOVWF   MENTRY          ;TO DATA ENTRY
049E 020D           0953         MOVF    HALARM,W        ;       /
049F 002F           0954         MOVWF   HENTRY          ;     /
04A0 0518           0955         BSF     ALFLAG,ALONOF   ;SET FLAG
04A1 0A95           0956         GOTO    SERV_COM        ;GOTO COMMON
                    0957 ;
                    0958 SERV_ALARM_ATM
04A2 0718           0959         BTFSS   ALFLAG,ALONOF   ;TEST ON/OFF
04A3 0AA6           0960         GOTO    SET_ALONOF      ;SET ON/OF FLG
04A4 0418           0961         BCF     ALFLAG,ALONOF   ;CLEAR FLAG
04A5 0AA7           0962         GOTO    SERV_ATM_COM    ;RET THRO COM
                    0963 SET_ALONOF
04A6 0518           0964         BSF     ALFLAG,ALONOF   ;SET FLAG
                    0965 SERV_ATM_COM
04A7 05B0           0966         BSF     FLAG,KEY_BEEP   ;BEEP
```

```
04A8 0CF0           0967         MOVLW   B'11110000'     ;CLEAR SEC COUNT
04A9 0176           0968         ANDWF   MIN_SEC         ;       /
04AA 0800           0969         RETLW   0               ;RETURN
                    0970 ;
                    0971 SERV_ALARM_RTM
04AB 05B0           0972         BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04AC 0510           0973         BSF     FLAG,0          ;SET TO ALARM TIME
04AD 0430           0974         BCF     FLAG,1          ;       /
04AE 0C05           0975         MOVLW   D'05'           ;SAVE 5 IN MIN_SEC
04AF 0036           0976         MOVWF   MIN_SEC         ;       /
04B0 0800           0977         RETLW   0
                    0978 ;
                    0979 SERV_SNOOZE
04B1 0CA0           0980         MOVLW   0A0             ;SNOOZE FOR 10 MINS
04B2 0036           0981         MOVWF   MIN_SEC         ;       /
04B3 0558           0982         BSF     ALFLAG,SILNC    ;SET FLAG
                    0983 CLR_AL_COM
04B4 05B0           0984         BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04B5 007A           0985         CLRF    AATMR           ;RESET AA TIMER
04B6 0079           0986         CLRF    AAFLAG          ;CLEAR AA FLAGS
04B7 0478           0987         BCF     ALFLAG,INAA     ;RESET INAA FLAG
04B8 0505           0988         BSF     PORT_A,BEP      ;TURN OFF BEEPER
04B9 0800           0989         RETLW   0               ;RET
                    0990 ;
                    0991 CHK_AL_KEYS
04BA 0718           0992         BTFSS   ALFLAG,ALONOF   ;ALARM ON?
04BB 0801           0993         RETLW   1               ;NO THEN RET
04BC 0738           0994         BTFSS   ALFLAG,INAL     ;IN ALARM?
04BD 0801           0995         RETLW   1               ;NO THEN SKIP
04BE 0C0E           0996         MOVLW   CLR_ALARM_KEY   ;CHECK IF CLR ALARM
04BF 0193           0997         XORWF   NEW_KEY,W       ;       /
04C0 0643           0998         BTFSC   STATUS,Z        ;NO THEN SKIP
04C1 0AC7           0999         GOTO    CLR_ALARM       ;ELSE CLEAR ALARM
04C2 0C0C           1000         MOVLW   SNOOZE_KEY      ;SEE IF SNOOZE HIT
04C3 0193           1001         XORWF   NEW_KEY,W       ;       /
04C4 0743           1002         BTFSS   STATUS,Z        ;YES THEN SKIP
04C5 0801           1003         RETLW   1
04C6 0AB1           1004         GOTO    SERV_SNOOZE
                    1005 ;
                    1006 CLR_ALARM
04C7 0438           1007         BCF     ALFLAG,INAL     ;CLEAR ALARM
04C8 0458           1008         BCF     ALFLAG,SILNC    ;CLEAR SILENCE
04C9 0C0F           1009         MOVLW   B'00001111'     ;CLEAR MINS
04CA 0176           1010         ANDWF   MIN_SEC         ;       /
04CB 0AB4           1011         GOTO    CLR_AL_COM
                    1012 ;
                    1013         ORG     600
                    1014 ;If the AA alarm is set, then this routine takes care of
                    1015 ;the timing in sounding the alarm.
                    1016 ;
                    1017 SOUND_AA
0600 0738           1018         BTFSS   ALFLAG,INAL     ;SKIP IF IN ALRM
0601 0800           1019         RETLW   0               ;ELSE RETURN
0602 0658           1020         BTFSC   ALFLAG,SILNC    ;SKIP IF NOT IN SIL
0603 0800           1021         RETLW   0               ;ELSE RET
0604 06B7           1022         BTFSC   ENTFLG,INKEYBEP ;SKIP IF NOT IN KEY BEP
0605 0A55           1023         GOTO    CHK_COLSN       ;CHK COLLISION
                    1024 SND_AA_0
0606 0778           1025         BTFSS   ALFLAG,INAA     ;SKIP IF IN AA
                    1026 SND_AA_1
0607 0919           1027         CALL    INIT_AA         ;INIT ALL
0608 0719           1028         BTFSS   AAFLAG,0        ;SKIP IF DONE
0609 0A21           1029         GOTO    DO_CYCL0        ;DO FIRST CYCL
060A 0739           1030         BTFSS   AAFLAG,1        ;SKIP IF DONE
060B 0A29           1031         GOTO    DO_CYCL1        ;ELSE 2ND CYCLE
060C 0759           1032         BTFSS   AAFLAG,2        ;SKIP IF DONE
060D 0A31           1033         GOTO    DO_CYCL2        ;ELSE DO 3RD CYCLE
060E 0779           1034         BTFSS   AAFLAG,3        ;SKIP IF DONE
060F 0A39           1035         GOTO    DO_CYCL3        ;DO CYCLE 4
```

```
0610 0799           1036         BTFSS   AAFLAG,4        ;SKIP IF DONE
0611 0A3E           1037         GOTO    DO_CYCL4        ;DO CYCLE 5
0612 07B9           1038         BTFSS   AAFLAG,5        ;SKIP IF DONE
0613 0A43           1039         GOTO    DO_CYCL5        ;DO CYCLE 6
0614 07D9           1040         BTFSS   AAFLAG,6        ;SKIP IF DONE
0615 0A48           1041         GOTO    DO_CYCL6        ;DO CYCLE 6
0616 07F9           1042         BTFSS   AAFLAG,7        ;SKIP IF DONE
0617 0A50           1043         GOTO    DO_CYCL7        ;DO CYCLE 7
0618 0A07           1044         GOTO    SND_AA_1        ;GO BACK
                    1045 ;
                    1046 INIT_AA
0619 0079           1047         CLRF    AAFLAG          ;CLEAR ALL FLAGS
061A 0578           1048         BSF     ALFLAG,INAA     ;SET IN AA FLAG
061B 0A2D           1049         GOTO    PUT_ON_100      ;ON 100 MSECS
                    1050 ;
                    1051 DEC_AA_TMR
061C 00FA           1052         DECF    AATMR           ;REDUCE TIMER
061D 021A           1053         MOVF    AATMR,W         ;GET IN W
061E 0743           1054         BTFSS   STATUS,Z        ;CHECK IF Z
061F 0801           1055         RETLW   1               ;NO THEN NZ
0620 0800           1056         RETLW   0               ;ELSE 0
                    1057 ;
                    1058 DO_CYCL0
0621 091C           1059         CALL    DEC_AA_TMR      ;REDUCE TIMER
0622 0743           1060         BTFSS   STATUS,Z        ;IF NZ THEN RET
0623 0800           1061         RETLW   0
0624 0519           1062         BSF     AAFLAG,0        ;SET DONE FLAG
                    1063 PUT_OFF_100
0625 0505           1064         BSF     PORT_A,BEP      ;TURN OFF BEEPER
0626 0C14           1065         MOVLW   D'20'           ;FOR 100 MSECS
0627 003A           1066         MOVWF   AATMR           ;       /
0628 0800           1067         RETLW   0
                    1068 ;
                    1069 DO_CYCL1
0629 091C           1070         CALL    DEC_AA_TMR      ;REDUCE TIMER
062A 0743           1071         BTFSS   STATUS,Z        ;IF NZ THEN RET
062B 0800           1072         RETLW   0
062C 0539           1073         BSF     AAFLAG,1        ;SET DONE FLAG
                    1074 PUT_ON_100
062D 0405           1075         BCF     PORT_A,BEP      ;TURN ON BEEPER
062E 0C14           1076         MOVLW   D'20'           ;FOR 100 MSECS
062F 003A           1077         MOVWF   AATMR           ;       /
0630 0800           1078         RETLW   0
                    1079 ;
                    1080 DO_CYCL2
0631 091C           1081         CALL    DEC_AA_TMR      ;REDUCE TIMER
0632 0743           1082         BTFSS   STATUS,Z        ;IF NZ THEN RET
0633 0800           1083         RETLW   0               ;       /
0634 0559           1084         BSF     AAFLAG,2        ;SET DONE FLAG
0635 0505           1085         BSF     PORT_A,BEP      ;TURN OFF BEEPER
0636 0C64           1086         MOVLW   D'100'          ;FOR 500 MSECS
0637 003A           1087         MOVWF   AATMR           ;       /
0638 0800           1088         RETLW   0
                    1089 ;
                    1090 DO_CYCL3
0639 091C           1091         CALL    DEC_AA_TMR      ;REDUCE TIMER
063A 0743           1092         BTFSS   STATUS,Z        ;IF NZ THEN RET
063B 0800           1093         RETLW   0               ;       /
063C 0579           1094         BSF     AAFLAG,3        ;SET DONE FLAG
063D 0A2D           1095         GOTO    PUT_ON_100      ;DO NEXT CYCLE
                    1096 ;
                    1097 DO_CYCL4
063E 091C           1098         CALL    DEC_AA_TMR      ;REDUCE TIMER
063F 0743           1099         BTFSS   STATUS,Z        ;IF NZ THEN RET
0640 0800           1100         RETLW   0               ;       /
0641 0599           1101         BSF     AAFLAG,4        ;SET DONE FLAG
0642 0A25           1102         GOTO    PUT_OFF_100     ;DO NEXT CYCLE
                    1103 ;
                    1104 DO_CYCL5
```

```
0643 091C           1105         CALL    DEC_AA_TMR      ;REDUCE TIMER
0644 0743           1106         BTFSS   STATUS,Z        ;IF NZ THEN RET
0645 0800           1107         RETLW   0               ;       /
0646 05B9           1108         BSF     AAFLAG,5        ;SET DONE FLAG
0647 0A2D           1109         GOTO    PUT_ON_100      ;DO NEXT CYCLE
                    1110 ;
                    1111 DO_CYCL6
0648 091C           1112         CALL    DEC_AA_TMR      ;REDUCE TIMER
0649 0743           1113         BTFSS   STATUS,Z        ;IF NZ THEN RET
064A 0800           1114         RETLW   0               ;       /
064B 05D9           1115         BSF     AAFLAG,6        ;SET DONE FLAG
064C 0505           1116         BSF     PORT_A,BEP      ;TURN OFF BEEPER
064D 0CC8           1117         MOVLW   D'200'          ;FOR 1000 MSECS
064E 003A           1118         MOVWF   AATMR           ;       /
064F 0800           1119         RETLW   0
                    1120 ;
                    1121 DO_CYCL7
0650 091C           1122         CALL    DEC_AA_TMR      ;REDUCE TIMER
0651 0743           1123         BTFSS   STATUS,Z        ;IF NZ THEN RET
0652 0800           1124         RETLW   0               ;       /
0653 05F9           1125         BSF     AAFLAG,7        ;SET DONE FLAG
0654 0A2D           1126         GOTO    PUT_ON_100      ;DO NEXT CYCLE
                    1127 ;
                    1128 CHK_COLSN
0655 0605           1129         BTFSC   PORT_A,BEP      ;IF ON THEN SKIP
0656 0A06           1130         GOTO    SND_AA_0        ;ELSE RET
0657 021A           1131         MOVF    AATMR,W         ;GET TIMER
0658 0643           1132         BTFSC   STATUS,Z        ;NZ THEN SKIP
0659 0A5C           1133         GOTO    LD_AAT_1        ;LOAD A 1 IN TMR
065A 00FA           1134         DECF    AATMR           ;REDUCE TIMER
065B 0800           1135         RETLW   0               ;RETURN
                    1136 LD_AAT_1
065C 02BA           1137         INCF    AATMR           ;INC TIMER
065D 0800           1138         RETLW   0               ;RET
                    1139 ;
                    1140         ORG     PIC57
                    1141 SYS_RESET
07FF 0A00           1142         GOTO    START
                    1143 ;
                    1144         END
                    1145
                    1146
                    1147
                    1148


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXX XXXXXXXXX--- -------- --------
0140 : -------- -------- -------- --------

0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX------ --------
02C0 : -------- -------- -------- --------

0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
04C0 : XXXXXXXXXXXX---- -------- -------- --------
```

```
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX_ ________ ________

0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ ________X

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

AN563

Using PIC16C5X Microcontrollers as LCD Drivers

INTRODUCTION

This application report describes an LCD controller implementation using a PIC16C55 microcontroller. This technique offers display capabilities for applications that require a small display at a low cost together with the capabilities of the standard PIC16C55 microcontroller. We start by an overview of LCD devices and their theory of operation followed by software implementation issues of the controller. The source code for controlling a multiplexed LCD display is included in Appendix A.

LIQUID CRYSTAL DISPLAYS

The Liquid Crystal Display (LCD) is a thin layer of "Liquid Crystal Material" deposited between two plates of glass. The raw LCD is often referred to as "glass". Electrodes are attached to both sides of the glass. One side is referred to as common or backplane, while the other side is referred to as segment.

An LCD is modelled as a capacitor, with one side connected to the common plane and the other side connected to the segment, as shown in Figure 1. LCDs are sensitive to Root Mean Square Voltage levels. When a V_{RMS} level of zero volts is applied to the LCD, the LCD is practically transparent.

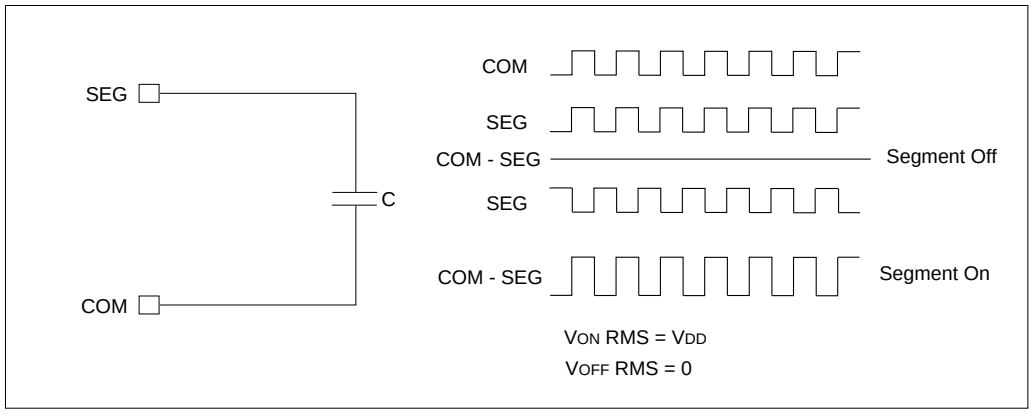
To turn an LCD segment "on", which makes the segment turn dark or opaque, an LCD RMS voltage that is greater than the LCD threshold voltage is applied to the LCD. The RMS LCD voltage is the RMS voltage across the capacitor C in Figure 1, which is equal to the potential difference between SEG and COM values.

Different LCDs have different characteristics; Figure 2 shows typical voltage vs relative contrast characteristics. Notation on curve shows operating points for multiplex operation with the threshold voltage set to 1.7 Vrms. This voltage is often used as the measure of voltage for LCD to be "off" or transparent. The curve is normalized and assumes a viewing angle of 90° to the plane of the LCD.

Contrast control, the process of turning on a segment, is achieved by moving the operating point of the LCD by applying voltage to the LCD that is greater than the LCD threshold voltages. A typical circuit to accomplish this task is shown in Figure 3.

Driving a liquid crystal display at direct current (DC) will cause permanent damage to the display unit. In order to prevent irreversible electrochemical action from destroying the display, the voltage at all segment locations must reverse polarity periodically so that a zero net voltage is applied to the device. This process is referred to as AC voltage application. There are two LCD driving methods available: Static driving method and multiplexed driving method.

FIGURE 1: ELECTRICAL MODEL OF AN LCD SEGMENT WITH DRIVING VOLTAGES



Using PIC16C5X Microcontrollers as LCD Drivers

Conventional LCDs have separate external connections for each and every segment plus a common plane. This is the most basic method that results in good display quality. The main disadvantage of this driving method is that each segment requires one liquid crystal driver. The static driving method uses the frame frequency, defined as a period of the common plane signal, of several tens to several hundred Hz. A lower frequency would result in blinking effects and higher frequencies would increase power requirements. To turn a segment on, a voltage that has an opposite polarity to the common plane signal must be applied resulting in a large RMS voltage across the plates. To turn off a segment a voltage that is of the same polarity to common plane signal is applied. This drive method is universal to driving LCD segments. Figure 1 shows an example of this driving method.

The LCD frequency is defined as the rate of output changes of the common plane and segment signals, whereas the frame rate is defined as $f_{frame} = \frac{f_{frame}}{N}$

where N is the multiplex rate or number of backplane. Typically, f_{frame} ranges from 25HZ to 300HZ. The most commonly used frame fequency is 40-70HZ. A lower frequency would result in flicker effects and higher frequency would increase power requirements.

Multiplexed LCDs maintain their liquid crystal characteristics. These are low power consumption, high contrast ratio under high ambient light levels, and reduce the number of external connections necessary for dot matrix and alphanumeric displays. The multiplex driving method reduces the number of driver circuits, or microcontroller I/O pins if a software method is used. The method of drive for multiplexed displays is Time Division Multiplex (TDM) with the number of time divisions equal to twice the number of common planes used in a given format. In order to prevent permanent damage to the LCD display, the voltage at all segment locations must reverse polarity periodically so that zero net voltage is applied. This is the reason for the doubling in time divisions; each common plane must be alternately driven with a voltage pulse of opposite polarity. The drive frequency should be greater than the flicker rate of 25 Hz. Since increasing the drive frequency significantly above this value increases current demand by the CMOS circuitry, an upper drive frequency level of 60 Hz is recommended by most LCD manufacturers. We have chosen a drive rate of 50 Hz for this application report which results in a frame period of 20 ms. The most commonly available formats are 2x4, 3x3, and 5x7. In this report we use a 2x4 format LCD to display hexadecimal digits.

FIGURE 2: TYPICAL LCD CHARACTERISTICS

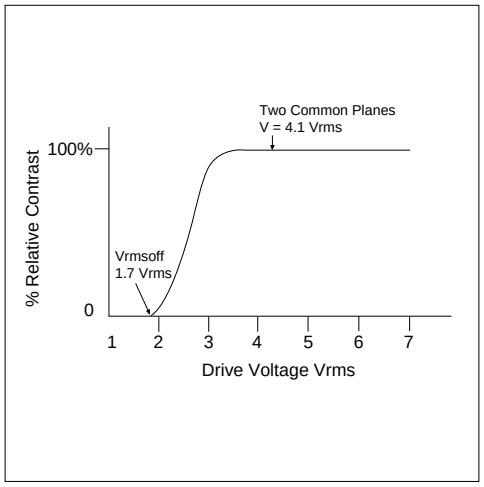
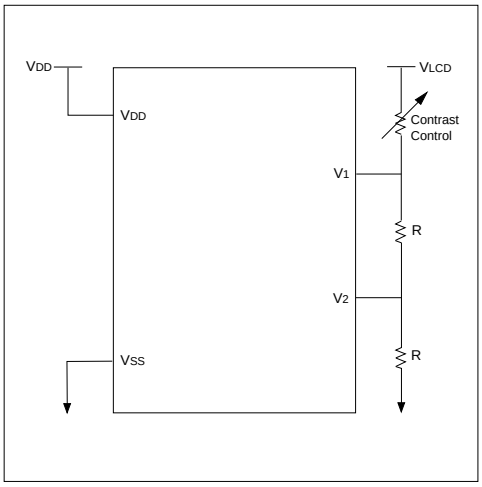


FIGURE 3: CONTRAST CONTROL CIRCUIT



To better understand multiplexed LCD control it is best to look at the general case. The segments in a multiplexed LCD are arranged in an X-Y grid form as shown in Figure 4. The common plane signals maintain their relative shape at all times, as shown in Figure 5. To turn on segment 1 (SEG1), we need to apply a voltage Vd, such that Vs+Vd turns the segment on and Vs-Vd turns the segment off. Note that the segment signal Vd is symmetrical. This is a consequence of the intervals that the common plane signal is not present at all times. Use of nonsymmetrical waveform will result in a higher Vrms present on the unaddressed segments. The symmetrical nature of the waveforms theoretically result in a zero DC voltage levels. CMOS drivers (e.g. microcontrollers) operate at 0 to +5V levels (rail voltage levels). This would require driving voltages beyond the range of operation. This constraint is addressed by a technique referred to as “level shifting” or “biasing”. Level shifting allows application of voltages in the range of 0 to +2.5V, which is compatible with these drivers. This would require an additional voltage level of +2.5V, which can be implemented through a simple resistive voltage divider circuit.

FIGURE 4: MULTIPLEXED LCD SEGMENT ARRANGEMENT

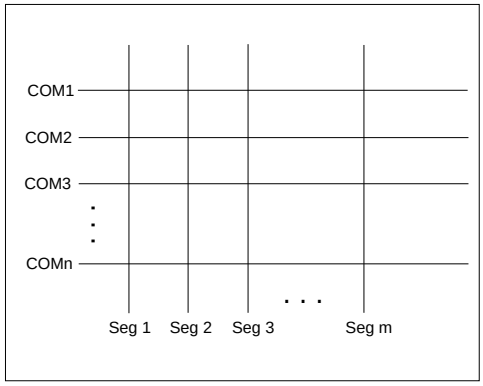
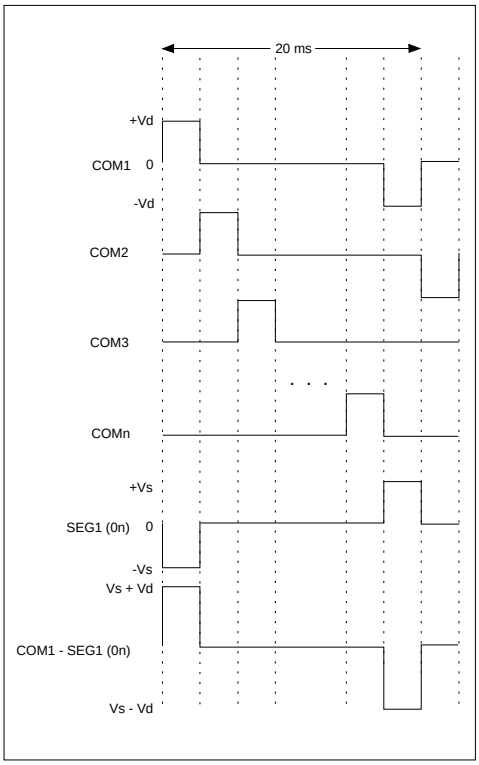


FIGURE 5: MULTIPLEXED LCD DRIVE WAVEFORMS



IMPLEMENTATION

The ideas presented in the previous section can be applied to any size multiplexed LCD display. In our implementation we used a 4-digit LCD from Ocular Inc. [1]. The circuit diagram used in this application report is shown in Figure 6. Each I/O pin on the PIC16C55 device controls the state of two segments (see Figure 6) which requires a total of 16 I/O pins. The reference voltages are generated through a simple resistive voltage divider circuit. The voltage levels are generated by taking advantage of PIC16C5X I/O pin set to input, which tristates the voltage level seen on the pin. This method uses 4 I/O pins to generate the proper voltage levels. Figure 7 shows the truth table for generating the voltage levels. Figure 8 shows how to create a bitmap for different digits. Figure 9 shows the waveforms generated for the accompanying software which implements a hexadecimal counter.

FIGURE 6: SYSTEM CONFIGURATION WITH LCD PINOUT

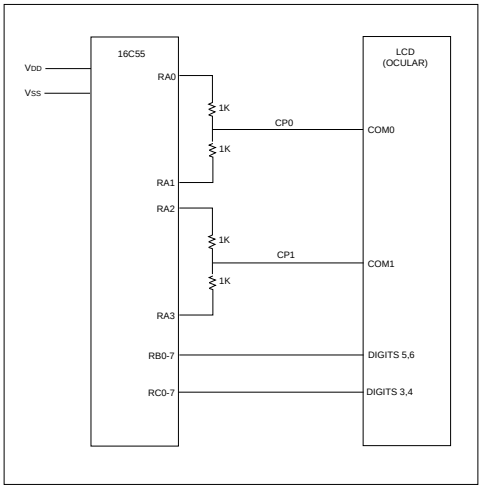


FIGURE 7: COMMON PLANE SIGNAL GENERATION

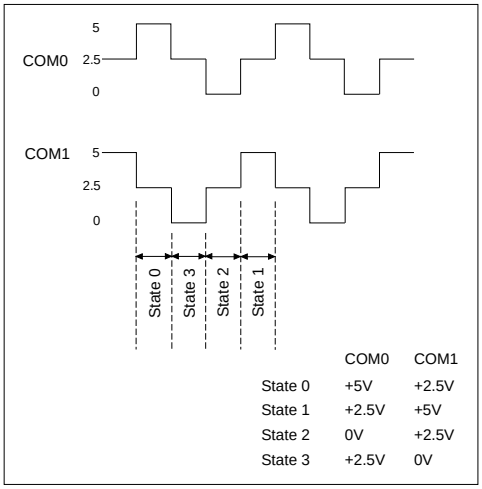
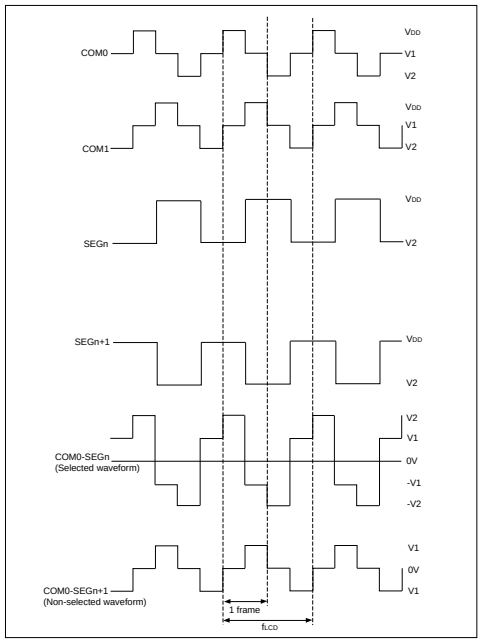


FIGURE 8: LCD CHARACTER BITMAP

Digit	COM0 SEG 0				COM1 SEG 1				COM0 SEG 2				COM1 SEG3			
	F	E	D	DP	A	G	C	B	F	E	D	DP	A	G	C	B
0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
3	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1
4	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1
5	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1
6	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
7	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1
8	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1
9	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
a	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1
b	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
c	1	0	0	1	1	1	0	1	0	1	1	0	0	0	1	0
d	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1
e	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0
f	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

FIGURE 9: EXAMPLE OF OUTPUT WAVEFORMS FOR DIGIT 4



CONCLUSION

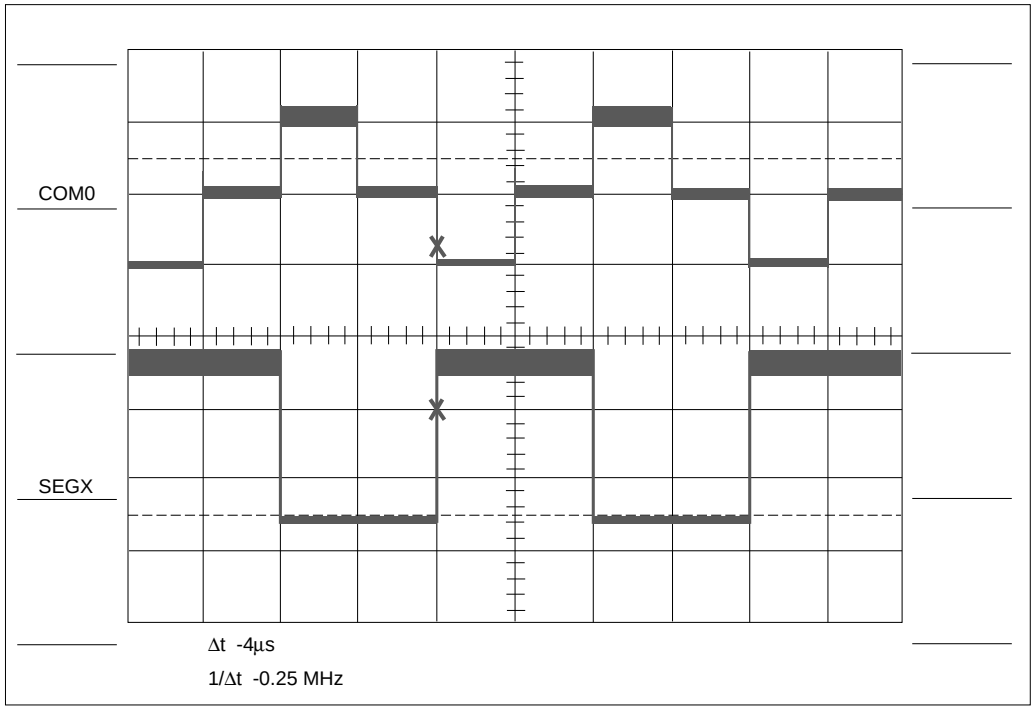
In this application report we have demonstrated the use of PIC16C5X devices to implement a simple LCD controller. As discussed earlier, it is important to keep the generated DC voltage to a minimum to extend the life of the LCD. Ideally one should switch all the I/O lines simultaneously, however, a software implementation of the LCD controller will necessarily introduce a delay which is proportional to the instruction cycle of the microcontroller, as shown in Figure 10. Therefore it is necessary to keep the switching time to a minimum. Our implementation introduced less than 50 mV of DC voltage on the segment lines which is below the manufacturer's recommended DC offset voltage of 60 mV.

REFERENCES

[1] Ocular Inc., Drawing number JH074.

AUTHOR: Al Lovrich
Logic Products Division

FIGURE 10: MICROCONTROLLER GENERATED OUTPUT WAVEFORM



```
MPASM B0.54                                                    PAGE  1


                LIST    C=132,n=0,p=16c55,r=dec

                ;*****************************************************
                ; Project: PIC16C5X as a multiplexed LCD driver.     *
                ;                                                   *
                ; Revision history:                                 *
                ;       04/14/93        original                    *
                ;*****************************************************

                ; Equates

01FF            pic54   equ     0x1ff                       ; Define Reset Vectors
01FF            pic55   equ     0x1ff
03FF            pic56   equ     0x3ff
07FF            pic57   equ     0x7ff

0001            rtcc    equ     1                           ; f1
0002            pc      equ     2                           ; f2
0003            status  equ     3                           ; f3
0004            fsr     equ     4                           ; f4

0005            porta   equ     5                           ; f5
0006            portb   equ     6                           ; f6
0007            portc   equ     7                           ; f8

                ; realtime mode registers

0008            currentState    equ     8
0009            msTimer         equ     currentState+1      ; Millisecond timer
000A            sTimerLow       equ     msTimer+1           ; Lower byte second timer
000B            sTimerHigh      equ     sTimerLow+1         ; Upper byte second timer
000C            digit56         equ     sTimerHigh+1
000D            digit34         equ     digit56+1

                ; Misc definitions

0060            FIVEMSEC        equ     96                  ; Assuming 4.096 MHz crystal

0000            w       equ     0
0001            f       equ     1

0002            z       equ     2

                ; Status register bits

                ;*****************************************************
                ; Port assignments                  *
                ;                                   *
                ;  porta - bit0: Common Plane 0     *
                ;          bit1: Common Plane 0     *
                ;          bit2: Common Plane 1     *
                ;          bit3: Common Plane 1     *
                ;                                   *
                ;  portb - bit0: 6B/DP              *
                ;          bit1: 6C/6D              *
                ;          bit2: 6G/6E              *
                ;          bit3: 6A/6F              *
                ;          bit4: 5B/DP              *
                ;          bit5: 5C/5D              *
                ;          bit6: 5G/5E              *
                ;          bit7: 5A/5F              *
                ;                                   *
                ;  portc - bit0: 4B/DP              *
```

```
                        ;        - bit1: 4C/4D         *
                        ;        - bit2: 4G/4E         *
                        ;        - bit3: 4A/4F         *
                        ;        - bit4: 3B/DP         *
                        ;        - bit5: 3C/3D         *
                        ;        - bit6: 3G/3E         *
                        ;        - bit7: 3A/3F         *
                        ;                      *
                        ;*****************************************************
                        ;
                        ;*****************************************************
                        ;               Macro definitions      *
                        ;*****************************************************
                        UpdateState     macro   State, Table

                                swapf   sTimerLow, w
                                andlw   0xf                     ; Isolate digit 5 (offset)
                                call    Table
                                movwf   digit56
                                swapf   digit56, f

                                movf    sTimerLow, w
                                andlw    0xf                    ; Isolate digit 6 (offset)
                                call    Table
                                iorwf   digit56, f

                                swapf   sTimerHigh, w
                                andlw   0xf                     ; Isolate digit 5 (offset)
                                call    Table
                                movwf   digit34
                                swapf   digit34, f

                                movf    sTimerHigh, w
                                andlw   0xf                     ; Isolate digit 6 (offset)
                                call    Table
                                iorwf   digit34, f

                                movf    digit34, w              ; Display digits 3 & 4
                                movwf   portc
                                movf    digit56, w
                                movwf   portb                   ; Display digits 5 & 6
                                endm

                                org     0

                        ; Initialize ports A, B, and C and RTCC. In case of output data
                        ; values, set the data latch first, then set the port direction.

                        Initialize
0000 0C01               movlw    00000001b                      ; Set data latch
0001 0025               movwf    porta
0002 0C08               movlw    00001000b                      ; Set I/O direction
0003 0005               tris     porta

0004 0C00               movlw    00000000b                      ; Set levels to low
0005 0026               movwf    portb
0006 0C00               movlw    00000000b                      ; Set as outputs
0007 0006               tris     portb

0008 0C00               movlw    00000000b                      ; Set levels to low
0009 0027               movwf    portc
000A 0C00               movlw    00000000b                      ; Set as outputs
000B 0007               tris     portc

000C 0C04               movlw    0x04                           ; Set prescaler
000D 0002               option

000E 0C60               movlw    FIVEMSEC                       ; rtcc = 5ms
000F 0021               movwf    rtcc
```

```
0010 0C04             movlw   4
0011 0028             movwf   currentState

0012 0C0D             movlw   0xd
0013 0029             movwf   msTimer                     ; Initialize millisecond timer

0014 006A             clrf    sTimerLow                   ; Clear second counter
0015 006B             clrf    sTimerHigh

0016 0800             retlw    0

              ; Check timer register for timing out (rtcc = 0). Remain in the
              ; loop until the timer times out.


              ; Wait for 5ms timer timeout

              Timer_Check
0017 0201             movf    rtcc, w
0018 0743             btfss   status, z
0019 0A17             goto    Timer_Check

001A 0C60             movlw   FIVEMSEC
001B 0021             movwf   rtcc

001C 02E9             decfsz  msTimer, f
001D 0A21             goto    Update_Backplane

001E 03EA             incfsz  sTimerLow                   ; Update second counter
001F 0A21             goto    Update_Backplane
0020 02AB             incf    sTimerHigh, f

              ; RA0 and RA1 are used to control voltage level for common plane 0.
              ; RA2 and RA3 are used to control voltage level for common plane 1.
              ; There are four possible states with different voltage levels as
              ; follows:
              ;
              ; State 0 - cp0 = +5v     ra0=1,  ra1=x
              ;           cp1 = +2.5v   ra2=1,  ra3=0
              ; State 1 - cp0 = +2.5v   ra0=1,  ra1=0
              ;           cp1 = +5v     ra2=1,  ra3=x
              ; State 2 - cp0 =  0v     ra0=0,  ra1=x
              ;           cp1 = +2.5v   ra2=1,  ra3=0
              ; State 3 - cp0 = +2.5v   ra0=1,  ra1=0
              ;           cp1 =  0v     ra2=0,  ra3=x


              Update_Backplane
0021 0004             clrwdt                              ; Reset watchdog timer

0022 00C8             decf    currentState, w             ; Update w register
0023 0E03             andlw   0x03                        ; Use only bit0/1
0024 0028             movwf   currentState                ; Update currentState
0025 01E2             addwf   pc, f

0026 0AAD             goto    State3
0027 0A81             goto    State2
0028 0A55             goto    State1
              ;       goto    State0

              ;             State 0

                      State0
              UpdateState     State0, S0_Table

0029 038A             swapf   sTimerLow, w
002A 0E0F             andlw   0xf                 ; Isolate digit 5 (offset)
002B 0944             call    S0_Table
```

```
002C 002C             movwf   digit56
002D 03AC             swapf   digit56, f

002E 020A             movf    sTimerLow, w
002F 0E0F             andlw   0xf                     ; Isolate digit 6 (offset)
0030 0944             call    S0_Table
0031 012C             iorwf   digit56, f

0032 038B             swapf   sTimerHigh, w
0033 0E0F             andlw   0xf                     ; Isolate digit 5 (offset)
0034 0944             call    S0_Table
0035 002D             movwf   digit34
0036 03AD             swapf   digit34, f

0037 020B             movf    sTimerHigh, w
0038 0E0F             andlw   0xf                     ; Isolate digit 6 (offset)
0039 0944             call    S0_Table
003A 012D             iorwf   digit34, f

003B 020D             movf    digit34, w              ; Display digits 3 & 4
003C 0027             movwf   portc
003D 020C             movf    digit56, w
003E 0026             movwf   portb                   ; Display digits 5 & 6

003F 0C05             movlw   00000101b
0040 0025             movwf   porta
0041 0C02             movlw   00000010b
0042 0005             tris    porta

0043 0800             retlw   0

              S0_Table
0044 01E2             addwf    pc, f                  ; Add offset to pc

0045 0804             retlw   0100b                   ; 0
0046 080C             retlw   1100b                   ; 1
0047 0802             retlw   0010b                   ; 2
0048 0800             retlw   0000b                   ; 3
0049 0808             retlw   1000b                   ; 4
004A 0801             retlw   0001b                   ; 5
004B 080F             retlw   1111b                   ; 6
004C 0804             retlw   0100b                   ; 7
004D 0800             retlw   0000b                   ; 8
004E 0800             retlw   0000b                   ; 9
004F 0800             retlw   0000b                   ; a
0050 0809             retlw   1001b                   ; b
0051 080B             retlw   1011b                   ; c
0052 0808             retlw   1000b                   ; d
0053 0803             retlw   0011b                   ; e
0054 0803             retlw   0011b                   ; f

              ; State 1

              State1
                           UpdateState      State1, S1_Table

0055 038A             swapf   sTimerLow, w
0056 0E0F             andlw   0xf                     ; Isolate digit 5 (offset)
0057 0970             call    S1_Table
0058 002C             movwf   digit56
0059 03AC             swapf   digit56, f

005A 020A             movf    sTimerLow, w
005B 0E0F             andlw   0xf                     ; Isolate digit 6 (offset)
005C 0970             call    S1_Table
005D 012C             iorwf   digit56, f

005E 038B             swapf   sTimerHigh, w
```

```
005F 0E0F             andlw   0xf                   ; Isolate digit 5 (offset)
0060 0970             call    S1_Table
0061 002D             movwf   digit34
0062 03AD             swapf   digit34, f

0063 020B             movf    sTimerHigh, w
0064 0E0F             andlw   0xf                   ; Isolate digit 6 (offset)
0065 0970             call    S1_Table
0066 012D             iorwf   digit34, f

0067 020D             movf    digit34, w            ; Display digits 3 & 4
0068 0027             movwf   portc
0069 020C             movf    digit56, w
006A 0026             movwf   portb                 ; Display digits 5 & 6



006B 0C05             movlw   00000101b
006C 0025             movwf   porta
006D 0C08             movlw   00001000b
006E 0005             tris    porta

006F 0800             retlw   0

              S1_Table
0070 01E2             addwf    pc, f

0071 0801             retlw   0001b                 ; 0
0072 080F             retlw   1111b                 ; 1
0073 0809             retlw   1001b                 ; 2
0074 080D             retlw   1101b                 ; 3
0075 0807             retlw   0111b                 ; 4
0076 0805             retlw   0101b                 ; 5
0077 080F             retlw   1111b                 ; 6
0078 080F             retlw   1111b                 ; 7
0079 0801             retlw   0001b                 ; 8
007A 0805             retlw   0101b                 ; 9
007B 0803             retlw   0011b                 ; a
007C 0801             retlw   0001b                 ; b
007D 0809             retlw   1001b                 ; c
007E 0809             retlw   1001b                 ; d
007F 0801             retlw   0001b                 ; e
0080 0803             retlw   0011b                 ; f

              ; State 2

              State2
              UpdateState     State2, S2_Table

0081 038A             swapf   sTimerLow, w
0082 0E0F             andlw   0xf                   ; Isolate digit 5 (offset)
0083 099C             call    S2_Table
0084 002C             movwf   digit56
0085 03AC             swapf   digit56, f

0086 020A             movf    sTimerLow, w
0087 0E0F             andlw   0xf                   ; Isolate digit 6 (offset)
0088 099C             call    S2_Table
0089 012C             iorwf   digit56, f

008A 038B             swapf   sTimerHigh, w
008B 0E0F             andlw   0xf                   ; Isolate digit 5 (offset)
008C 099C             call    S2_Table
008D 002D             movwf   digit34
008E 03AD             swapf   digit34, f

008F 020B             movf    sTimerHigh, w
0090 0E0F             andlw   0xf                   ; Isolate digit 6 (offset)
0091 099C             call    S2_Table
```

```
0092 012D             iorwf   digit34, f

0093 020D             movf    digit34, w            ; Display digits 3 & 4
0094 0027             movwf    portc
0095 020C             movf    digit56, w
0096 0026             movwf   portb                 ; Display digits 5 & 6


0097 0C04             movlw   00000100b
0098 0025             movwf   porta
0099 0C02             movlw   00000010b
009A 0005             tris    porta

009B 0800             retlw   0

              S2_Table
009C 01E2             addwf   pc, f

009D 080B             retlw   1011b   ; 0
009E 0803             retlw   0011b   ; 1
009F 080D             retlw   1101b   ; 2
00A0 080F             retlw   1111b   ; 3
00A1 0807             retlw   0111b   ; 4
00A2 080E             retlw   1110b   ; 5
00A3 080E             retlw   1110b   ; 6
00A4 080B             retlw   1011b   ; 7
00A5 080F             retlw   1111b   ; 8
00A6 080F             retlw   1111b   ; 9
00A7 080F             retlw   1111b   ; a
00A8 0806             retlw   0110b   ; b
00A9 0804             retlw   0100b   ; c
00AA 0807             retlw   0111b   ; d
00AB 080C             retlw   1100b   ; e
00AC 080C             retlw   1100b   ; f

              ; State 3

              State3
              UpdateState     State3, S3_Table

00AD 038A             swapf   sTimerLow, w
00AE 0E0F             andlw   0xf                   ; Isolate digit 5 (offset)
00AF 09C8             call    S3_Table
00B0 002C             movwf   digit56
00B1 03AC             swapf   digit56, f

00B2 020A             movf    sTimerLow, w
00B3 0E0F             andlw   0xf                   ; Isolate digit 6 (offset)
00B4 09C8             call    S3_Table
00B5 012C             iorwf   digit56, f

00B6 038B             swapf   sTimerHigh, w
00B7 0E0F             andlw   0xf                   ; Isolate digit 5 (offset)
00B8 09C8             call    S3_Table
00B9 002D             movwf   digit34
00BA 03AD             swapf   digit34, f

00BB 020B             movf    sTimerHigh, w
00BC 0E0F             andlw   0xf                   ; Isolate digit 6 (offset)
00BD 09C8             call    S3_Table
00BE 012D             iorwf   digit34, f

00BF 020D             movf    digit34, w            ; Display digits 3 & 4
00C0 0027             movwf   portc
00C1 020C             movf    digit56, w
00C2 0026             movwf   portb          ; Display digits 5 & 6
```

```
00C3 0C01             movlw   00000001b
00C4 0025             movwf   porta
00C5 0C08             movlw   00001000b
00C6 0005             tris    porta

00C7 0800             retlw   0

               S3_Table
00C8 01E2             addwf   pc, f

00C9 080E             retlw   1110b          ; 0
00CA 0800             retlw   0000b          ; 1
00CB 0806             retlw   0110b          ; 2
00CC 0802             retlw   0010b          ; 3
00CD 0808             retlw   1000b          ; 4
00CE 080A             retlw   1010b          ; 5
00CF 080E             retlw   1110b          ; 6
00D0 0800             retlw   0000b          ; 7
00D1 080E             retlw   1110b          ; 8
00D2 080A             retlw   1010b          ; 9
00D3 080C             retlw   1100b          ; a
00D4 080E             retlw   1110b          ; b
00D5 0806             retlw   0110b          ; c
00D6 0806             retlw   0110b          ; d
00D7 080E             retlw   1110b          ; e
00D8 080C             retlw   1100b          ; f

               ; Main code

               Start
00D9 0900             call    Initialize
               Repeat
00DA 0917             call    Timer_Check
00DB 0ADA             goto    Repeat

                      org     pic55

               System_Reset
01FF 0AD9                     goto    Start

                   END

Errors   :    0
Warnings :    0
```

PLD Replacement

INTRODUCTION

The PIC16C5X microcontrollers are ideal for implementing low cost combinational and sequential logic circuits that traditionally have been implemented using either numerous TTL gates or using programmable logic chips such as PLAs or EPLDs.

PIC16C5X is a family of high-performance 8-bit microcontrollers from Microchip Technology. It employs Harvard architecture, i.e has a separate data bus (8-bit) and a program bus (12-bit wide). All instructions are single word and execute in one cycle except for program branches. The instruction cycle time is 200 ns at 20 MHz and faster versions with clock frequency of 20 MHz (instruction cycle = 200 ns) are planned. The PIC16C5X microcontrollers are ideal for PLD-type application because:

* Very low cost. Extremely cost effective to replace TTL gates or expensive EPLD's.
* Fully programmable. PIC16C5X microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
* Available off the shelf from distributors.
* PC board real estate saving can be substantial when replacing multitude of TTL's or several PLD's with PIC16C5X microcontrollers which are packaged in 18 and 28 pin packages (DIP, PLCC, SOIC).
* Substantial power savings can be attained by using PIC16C5X's SLEEP mode. In this mode typical power consumption of PIC16C5X is less than 1uA.
* PIC16C5X's I/O ports are bidirectional and software configurable as input or output. The user can mix and match number of inputs or outputs as long as the total does not exceed 20 (PIC16C55/57).
* PIC16C5X's output pins have large current source/sink capability. They can directly drive LED's.
* The speed and efficiency of the PIC16C5X allows it to perform other control, timing, and compute functions in addition to implementing a PLA function.

IMPLEMENTING A PLA

To implement a generic combinational logic function, we can simply emulate an AND-OR PLA in software. This will require that the logic outputs be described as sum of products. To describe our algorithm, we will use a simple 8-input, 8-bit output PLA with 24 product terms (Figure 1). We will further use the truth table in Figure 2 as the PLA function being implemented. In this example, only four inputs (A3: A0) are used and the other four inputs (A7: A4) are don't care. On the output side, seven output pins (Y6: Y0) are used and Y7 is unused. To implement this PLA, the logic inputs A0,A1,...,A7 can be connected to port RB pins RB0,RB1,...,RB7 respectively. The logic outputs Y0,Y1,...,Y7 will appear on port RC pins RC0,RC1,...,RC7 respectively. Port RB is configured as input and port RC will be configured as output. To evaluate each product term one XOR (exclusive OR) and one AND operation will be required. For example, to determine product term P3 = A3.A2.A1.A0 , the expression to evaluate is:

(A<7:0> .XOR. XXXX0011B) .AND. 00001111B).

The constant with which XOR is done will be referred to as P3_x in our discussion. P3_x = XXXX0011B will ensure that if A<3:0> = 0011B, the least significant 4 bits of the result will be 0000b. The AND constant, referred to here as P3_a (Product term 3, AND constant) basically eliminates the don't care inputs (here A<7:4>) by masking them. Therefore, if the result of the XOR-AND operation is zero then P3 = 1 else P3 = 0. Once the Product terms are evaluated they are stored in four product registers Preg_0 to Preg_3. To determine an output term:

Y0 = P0 + P2 + P3 + P5 + P6 + P7 + P8 + P9 + P10 + P12 + P13 + P14

we need to evaluate the following expression:

(Preg_a .AND. OR_a0) .OR. (Preg_b .AND. OR_b0) .OR. (Preg_c .AND. OR_c0)

In our case the constant values to implement Y0 are as follows:

OR_a0 =	1	1	1	0	1	1	0	1
	P7	P6	P5		P3	P2		P0

OR_b0 = 11010111

OR_c0 = 00000000

For larger number of inputs, outputs or product terms, the evaluation will be more complex but following the same principle. Appendix A shows the assembly code to implement this 8 input X 8 output X 24 Product PLA. This example optimizes speed as well as program memory requirement. Appendix B shows a slightly different implementation (only EVAL_Y MACRO is different) that optimizes program memory usage over speed. Table 1 shows time and resources required to implement different size PLAs.

FIGURE 1 - A SIMPLE PLA

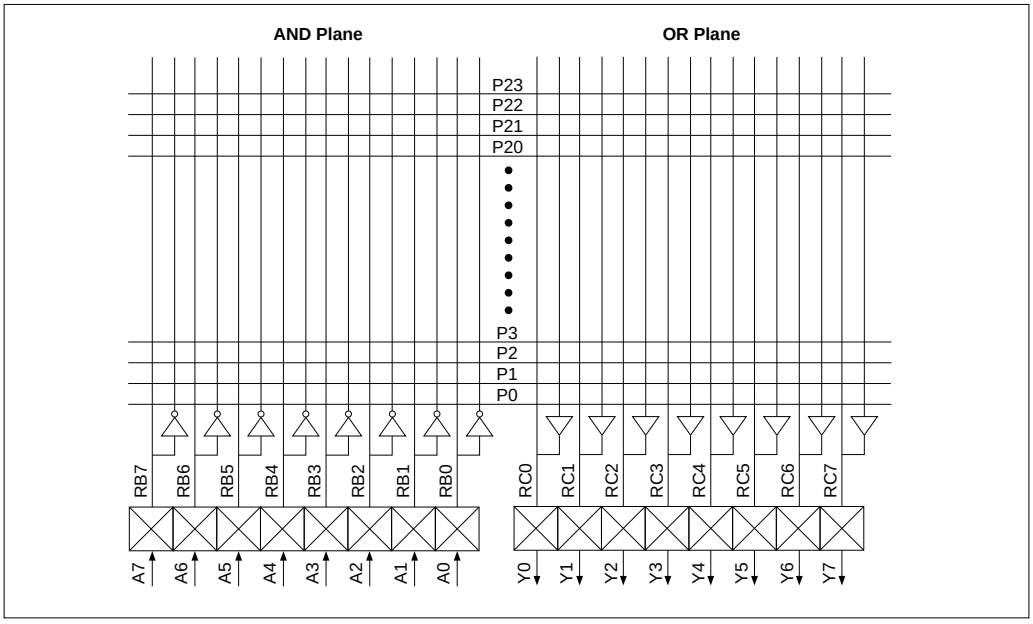
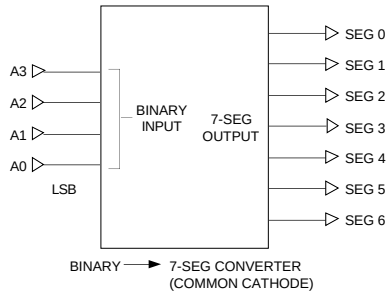
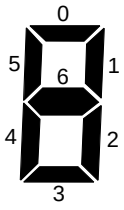


FIGURE 2 - BINARY TO 7-SEGMENT CONVERSION EXAMPLE





Truth Table:

HEX	A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Product Terms
0	0	0	0	0	1	1	1	1	1	1	0	P0 = $\bar{A}3, \bar{A}2, \bar{A}1, \bar{A}0$
1	0	0	0	1	0	1	1	0	0	0	0	P1
2	0	0	1	0	1	1	0	1	1	0	1	P2
3	0	1	1	1	1	1	1	1	0	0	1	P3
4	0	1	0	0	0	1	1	0	0	1	1	P4
5	0	1	0	1	1	0	1	1	0	1	1	P5
6	0	1	1	0	1	0	1	1	1	1	1	P6
7	0	1	1	1	1	1	1	0	0	0	0	P7
8	1	0	0	0	1	1	1	1	1	1	1	P8
9	1	0	0	1	1	1	1	0	0	1	1	P9
A	1	0	1	0	1	1	1	0	1	1	1	P10
B	1	0	1	1	0	0	0	1	1	1	1	P11
C	1	1	0	0	1	0	0	1	1	1	0	P12
D	1	1	0	1	0	1	1	1	1	0	1	P13
E	1	1	1	0	1	0	0	1	1	1	1	P14
F	1	1	1	1	1	0	0	0	1	1	1	P15 = A3, A2, A1, A0

SPEED/RESPONSE TIME

The worst case response time of a PLA implemented in this fashion can be calculated as follows. First, we define td = time required to execute the PLA program assuming the worst case program branches are taken. Then the maximum propagation delay time from input change to valid output = 2td. This is because if an input changes just after the program reads input port, its effect will not show up until the program completes the current execution cycle, re-reads input and recalculates output. This is shown in Figure 3. There are ways to improve the delay time such as sample inputs several times throughout the PLA program and if input change is sensed, return to the beginning rather than execute the rest of the evaluation code.

A Table Look-Up Method For Small PLA Implementation

If the number of inputs is small (8 or less) then a simple table look-up method can be used to implement the PLA. This will improve execution time to around 5 µs (@ 8 MHz input clock). The following code implements the BCD to 7-segment conversion (Figure 2) using this technique.

```
begin   movlw   0ffh        ;
        tris    6           ;Port_b = input
        clrw                ;
        tris    7           ;Port_c = output
pla 88  movf    Port_b, w   ;Read input
        andlw   0fh         ;Mask off bits 7:4
        call    op_tbl      ;
        movwf   Port_c      ;Write output
        goto    pla88       ;
op_tbl  addwf   pc          ;Computed jump for
                             table look-up
        retlw   b"00111111";
        retlw   b"00000110";
        retlw   b"01011011";
        retlw   b"01001111";
        retlw   b"01100110";
        retlw   b"01101101";
        retlw   b"01111101";
        retlw   b"00000111";
        retlw   b"01111111";
        retlw   b"01100111";
        retlw   b"01110111";
        retlw   b"01111000";
        retlw   b"00111001";
        retlw   b"01011110";
        retlw   b"01111001";
        retlw   b"01110001";
```

TABLE 1 - EXECUTION TIME AND RESOURCES NECESSARY FOR DIFFERENT SIZE PLA's

Number of Inputs Including fdbk	Number of Outputs Including fdbk and o/e Control	Number of Products	Number of RAM Locations Required N_{RAM}	Number of Program Memory Locations Required N_{ROM}	Number of Instruction Cycle To Execute PLA N_{CYC}	Real Time @ 20 MHz to Execute PLA	
8	8	24	5	228	228	45.6 µs	Time Efficient
			10	222	352	70.4 µs	Code Efficient
8	8	48	8	447	447	89.4 µs	Time Efficient
			13	384	535	107 µs	Code Efficient
16	16	64	12	1042	1042	208.4 µs	Time Efficient
			22	843	1250	250 µs	Code Efficient
20	24	80	16	1661	1661	372.2 µs	Time Efficient
			40	1462	2221	444.2 µs	Code Efficient

If Ni = Number of inputs
NIW = Number of input words, NIW = $\lceil \frac{Ni}{8} \rceil$
NP = Number of products
NPW = Number of product words, i.e. NPW = $\lceil \frac{NP}{8} \rceil$
NO = Number of outputs
NOW = Number of output words, i.e. NOW = $\lceil \frac{No}{8} \rceil$

Then,
NRAM @ NIW + NOW + N PW : Time efficient
NRAM @ NIW + NOW + 2 N PW + 2: Code efficient
NROM @ 8 + N PW + NOW + N P [2 + 3 NIW] +NO • NPW • 4: Time efficient
NROM @ 17 + N PW + NOW + N P [2 + 3 NIW] +NO [NPW + 3]: Code efficient
NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] +NO [2 NPW • 4]: Time efficient
NCYC @ 8 + N PW + NOW + N P [2 + 3 NIW] +5 N [NPW + 1]: Code efficient

FIGURE 3 - PLA PROGRAM FLOW

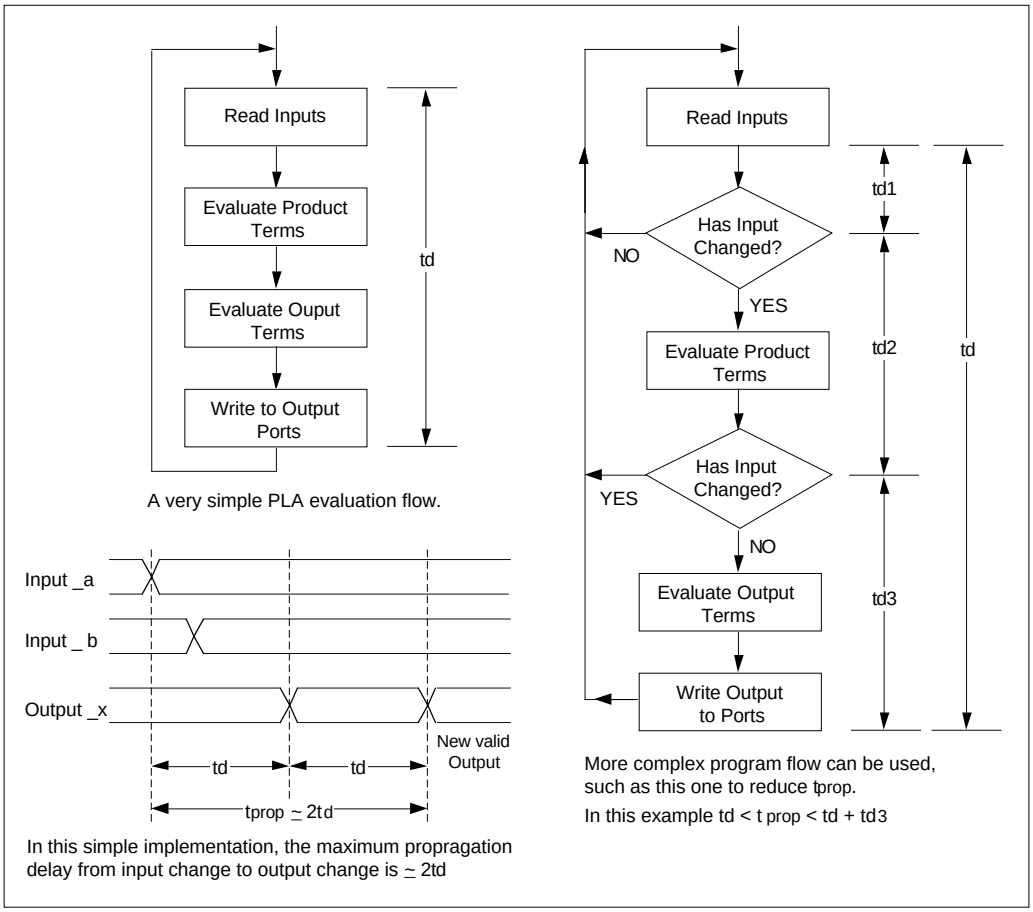
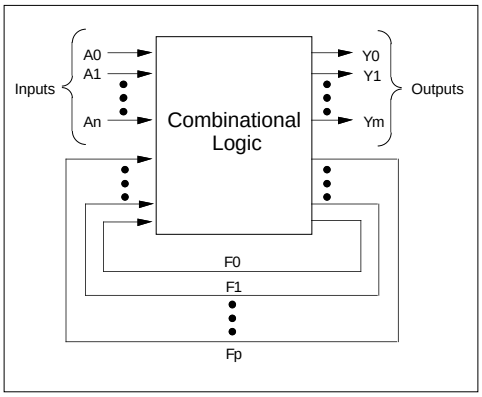


FIGURE 4 - AN ASYNCHRONOUS STATE MACHINE



IMPLEMENTING AN ASYNCHRONOUS STATE MACHINE

The concept can be easily extended to implement sequential logic i.e. a state machine. Figure 4 shows a state machine with n inputs (A0-An), m outputs (Y0-Ym) and p states that feedback as inputs to the PLA (F0-Fp). In PIC16C5X the states will be stored as bits in RAM location. Input will now mean input from a port as well as from the feedback registers. Figure 5 shows an example PLA with eight inputs A0,A1,..,A7 that are connected to port RB pins RB0,RB1,..,RB7. This example PLA has a total of 24 outputs of which eight are actual outputs, another eight are output enable control for the outputs and the other eight are feedbacks (or states). The PLA shown here, therefore, in essence implements an asynchronous state machine (i.e. there is no system clock).

FIGURE 5 - EXAMPLE OF A LARGER PLA IMPLEMENTATION

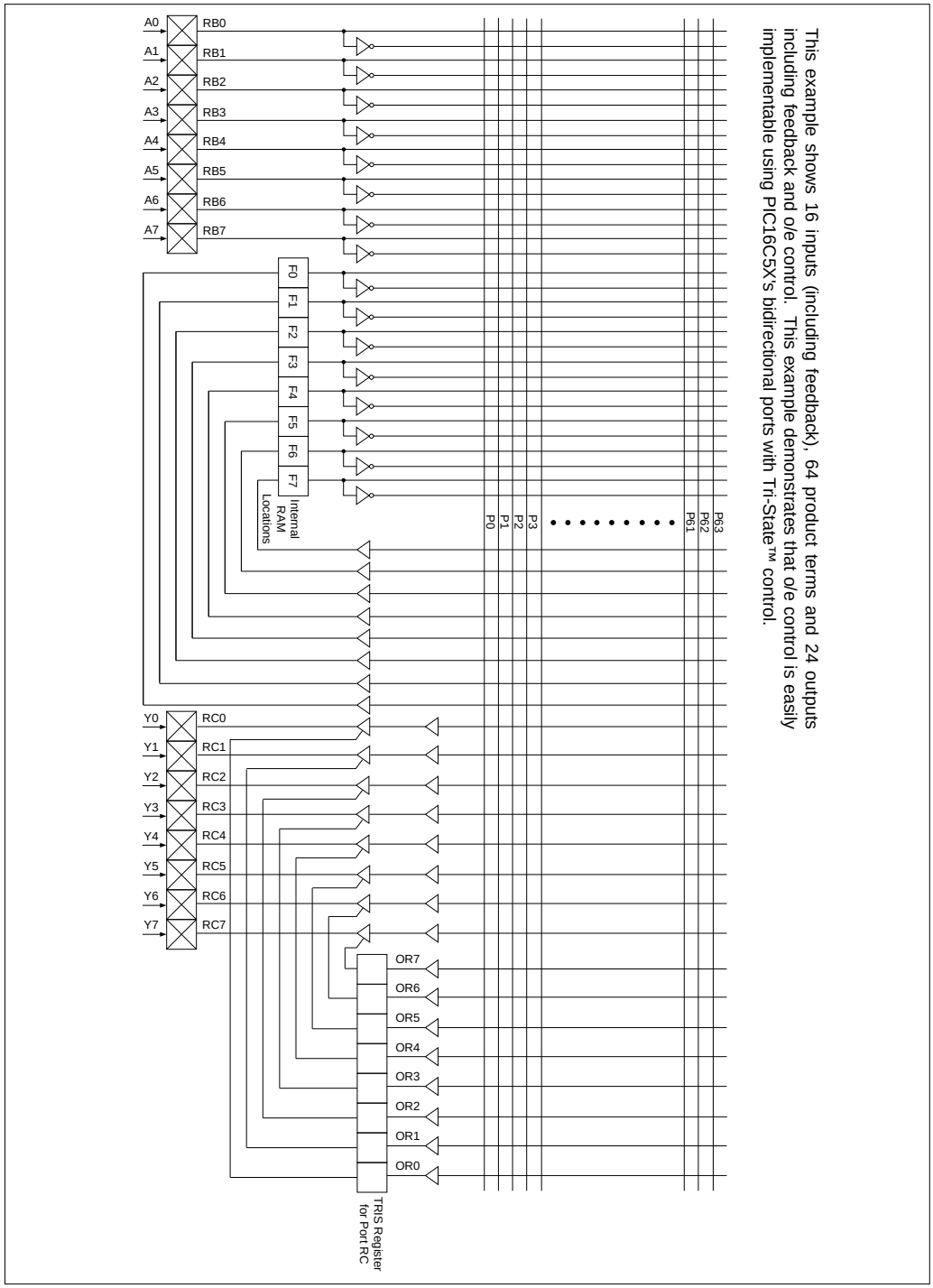
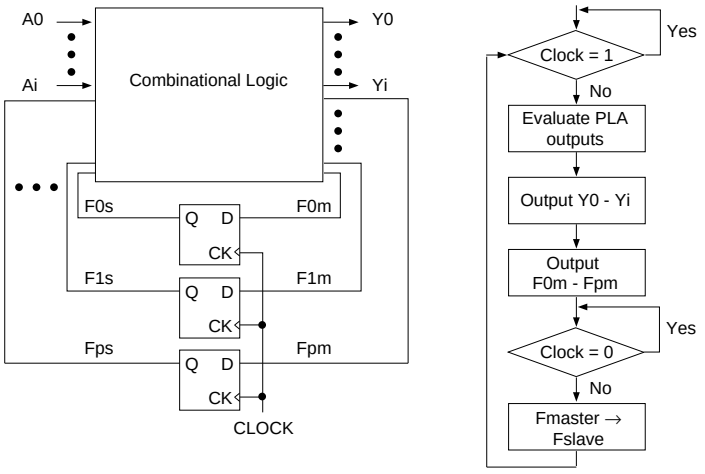
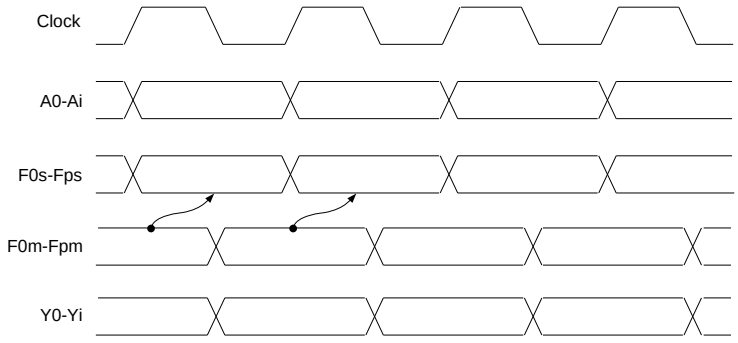


FIGURE 6 - A SYNCHRONOUS STATE MACHINE IMPLEMENTATION



Flowchart for implementing synchronous state machine. Clock is input to an I/O port. Fmaster and Fslave are RAM locations that store state variables F0m, ...Fpm and F0s, ... Fps respectively.



IMPLEMENTING A SYNCHRONOUS STATE MACHINE

In a synchronous system (see Figure 6) usually all inputs are stable at the falling edge (or rising) edge of the system clock. The state machine samples input on the falling edge, evaluates state and output information. The state outputs are latched by the rising edge of the clock before feeding them back to the input (so that they are stable at the falling edge of the clock). To implement such a state machine, the system clock will have to be polled by an input pin. When a falling edge is detected, the PLA evaluation procedure will be invoked to compute outputs and write them to output pins. The PLA procedure will also determine the new state variables, F0m, F1m,...,Fpm and store them in RAM. The program will then wait until a rising edge on the clock input is detected and copy the "master" state variables (F0m,..,Fpm) to slave state variables (F0s,F1s,..,Fps). This step emulates the feedback flip-flops.

SUMMARY

In conclusion, the PIC16C5X can implement a generic PLA equation and provide quick, low cost solution where system operation speed is not critical.

Author: Sumit Mitra
Logic Products Division

APPENDIX A: PLA IMPLEMENTATION: TIME EFFICIENT APPROACH

MPASM B0.54 PAGE 1

```
                ;*****************************************************************
                ; pla1a.asm :
                ; This procedure implements a simple AND-OR PLA with:
                ;
                ;      8  inputs        := A7 A6 A5 A4 A3 A2 A1 A0
                ;      24 product terms := P23 P22 ..... P0
                ;      8  outputs       := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                ;
                ; The eight inputs are assumed to be connected to PORT RB such that
                ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                ; The outputs are programmed to appear on port RC such that
                ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                ;
                ; This implementation optimizes both speed & program memory usage
                ;
                ;*****************************************************************
                ;
                ; define RAM locations used:
                ;
                                LIST    P=16C57
000C            input           equ     d'12'        ;RAM location 12 holds input
000D            Y_reg           equ     d'13'        ;holds output result

000E            Preg_a          equ     d'14'        ;Product terms P0 to P7. Preg_a<0> = P0
000F            Preg_b          equ     d'15'        ;Product terms P8 to P15. Preg_b<0> = P8
0010            Preg_c          equ     d'16'        ;Product terms P16 to P23. Preg_c<0> =
P16
                ; define some constants and file addresses:
                ;
0000            bit0            equ     0            ;
0001            bit1            equ     1            ;
0002            bit2            equ     2            ;
0003            bit3            equ     3            ;
0004            bit4            equ     4            ;
0005            bit5            equ     5            ;
0006            bit6            equ     6            ;
0007            bit7            equ     7            ;
                ;
0003            status          equ     3            ;
0006            port_b          equ     6            ;
0007            port_c          equ     7            ;
                ;
                ; define the AND plane programming variables:
                ;
0000            P0_x            equ     b'00000000'  ;
000F            P0_a            equ     b'00001111'  ;
0001            P1_x            equ     b'00000001'  ;
000F            P1_a            equ     b'00001111'  ;
0002            P2_x            equ     b'00000010'  ;
000F            P2_a            equ     b'00001111'  ;
0003            P3_x            equ     b'00000011'  ;
000F            P3_a            equ     b'00001111'  ;
0004            P4_x            equ     b'00000100'  ;
000F            P4_a            equ     b'00001111'  ;
0005            P5_x            equ     b'00000101'  ;
000F            P5_a            equ     b'00001111'  ;
0006            P6_x            equ     b'00000110'  ;
000F            P6_a            equ     b'00001111'  ;
0007            P7_x            equ     b'00000111'  ;
000F            P7_a            equ     b'00001111'  ;
0008            P8_x            equ     b'00001000'  ;
000F            P8_a            equ     b'00001111'  ;
0009            P9_x            equ     b'00001001'  ;
```

```
000F                P9_a            equ     b’00001111'     ;
O00A                P10_x           equ     b’00001010'     ;
000F                P10_a           equ     b’00001111'     ;
000B                P11_x           equ     b’00001011'     ;
000F                P11_a           equ     b’00001111'     ;
000C                P12_x           equ     b’00001100'     ;
000F                P12_a           equ     b’00001111'     ;
000D                P13_x           equ     b’00001101'     ;
000F                P13_a           equ     b’00001111'     ;
000E                P14_x           equ     b’00001110'     ;
000F                P14_a           equ     b’00001111'     ;
000F                P15_x           equ     b’00001111'     ;
000F                P15_a           equ     b’00001111'     ;
0000                P16_x           equ     b’00000000'     ;
0000                P16_a           equ     b’00000000'     ;
0000                P17_x           equ     b’00000000'     ;
0000                P17_a           equ     b’00000000'     ;
0000                P18_x           equ     b’00000000'     ;
0000                P18_a           equ     b’00000000'     ;
0000                P19_x           equ     b’00000000'     ;
0000                P19_a           equ     b’00000000'     ;
0000                P20_x           equ     b’00000000'     ;
0000                P20_a           equ     b’00000000'     ;
0000                P21_x           equ     b’00000000'     ;
0000                P21_a           equ     b’00000000'     ;
0000                P22_x           equ     b’00000000'     ;
0000                P22_a           equ     b’00000000'     ;
0000                P23_x           equ     b’00000000'     ;
0000                P23_a           equ     b’00000000'     ;

                    ; define OR plane programming variables:
                    x
00ED                OR_a0           equ     b’11101101'     ; for output Y0
00D7                OR_b0           equ     b’11010111'     ;
0000                OR_c0           equ     b’00000000'     ;
009F                OR_a1           equ     b’10011111'     ; for output Y1
0027                OR_b1           equ     b’00100111'     ;
0000                OR_c1           equ     b’00000000'     ;
00FB                OR_a2           equ     b’11111011'     ; for output Y2
002F                OR_b2           equ     b’00101111'     ;
0000                OR_c2           equ     b’00000000'     ;
006D                OR_a3           equ     b’01101101'     ; for output Y3
0079                OR_b3           equ     b’01111001'     ;
0000                OR_c3           equ     b’00000000'     ;
0045                OR_a4           equ     b’01000101'     ; for output Y4
00FD                OR_b4           equ     b’11111101'     ;
0000                OR_c4           equ     b’00000000'     ;
0071                OR_a5           equ     b’01110001'     ; for output Y5
00DF                OR_b5           equ     b’11011111'     ;
0000                OR_c5           equ     b’00000000'     ;
007C                OR_a6           equ     b’01111100'     ; for output Y6
00EF                OR_b6           equ     b’11101111'     ;
0000                OR_c6           equ     b’00000000'     ;
0000                OR_a7           equ     b’00000000'     ; for output Y7
0000                OR_b7           equ     b’00000000'     ;
0000                OR_c7           equ     b’00000000'     ;


                                    org     01ffh           ;
01FF 0A00           begin           goto    main            ;

                                    org     000h            ;

                    ; define macro to evaluate 1 product (AND) term:
                    ;
0000 0902           main            call    pla88           ;
0001 0A00                           goto    main            ;
                    ;
```

```
                        EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                                movf    input,W               ;
                                xorlw   Pn_x                  ;
                                andlw   Pn_a                  ;
                                btfsc   status,bit2           ; skip if zero bit not set
                                bsf     Preg_x,bit_n          ; product term = 1
                                ENDM


                        ; define macro to load OR term constants:
                        ;
                        EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                                LOCAL   SETBIT                ;
                                movf    Preg_a,W              ;
                                andlw   OR_an                 ;
                                btfss   status,bit2           ;
                                goto    SETBIT                ;

                                movf    Preg_b,W              ;
                                andlw   OR_bn                 ;
                                btfss   status,bit2           ;
                                goto    SETBIT                ;

                                movf    Preg_c,W              ;
                                andlw   OR_cn                 ;
                                btfss   status,bit2           ;
                        SETBIT  bsf     Y_reg,bit_n           ;
                                ENDM

                        ; now the  PLA evaluation procedure:
                        ;
0002 0CFF               pla88   movlw   0ffh                  ;
0003 0006                       tris    6                     ; port_b = input
0004 0206                       movf    port_b,W              ; read input
0005 002C                       movwf   input                 ; store input in a register
0006 006E                       clrf    Preg_a                ; clear Product register a
0007 006F                       clrf    Preg_b                ; clear Product register b
0008 0070                       clrf    Preg_c                ; clear Product register c
0009 006D                       clrf    Y_reg                 ; clear output register

                                EVAL_P  Preg_a,bit0,P0_x,P0_a
000A 020C                               movf    input,W       ;
000B 0F00                               xorlw   P0_x          ;
000C 0E0F                               andlw   P0_a          ;
000D 0643                               btfsc   status,bit2   ; skip if zero bit not set
000E 050E                               bsf     Preg_a,bit0   ; product term = 1

                                EVAL_P  Preg_a,bit1,P1_x,P1_a
000F 020C                               movf    input,W       ;
0010 0F01                               xorlw   P1_x          ;
0011 0E0F                               andlw   P1_a          ;
0012 0643                               btfsc   status,bit2   ; skip if zero bit not set
0013 052E                               bsf     Preg_a,bit1   ; product term = 1

                                EVAL_P  Preg_a,bit2,P2_x,P2_a
0014 020C                               movf    input,W       ;
0015 0F02                               xorlw   P2_x          ;
0016 0E0F                               andlw   P2_a          ;
0017 0643                               btfsc   status,bit2   ; skip if zero bit not set
0018 054E                               bsf     Preg_a,bit2   ; product term = 1

                                EVAL_P  Preg_a,bit3,P3_x,P3_a
0019 020C                               movf    input,W       ;
001A 0F03                               xorlw   P3_x          ;
001B 0E0F                               andlw   P3_a          ;
001C 0643                               btfsc   status,bit2   ; skip if zero bit not set
001D 056E                               bsf     Preg_a,bit3   ; product term = 1
```

```
                        EVAL_P  Preg_a,bit4,P4_x,P4_a
001E 020C                       movf    input,W         ;
001F 0F04                       xorlw   P4_x            ;
0020 0E0F                       andlw   P4_a            ;
0021 0643                       btfsc   status,bit2     ; skip if zero bit not set
0022 058E                       bsf     Preg_a,bit4     ; product term = 1

                        EVAL_P  Preg_a,bit5,P5_x,P5_a
0023 020C                       movf    input,W         ;
0024 0F05                       xorlw   P5_x            ;
0025 0E0F                       andlw   P5_a            ;
0026 0643                       btfsc   status,bit2     ; skip if zero bit not set
0027 05AE                       bsf     Preg_a,bit5     ; product term = 1

                        EVAL_P  Preg_a,bit6,P6_x,P6_a
0028 020C                       movf    input,W         ;
0029 0F06                       xorlw   P6_x            ;
002A 0E0F                       andlw   P6_a            ;
002B 0643                       btfsc   status,bit2     ; skip if zero bit not set
002C 05CE                       bsf     Preg_a,bit6     ; product term = 1

                        EVAL_P  Preg_a,bit7,P7_x,P7_a
002D 020C                       movf    input,W         ;
002E 0F07                       xorlw   P7_x            ;
002F 0E0F                       andlw   P7_a            ;
0030 0643                       btfsc   status,bit2     ; skip if zero bit not set
0031 05EE                       bsf     Preg_a,bit7     ; product term = 1


                        EVAL_P  Preg_b,bit0,P8_x,P8_a
0032 020C                       movf    input,W         ;
0033 0F08                       xorlw   P8_x            ;
0034 0E0F                       andlw   P8_a            ;
0035 0643                       btfsc   status,bit2     ; skip if zero bit not set
0036 050F                       bsf     Preg_b,bit0     ; product term = 1

                        EVAL_P  Preg_b,bit1,P9_x,P9_a
0037 020C                       movf    input,W         ;
0038 0F09                       xorlw   P9_x            ;
0039 0E0F                       andlw   P9_a            ;
003A 0643                       btfsc   status,bit2     ; skip if zero bit not set
003B 052F                       bsf     Preg_b,bit1     ; product term = 1

                        EVAL_P  Preg_b,bit2,P10_x,P10_a
003C 020C                       movf    input,W          ;
003D 0F0A                       xorlw   P10_x            ;
003E 0E0F                       andlw   P10_a            ;
003F 0643                       btfsc   status,bit2      ; skip if zero bit not set
0040 054F                       bsf     Preg_b,bit2      ; product term = 1

                        EVAL_P  Preg_b,bit3,P11_x,P11_a
0041 020C                       movf    input,W          ;
0042 0F0B                       xorlw   P11_x            ;
0043 0E0F                       andlw   P11_a            ;
0044 0643                       btfsc   status,bit2      ; skip if zero bit not set
0045 056F                       bsf     Preg_b,bit3      ; product term = 1

                        EVAL_P  Preg_b,bit4,P12_x,P12_a
0046 020C                       movf    input,W          ;
0047 0F0C                       xorlw   P12_x            ;
0048 0E0F                       andlw   P12_a            ;
0049 0643                       btfsc   status,bit2      ; skip if zero bit not set
004A 058F                       bsf     Preg_b,bit4      ; product term = 1

                        EVAL_P  Preg_b,bit5,P13_x,P13_a
004B 020C                       movf    input,W          ;
004C 0F0D                       xorlw   P13_x            ;
004D 0E0F                       andlw   P13_a            ;
004E 0643                       btfsc   status,bit2      ; skip if zero bit not set
004F 05AF                       bsf     Preg_b,bit5      ; product term = 1
```

```
                        EVAL_P  Preg_b,bit6,P14_x,P14_a
0050 020C                       movf    input,W         ;
0051 0F0E                       xorlw   P14_x           ;
0052 0E0F                       andlw   P14_a           ;
0053 0643                       btfsc   status,bit2     ; skip if zero bit not set
0054 05CF                       bsf     Preg_b,bit6     ; product term = 1

                        EVAL_P  Preg_b,bit7,P15_x,P15_a
0055 020C                       movf    input,W         ;
0056 0F0F                       xorlw   P15_x           ;
0057 0E0F                       andlw   P15_a           ;
0058 0643                       btfsc   status,bit2     ; skip if zero bit not set
0059 05EF                       bsf     Preg_b,bit7     ; product term = 1


                        EVAL_P  Preg_c,bit0,P16_x,P16_a
005A 020C                       movf    input,W         ;
005B 0F00                       xorlw   P16_x           ;
005C 0E00                       andlw   P16_a           ;
005D 0643                       btfsc   status,bit2     ; skip if zero bit not set
005E 0510                       bsf     Preg_c,bit0     ; product term = 1

                        EVAL_P  Preg_c,bit1,P17_x,P17_a
005F 020C                       movf    input,W         ;
0060 0F00                       xorlw   P17_x           ;
0061 0E00                       andlw   P17_a           ;
0062 0643                       btfsc   status,bit2     ; skip if zero bit not set
0063 0530                       bsf     Preg_c,bit1     ; product term = 1

                        EVAL_P  Preg_c,bit2,P18_x,P18_a
0064 020C                       movf    input,W         ;
0065 0F00                       xorlw   P18_x           ;
0066 0E00                       andlw   P18_a           ;
0067 0643                       btfsc   status,bit2     ; skip if zero bit not set
0068 0550                       bsf     Preg_c,bit2     ; product term = 1

                        EVAL_P  Preg_c,bit3,P19_x,P19_a
0069 020C                       movf    input,W         ;
006A 0F00                       xorlw   P19_x           ;
006B 0E00                       andlw   P19_a           ;
006C 0643                       btfsc   status,bit2     ; skip if zero bit not set
006D 0570                       bsf     Preg_c,bit3     ; product term = 1

                        EVAL_P  Preg_c,bit4,P20_x,P20_a
006E 020C                       movf    input,W         ;
006F 0F00                       xorlw   P20_x           ;
0070 0E00                       andlw   P20_a           ;
0071 0643                       btfsc   status,bit2     ; skip if zero bit not set
0072 0590                       bsf     Preg_c,bit4     ; product term = 1

                        EVAL_P  Preg_c,bit5,P21_x,P21_a
0073 020C                       movf    input,W         ;
0074 0F00                       xorlw   P21_x           ;
0075 0E00                       andlw   P21_a           ;
0076 0643                       btfsc   status,bit2     ; skip if zero bit not set
0077 05B0                       bsf     Preg_c,bit5     ; product term = 1

                        EVAL_P  Preg_c,bit6,P22_x,P22_a
0078 020C                       movf    input,W         ;
0079 0F00                       xorlw   P22_x           ;
007A 0E00                       andlw   P22_a           ;
007B 0643                       btfsc   status,bit2     ; skip if zero bit not set
007C 05D0                       bsf     Preg_c,bit6     ; product term = 1

                        EVAL_P  Preg_c,bit7,P23_x,P23_a
007D 020C                       movf    input,W         ;
007E 0F00                       xorlw   P23_x           ;
007F 0E00                       andlw   P23_a           ;
0080 0643                       btfsc   status,bit2     ; skip if zero bit not set
```

```
0081 05F0                           bsf     Preg_c,bit7     ; product term = 1
                    or_pl     EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
                              LOCAL   SETBIT          ;
0082 020E                           movf    Preg_a,W        ;
0083 0EED                           andlw   OR_a0           ;
0084 0743                           btfss   status,bit2     ;
0085 0A8D                           goto    SETBIT          ;

0086 020F                           movf    Preg_b,W        ;
0087 0ED7                           andlw   OR_b0           ;
0088 0743                           btfss   status,bit2     ;
0089 0A8D                           goto    SETBIT          ;

008A 0210                           movf    Preg_c,W        ;
008B 0E00                           andlw   OR_c0           ;
008C 0743                           btfss   status,bit2     ;
008D 050D            SETBIT         bsf     Y_reg,bit0      ;

                              EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
                              LOCAL   SETBIT          ;
008E 020E                           movf    Preg_a,W        ;
008F 0E9F                           andlw   OR_a1           ;
0090 0743                           btfss   status,bit2     ;
0091 0A99                           goto    SETBIT          ;

0092 020F                           movf    Preg_b,W        ;
0093 0E27                           andlw   OR_b1           ;
0094 0743                           btfss   status,bit2     ;
0095 0A99                           goto    SETBIT          ;

0096 0210                           movf    Preg_c,W        ;
0097 0E00                           andlw   OR_c1           ;
0098 0743                           btfss   status,bit2     ;
0099 052D            SETBIT         bsf     Y_reg,bit1      ;

                              EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
                              LOCAL   SETBIT          ;
009A 020E                           movf    Preg_a,W        ;
009B 0EFB                           andlw   OR_a2           ;
009C 0743                           btfss   status,bit2     ;
009D 0AA5                           goto    SETBIT          ;

009E 020F                           movf    Preg_b,W        ;
009F 0E2F                           andlw   OR_b2           ;
00A0 0743                           btfss   status,bit2     ;
00A1 0AA5                           goto    SETBIT          ;

00A2 0210                           movf    Preg_c,W        ;
00A3 0E00                           andlw   OR_c2           ;
00A4 0743                           btfss   status,bit2     ;
00A5 054D            SETBIT         bsf     Y_reg,bit2      ;

                              EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
                              LOCAL   SETBIT          ;
00A6 020E                           movf    Preg_a,W        ;
00A7 0E6D                           andlw   OR_a3           ;
00A8 0743                           btfss   status,bit2     ;
00A9 0AB1                           goto    SETBIT          ;

00AA 020F                           movf    Preg_b,W        ;
00AB 0E79                           andlw   OR_b3           ;
00AC 0743                           btfss   status,bit2     ;
00AD 0AB1                           goto    SETBIT          ;

00AE 0210                           movf    Preg_c,W        ;
00AF 0E00                           andlw   OR_c3           ;
00B0 0743                           btfss   status,bit2     ;
00B1 056D            SETBIT         bsf     Y_reg,bit3      ;
```

```
                            EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
                                 LOCAL   SETBIT            ;
00B2 020E                          movf    Preg_a,W        ;
00B3 0E45                          andlw   OR_a4           ;
00B4 0743                          btfss   status,bit2     ;
00B5 0ABD                          goto    SETBIT          ;

00B6 020F                          movf    Preg_b,W        ;
00B7 0EFD                          andlw   OR_b4           ;
00B8 0743                          btfss   status,bit2     ;
00B9 0ABD                          goto    SETBIT          ;

00BA 0210                          movf    Preg_c,W        ;
00BB 0E00                          andlw   OR_c4           ;
00BC 0743                          btfss   status,bit2     ;
00BD 058D           SETBIT         bsf     Y_reg,bit4      ;

                            EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
                                 LOCAL   SETBIT            ;
00BE 020E                          movf    Preg_a,W        ;
00BF 0E71                          andlw   OR_a5           ;
00C0 0743                          btfss   status,bit2     ;
00C1 0AC9                          goto    SETBIT          ;

00C2 020F                          movf    Preg_b,W        ;
00C3 0EDF                          andlw   OR_b5           ;
00C4 0743                          btfss   status,bit2     ;
00C5 0AC9                          goto    SETBIT          ;

00C6 0210                          movf    Preg_c,W        ;
00C7 0E00                          andlw   OR_c5           ;
00C8 0743                          btfss   status,bit2     ;
00C9 05AD           SETBIT         bsf     Y_reg,bit5      ;

                            EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
                                 LOCAL   SETBIT            ;
00CA 020E                          movf    Preg_a,W        ;
00CB 0E7C                          andlw   OR_a6           ;
00CC 0743                          btfss   status,bit2     ;
00CD 0AD5                          goto    SETBIT          ;

00CE 020F                          movf    Preg_b,W        ;
00CF 0EEF                          andlw   OR_b6           ;
00D0 0743                          btfss   status,bit2     ;
00D1 0AD5                          goto    SETBIT          ;

00D2 0210                          movf    Preg_c,W        ;
00D3 0E00                          andlw   OR_c6           ;
00D4 0743                          btfss   status,bit2     ;
00D5 05CD           SETBIT         bsf     Y_reg,bit6      ;

                            EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
                                 LOCAL   SETBIT            ;
00D6 020E                          movf    Preg_a,W        ;
00D7 0E00                          andlw   OR_a7           ;
00D8 0743                          btfss   status,bit2     ;
00D9 0AE1                          goto    SETBIT          ;

00DA 020F                          movf    Preg_b,W        ;
00DB 0E00                          andlw   OR_b7           ;
00DC 0743                          btfss   status,bit2     ;
00DD 0AE1                          goto    SETBIT          ;

00DE 0210                          movf    Preg_c,W        ;
00DF 0E00                          andlw   OR_c7           ;
00E0 0743                          btfss   status,bit2     ;
00E1 05ED           SETBIT         bsf     Y_reg,bit7      ;
```

```
                    ; Y_reg now contains 8 output values:
00E2 0040            wr_out  clrw                            ;
00E3 0007                            tris    7               ; port_c = output
00E4 020D                            movf    Y_reg,W         ;
00E5 0027                            movwf   port_c          ; Y_reg -> port_c
00E6 0800                            retlw   0               ;

00E7 0000            ZZZ             nop

                                    END


Errors   :    0
Warnings :    0
```

APPENDIX B: PLA IMPLEMENTATION: CODE EFFICIENT APPROACH

```
MPASM B0.54                                                          PAGE  1



                    ;*********************************************************************
                    ; pla1b.asm :
                    ; This procedure implements a simple AND-OR PLA with:
                    ;
                    ;      8  inputs        := A7 A6 A5 A4 A3 A2 A1 A0
                    ;      24 product terms := P23 P22 ..... P0
                    ;      8  outputs       := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                    ;
                    ; The eight inputs are assumed to be connected to PORT RB such that
                    ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                    ; The outputs are programmed to appear on port RC such that
                    ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                    ;
                    ; This implementation optimizes program memory usage over
                    ; speed
                    ;*********************************************************************
                    ;
                    ; define RAM locations used:
                    ;
                                   LIST    P=16C57
000C                input           equ     d"12"     ; RAM location 12 holds input
000D                Y_reg           equ     d"13"     ; holds output result

000E                Preg_a          equ     d"14"     ; Product terms P0 to P7. Preg_a<0> = P0
000F                Preg_b          equ     d"15"     ; Product terms P8 to P15. Preg_b<0> = P8
0010                Preg_c          equ     d"16"     ; Product terms P16 to P23. Preg_c<0> = P16
0012                Pn_x            equ     d"18"     ;
0013                Pn_a            equ     d"19"     ;
0014                OR_a            equ     d"20"     ;
0015                OR_b            equ     d"21"     ;
0016                OR_c            equ     d"22"     ;

                    ; define some constants and file addresses:
                    ;
0000                bit0            equ     0          ;
0001                bit1            equ     1          ;
0002                bit2            equ     2          ;
0003                bit3            equ     3          ;
0004                bit4            equ     4          ;
0005                bit5            equ     5          ;
0006                bit6            equ     6          ;
0007                bit7            equ     7          ;
                    ;
0003                status          equ     3          ;
0006                port_b          equ     6          ;
0007                port_c          equ     7          ;
```

```
                    ;
                    ; define the AND plane programming variables:
                    ;
0000                P0_x            equ     b"00000000"     ;
000F                P0_a            equ     b"00001111"     ;
0001                P1_x            equ     b"00000001"     ;
000F                P1_a            equ     b"00001111"     ;
0002                P2_x            equ     b"00000010"     ;
000F                P2_a            equ     b"00001111"     ;
0003                P3_x            equ     b"00000011"     ;
000F                P3_a            equ     b"00001111"     ;
0004                P4_x            equ     b"00000100"     ;
000F                P4_a            equ     b"00001111"     ;
0005                P5_x            equ     b"00000101"     ;
000F                P5_a            equ     b"00001111"     ;
0006                P6_x            equ     b"00000110"     ;
000F                P6_a            equ     b"00001111"     ;
0007                P7_x            equ     b"00000111"     ;
000F                P7_a            equ     b"00001111"     ;
0008                P8_x            equ     b"00001000"     ;
000F                P8_a            equ     b"00001111"     ;
0009                P9_x            equ     b"00001001"     ;
000F                P9_a            equ     b"00001111"     ;
000A                P10_x           equ     b"00001010"     ;
000F                P10_a           equ     b"00001111"     ;
000B                P11_x           equ     b"00001011"     ;
000F                P11_a           equ     b"00001111"     ;
000C                P12_x           equ     b"00001100"     ;
000F                P12_a           equ     b"00001111"     ;
000D                P13_x           equ     b"00001101"     ;
000F                P13_a           equ     b"00001111"     ;
000E                P14_x           equ     b"00001110"     ;
000F                P14_a           equ     b"00001111"     ;
000F                P15_x           equ     b"00001111"     ;
000F                P15_a           equ     b"00001111"     ;
0000                P16_x           equ     b"00000000"     ;
0000                P16_a           equ     b"00000000"     ;
0000                P17_x           equ     b"00000000"     ;
0000                P17_a           equ     b"00000000"     ;
0000                P18_x           equ     b"00000000"     ;
0000                P18_a           equ     b"00000000"     ;
0000                P19_x           equ     b"00000000"     ;
0000                P19_a           equ     b"00000000"     ;
0000                P20_x           equ     b"00000000"     ;
0000                P20_a           equ     b"00000000"     ;
0000                P21_x           equ     b"00000000"     ;
0000                P21_a           equ     b"00000000"     ;
0000                P22_x           equ     b"00000000"     ;
0000                P22_a           equ     b"00000000"     ;
0000                P23_x           equ     b"00000000"     ;
0000                P23_a           equ     b"00000000"     ;

                    ; define OR plane programming variables:

00ED                OR_a0           equ     b"11101101"     ; for output Y0
00D7                OR_b0           equ     b"11010111"     ;
0000                OR_c0           equ     b"00000000"     ;
009F                OR_a1           equ     b"10011111"     ; for output Y1
0027                OR_b1           equ     b"00100111"     ;
0000                OR_c1           equ     b"00000000"     ;
00FB                OR_a2           equ     b"11111011"     ; for output Y2
002F                OR_b2           equ     b"00101111"     ;
0000                OR_c2           equ     b"00000000"     ;
006D                OR_a3           equ     b"01101101"     ; for output Y3
0079                OR_b3           equ     b"01111001"     ;
0000                OR_c3           equ     b"00000000"     ;
0045                OR_a4           equ     b"01000101"     ; for output Y4
00FD                OR_b4           equ     b"11111101"     ;
0000                OR_c4           equ     b"00000000"     ;
```

```
0071                OR_a5           equ     b”01110001"     ; for output Y5
00DF                OR_b5           equ     b”11011111"     ;
0000                OR_c5           equ     b”00000000"     ;
007C                OR_a6           equ     b”01111100"     ; for output Y6
00EF                OR_b6           equ     b”11101111"     ;
0000                OR_c6           equ     b”00000000"     ;
0000                OR_a7           equ     b”00000000"     ; for output Y7
0000                OR_b7           equ     b”00000000"     ;
0000                OR_c7           equ     b”00000000"     ;


                                    org     01ffh           ;
01FF 0A00           begin           goto    main            ;

                                    org     000h            ;
                    ; define macro to evaluate 1 product (AND) term:
0000 090B           main            call    pla88           ;
0001 0A00                           goto    main            ;
                    ;

                    EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                              movf    input,W               ;
                              xorlw   Pn_x                  ;
                              andlw   Pn_a                  ;
                              btfsc   status,bit2           ; skip if zero bit not set
                              bsf     Preg_x,bit_n          ; product term = 1
                              ENDM


                    ; define macro to load OR term constants:
                    ;
                    EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                              movlw   OR_an                 ; load constants
                              movwf   OR_a                  ;
                              movlw   OR_bn                 ;
                              movwf   OR_b                  ;
                              movlw   OR_cn                 ;
                              movwf   OR_c                  ;
                              call    EVAL1                 ;
                              btfss   status,bit2           ;
                              bsf     Y_reg,bit_n           ;
                              ENDM
                    ; define procedure to evaluate 1 output (OR) term:
                    ;
0002 020E           EVAL1           movf    Preg_a,W        ;
0003 0174                           andwf   OR_a,1          ;

0004 020F                           movf    Preg_b,W        ;
0005 0175                           andwf   OR_b,1          ;

0006 0210                           movf    Preg_c,W        ;
0007 0156                           andwf   OR_c,W          ;

0008 0114                           iorwf   OR_a,W          ;
0009 0115                           iorwf   OR_b,W          ;
000A 0800                           retlw   0               ;  W = 1 implies Yn = 1

                    ; now the  PLA evaluation procedure:
                    ;
000B 0CFF           pla88           movlw   0ffh            ;
000C 0006                           tris    6               ; port_b = input
000D 0206                           movf    port_b,W        ; read input
000E 002C                           movwf   input           ; store input in a register
000F 006E                           clrf    Preg_a          ; clear Product register a
0010 006F                           clrf    Preg_b          ; clear Product register b
0011 0070                           clrf    Preg_c          ; clear Product register c
0012 006D                           clrf    Y_reg           ; clear output register
```

```
                    and_pl     EVAL_P  Preg_a,bit0,P0_x,P0_a
0013 020C                             movf    input,W        ;
0014 0F00                             xorlw   P0_x           ;
0015 0E0F                             andlw   P0_a           ;
0016 0643                             btfsc   status,bit2    ; skip if zero bit not set
0017 050E                             bsf     Preg_a,bit0    ; product term = 1

                                    EVAL_P  Preg_a,bit1,P1_x,P1_a
0018 020C                             movf    input,W        ;
0019 0F01                             xorlw   P1_x           ;
001A 0E0F                             andlw   P1_a           ;
001B 0643                             btfsc   status,bit2    ; skip if zero bit not set
001C 052E                             bsf     Preg_a,bit1    ; product term = 1

                                    EVAL_P  Preg_a,bit2,P2_x,P2_a
001D 020C                             movf    input,W        ;
001E 0F02                             xorlw   P2_x           ;
001F 0E0F                             andlw   P2_a           ;
0020 0643                             btfsc   status,bit2    ; skip if zero bit not set
0021 054E                             bsf     Preg_a,bit2    ; product term = 1

                                    EVAL_P  Preg_a,bit3,P3_x,P3_a
0022 020C                             movf    input,W        ;
0023 0F03                             xorlw   P3_x           ;
0024 0E0F                             andlw   P3_a           ;
0025 0643                             btfsc   status,bit2    ; skip if zero bit not set
0026 056E                             bsf     Preg_a,bit3    ; product term = 1

                                    EVAL_P  Preg_a,bit4,P4_x,P4_a
0027 020C                             movf    input,W        ;
0028 0F04                             xorlw   P4_x           ;
0029 0E0F                             andlw   P4_a           ;
002A 0643                             btfsc   status,bit2    ; skip if zero bit not set
002B 058E                             bsf     Preg_a,bit4    ; product term = 1

                                    EVAL_P  Preg_a,bit5,P5_x,P5_a
002C 020C                             movf    input,W        ;
002D 0F05                             xorlw   P5_x           ;
002E 0E0F                             andlw   P5_a           ;
002F 0643                             btfsc   status,bit2    ; skip if zero bit not set
0030 05AE                             bsf     Preg_a,bit5    ; product term = 1

                                    EVAL_P  Preg_a,bit6,P6_x,P6_a
0031 020C                             movf    input,W        ;
0032 0F06                             xorlw   P6_x           ;
0033 0E0F                             andlw   P6_a           ;
0034 0643                             btfsc   status,bit2    ; skip if zero bit not set
0035 05CE                             bsf     Preg_a,bit6    ; product term = 1

                                    EVAL_P  Preg_a,bit7,P7_x,P7_a
0036 020C                             movf    input,W        ;
0037 0F07                             xorlw   P7_x           ;
0038 0E0F                             andlw   P7_a           ;
0039 0643                             btfsc   status,bit2    ; skip if zero bit not set
003A 05EE                             bsf     Preg_a,bit7    ; product term = 1


                                    EVAL_P  Preg_b,bit0,P8_x,P8_a
003B 020C                             movf    input,W        ;
003C 0F08                             xorlw   P8_x           ;
003D 0E0F                             andlw   P8_a           ;
003E 0643                             btfsc   status,bit2    ; skip if zero bit not set
003F 050F                             bsf     Preg_b,bit0    ; product term = 1

                                    EVAL_P  Preg_b,bit1,P9_x,P9_a
0040 020C                             movf    input,W        ;
0041 0F09                             xorlw   P9_x           ;
0042 0E0F                             andlw   P9_a           ;
0043 0643                             btfsc   status,bit2    ; skip if zero bit not set
0044 052F                             bsf     Preg_b,bit1    ; product term = 1
```

```
                          EVAL_P  Preg_b,bit2,P10_x,P10_a
0045 020C                        movf    input,W         ;
0046 0F0A                        xorlw   P10_x           ;
0047 0E0F                        andlw   P10_a           ;
0048 0643                        btfsc   status,bit2     ; skip if zero bit not set
0049 054F                        bsf     Preg_b,bit2     ; product term = 1

                          EVAL_P  Preg_b,bit3,P11_x,P11_a
004A 020C                        movf    input,W         ;
004B 0F0B                        xorlw   P11_x           ;
004C 0E0F                        andlw   P11_a           ;
004D 0643                        btfsc   status,bit2     ; skip if zero bit not set
004E 056F                        bsf     Preg_b,bit3     ; product term = 1

                          EVAL_P  Preg_b,bit4,P12_x,P12_a
004F 020C                        movf    input,W         ;
0050 0F0C                        xorlw   P12_x           ;
0051 0E0F                        andlw   P12_a           ;
0052 0643                        btfsc   status,bit2     ; skip if zero bit not set
0053 058F                        bsf     Preg_b,bit4     ; product term = 1
                          EVAL_P  Preg_b,bit5,P13_x,P13_a
0054 020C                        movf    input,W         ;
0055 0F0D                        xorlw   P13_x           ;
0056 0E0F                        andlw   P13_a           ;
0057 0643                        btfsc   status,bit2     ; skip if zero bit not set
0058 05AF                        bsf     Preg_b,bit5     ; product term = 1

                          EVAL_P  Preg_b,bit6,P14_x,P14_a
0059 020C                        movf    input,W         ;
005A 0F0E                        xorlw   P14_x           ;
005B 0E0F                        andlw   P14_a           ;
005C 0643                        btfsc   status,bit2     ; skip if zero bit not set
005D 05CF                        bsf     Preg_b,bit6     ; product term = 1

                          EVAL_P  Preg_b,bit7,P15_x,P15_a
005E 020C                        movf    input,W         ;
005F 0F0F                        xorlw   P15_x           ;
0060 0E0F                        andlw   P15_a           ;
0061 0643                        btfsc   status,bit2     ; skip if zero bit not set
0062 05EF                        bsf     Preg_b,bit7     ; product term = 1


                          EVAL_P  Preg_c,bit0,P16_x,P16_a
0063 020C                        movf    input,W         ;
0064 0F00                        xorlw   P16_x           ;
0065 0E00                        andlw   P16_a           ;
0066 0643                        btfsc   status,bit2     ; skip if zero bit not set
0067 0510                        bsf     Preg_c,bit0     ; product term = 1

                          EVAL_P  Preg_c,bit1,P17_x,P17_a
0068 020C                        movf    input,W         ;
0069 0F00                        xorlw   P17_x           ;
006A 0E00                        andlw   P17_a           ;
006B 0643                        btfsc   status,bit2     ; skip if zero bit not set
006C 0530                        bsf     Preg_c,bit1     ; product term = 1

                          EVAL_P  Preg_c,bit2,P18_x,P18_a
006D 020C                        movf    input,W         ;
006E 0F00                        xorlw   P18_x           ;
006F 0E00                        andlw   P18_a           ;
0070 0643                        btfsc   status,bit2     ; skip if zero bit not set
0071 0550                        bsf     Preg_c,bit2     ; product term = 1

                          EVAL_P  Preg_c,bit3,P19_x,P19_a
0072 020C                        movf    input,W         ;
0073 0F00                        xorlw   P19_x           ;
0074 0E00                        andlw   P19_a           ;
0075 0643                        btfsc   status,bit2     ; skip if zero bit not set
0076 0570                        bsf     Preg_c,bit3     ; product term = 1
```

```
                            EVAL_P  Preg_c,bit4,P20_x,P20_a
0077 020C                           movf    input,W         ;
0078 0F00                           xorlw   P20_x           ;
0079 0E00                           andlw   P20_a           ;
007A 0643                           btfsc   status,bit2     ; skip if zero bit not set
007B 0590                           bsf     Preg_c,bit4     ; product term = 1

                            EVAL_P  Preg_c,bit5,P21_x,P21_a
007C 020C                           movf    input,W         ;
007D 0F00                           xorlw   P21_x           ;
007E 0E00                           andlw   P21_a           ;
007F 0643                           btfsc   status,bit2     ; skip if zero bit not set
0080 05B0                           bsf     Preg_c,bit5     ; product term = 1

                            EVAL_P  Preg_c,bit6,P22_x,P22_a
0081 020C                           movf    input,W         ;
0082 0F00                           xorlw   P22_x           ;
0083 0E00                           andlw   P22_a           ;
0084 0643                           btfsc   status,bit2     ; skip if zero bit not set
0085 05D0                           bsf     Preg_c,bit6     ; product term = 1

                            EVAL_P  Preg_c,bit7,P23_x,P23_a
0086 020C                           movf    input,W         ;
0087 0F00                           xorlw   P23_x           ;
0088 0E00                           andlw   P23_a           ;
0089 0643                           btfsc   status,bit2     ; skip if zero bit not set
008A 05F0                           bsf     Preg_c,bit7     ; product term = 1


                    or_pl       EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
008B 0CED                           movlw   OR_a0           ; load constants
008C 0034                           movwf   OR_a            ;
008D 0CD7                           movlw   OR_b0           ;
008E 0035                           movwf   OR_b            ;
008F 0C00                           movlw   OR_c0           ;
0090 0036                           movwf   OR_c            ;
0091 0902                           call    EVAL1           ;
0092 0743                           btfss   status,bit2     ;
0093 050D                           bsf     Y_reg,bit0      ;

                            EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
0094 0C9F                           movlw   OR_a1           ; load constants
0095 0034                           movwf   OR_a            ;
0096 0C27                           movlw   OR_b1           ;
0097 0035                           movwf   OR_b            ;
0098 0C00                           movlw   OR_c1           ;
0099 0036                           movwf   OR_c            ;
009A 0902                           call    EVAL1           ;
009B 0743                           btfss   status,bit2     ;
009C 052D                           bsf     Y_reg,bit1      ;

                            EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
009D 0CFB                           movlw   OR_a2           ; load constants
009E 0034                           movwf   OR_a            ;
009F 0C2F                           movlw   OR_b2           ;
00A0 0035                           movwf   OR_b            ;
00A1 0C00                           movlw   OR_c2           ;
00A2 0036                           movwf   OR_c            ;
00A3 0902                           call    EVAL1           ;
00A4 0743                           btfss   status,bit2     ;
```

```
00A5 054D                               bsf     Y_reg,bit2      ;

                                EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
00A6 0C6D                               movlw   OR_a3           ; load constants
00A7 0034                               movwf   OR_a            ;
00A8 0C79                               movlw   OR_b3           ;
00A9 0035                               movwf   OR_b            ;
00AA 0C00                               movlw   OR_c3           ;
00AB 0036                               movwf   OR_c            ;
00AC 0902                               call    EVAL1           ;
00AD 0743                               btfss   status,bit2     ;
00AE 056D                               bsf     Y_reg,bit3      ;

                                EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
00AF 0C45                               movlw   OR_a4           ; load constants
00B0 0034                               movwf   OR_a            ;
00B1 0CFD                               movlw   OR_b4           ;
00B2 0035                               movwf   OR_b            ;
00B3 0C00                               movlw   OR_c4           ;
00B4 0036                               movwf   OR_c            ;
00B5 0902                               call    EVAL1           ;
00B6 0743                               btfss   status,bit2     ;
00B7 058D                               bsf     Y_reg,bit4      ;

                                EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
00B8 0C71                               movlw   OR_a5           ; load constants
00B9 0034                               movwf   OR_a            ;
00BA 0CDF                               movlw   OR_b5           ;
00BB 0035                               movwf   OR_b            ;
00BC 0C00                               movlw   OR_c5           ;
00BD 0036                               movwf   OR_c            ;
00BE 0902                               call    EVAL1           ;
00BF 0743                               btfss   status,bit2     ;
00C0 05AD                               bsf     Y_reg,bit5      ;

                                EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
00C1 0C7C                               movlw   OR_a6           ; load constants
00C2 0034                               movwf   OR_a            ;
00C3 0CEF                               movlw   OR_b6           ;
00C4 0035                               movwf   OR_b            ;
00C5 0C00                               movlw   OR_c6           ;
00C6 0036                               movwf   OR_c            ;
00C7 0902                               call    EVAL1           ;
00C8 0743                               btfss   status,bit2     ;
00C9 05CD                               bsf     Y_reg,bit6      ;

                                EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
00CA 0C00                               movlw   OR_a7           ; load constants
00CB 0034                               movwf   OR_a            ;
00CC 0C00                               movlw   OR_b7           ;
00CD 0035                               movwf   OR_b            ;
00CE 0C00                               movlw   OR_c7           ;
00CF 0036                               movwf   OR_c            ;
00D0 0902                               call    EVAL1           ;
00D1 0743                               btfss   status,bit2     ;
00D2 05ED                               bsf     Y_reg,bit7      ;


                        ; Y_reg now contains 8 output values:
00D3 0040               wr_out          clrw                    ;
00D4 0007                               tris    7               ; port_c = output
00D5 020D                               movf    Y_reg,W         ;
00D6 0027                               movwf   port_c          ; Y_reg -> port_c
00D7 0800                               retlw   0               ;

00D8 0000               ZZZ             nop

                                END
Errors   :    0
Warnings :    0
```

AN593

Serial Port Routines Without Using the RTCC

INTRODUCTION

The PIC16C5X has one 8-bit timer (RTCC) which can use an 8-bit prescaler. In some instances, the user would like to use this timer for some other purposes and yet be able to do a transmit and receive using the serial port. This application note offers routines to do a simple 8-bit transmit and receive with no handshake, at baud rates from 1200 to 9600. Please note that these routines use a timed loop which is as accurate as the clock which drives the PIC16C5X. The user enters the frequency and baud rate desired. The calculated value “delay” in the serial routine has to be an 8-bit value only. If the value is greater than 8-bits, the frequency and baud rate values have to be changed.

CONCLUSION

Simple transmit and receive routines can be written without using RTCC to generate the baud rate.

Author: Stan D'Souza
Logic Products Division

APPENDIX A

```
MPASM 00.00.66 Beta               6-24-1994  7:4:48                PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;
                     0002 ;These routines were written to work on the PICDEM1 hardware.
                     0003 ; The frequency of the clock is 16 Mhz and the hardware uses no
                     0004 ;handshake
                     0005 ;       TX —> RA3
                     0006 ;       RX —> RA2
                     0007         list p=16c54,f=inhx8m
                     0008 ;
00F4 2400            0009 clockrate equ .16000000
2580                 0010 baudrate equ    .9600
                     0011 ;
003D 0900            0012 fclk  equ     clockrate/4
                     0013 ;*****************************************************
                     0014 ;The value baudconst must be a 8 bit value only
0088                 0015 baudconst       equ     ((fclk/baudrate)/3 - 2)
                     0016 ;*****************************************************
0010                 0017 count   equ     0x10
0011                 0018 txreg   equ     0x11
0011                 0019 rcreg   equ     0x11
0012                 0020 delay   equ     0x12
0013                 0021 tempa   equ     0x13
0010                 0022 hi      equ     0x10
0011                 0023 lo      equ     0x11
0015                 0024 gpram   equ     0x15
                     0025 ;
                     0026         include “pic5x.h”
                     0001 ;This is the common header file for all PIC16C5X parts.
                     0002 ;
                     0003 ;
                     0004         CBLOCK 0x00
0000 0005            0005                 _indf, _rtcc, _pcl, _status, _fsr
0005 0002            0006                 _porta, _portb
                     0007         ENDC
                     0008 ;
                     0009 ; Porta Bits
0001                 0010  #define         _ra0            _porta,0
0002                 0011  #define         _ra1            _porta,1
0003                 0012  #define         _ra2            _porta,2
0004                 0013  #define         _ra3            _porta,3
                     0014
                     0015
                     0016 ; Portb bits
0005                 0017  #define         _rb0            _portb,0
0006                 0018  #define         _rb1            _portb,1
0007                 0019  #define         _rb2            _portb,2
0008                 0020  #define         _rb3            _portb,3
0009                 0021  #define         _rb4            _portb,4
000A                 0022  #define         _rb5            _portb,5
000B                 0023  #define         _rb6            _portb,6
000C                 0024  #define         _rb7            _portb,7
                     0025 ;
                     0026 ; STATUS Reg Bits
000D                 0027  #define         _carry          _status,0
000E                 0028  #define         _c              _status,0
000F                 0029  #define         _dc             _status,1
0010                 0030  #define         _z              _status,2
0011                 0031  #define         _pd             _status,3
0012                 0032  #define         _to             _status,4
0013                 0033  #define         _pa0            _status,5
0014                 0034  #define         _pa1            _status,6
                     0035 ;
                     0026
```

```
                    0027 ;
0015                0028  #define _tx     _porta,3
0016                0029  #define _rx     _porta,2
                    0030 ;
                    0031          org     0
                    0032 start
0000 095A           0033          call    wait
0001 0C0F           0034          movlw   B'00001111'
0002 0025           0035          movwf   _porta
0003 0C07           0036          movlw   B'00000111'
0004 0005           0037          tris    _porta
0005 0066           0038          clrf    _portb
0006 0040           0039          clrw
0007 0006           0040          tris    _portb
0008 0CA5           0041          movlw   0xa5
0009 0195           0042          xorwf   gpram,w
000A 0643           0043          btfsc   _z
000B 0964           0044          call    Mclr
000C 0CA5           0045          movlw   0xa5
000D 0035           0046          movwf   gpram
                    0047 ;
                    0048 ;
000E 0C2F           0049          movlw   0x2f
000F 0033           0050          movwf   tempa
0010 0C38           0051          movlw   B'00111000'
0011 0002           0052          option
0012 0061           0053          clrf    _rtcc
                    0054 S1check
0013 0201           0055          movf    _rtcc,w
0014 0643           0056          btfsc   _z              ;if S1 pressed then skip
0015 0A13           0057          goto    S1check
                    0058 ;
                    0059 next
0016 02B3           0060          incf    tempa
0017 06F3           0061          btfsc   tempa,7
0018 0A61           0062          goto    AllDone
0019 0213           0063          movf    tempa,w
001A 0920           0064          call    transmit
001B 0937           0065          call    receive
001C 0093           0066          subwf   tempa,w
001D 0643           0067          btfsc   _z
001E 0A47           0068          goto    fail
001F 0A16           0069          goto    next
                    0070 ;
                    0071 ;
                    0072 transmit
0020 0031           0073          movwf   txreg
0021 0465           0074          bcf     _tx             ;send start bit
0022 0C88           0075          movlw   baudconst
0023 0032           0076          movwf   delay
0024 0C09           0077          movlw   .9
0025 0030           0078          movwf   count
                    0079 txbaudwait
0026 02F2           0080          decfsz  delay
0027 0A26           0081          goto    txbaudwait
0028 0C88           0082          movlw   baudconst
0029 0032           0083          movwf   delay
002A 02F0           0084          decfsz  count
002B 0A30           0085          goto    SendNextBit
002C 0C09           0086          movlw   .9
002D 0030           0087          movwf   count
002E 0565           0088          bsf     _tx             ;send stop bit
002F 0800           0089          return
                    0090 SendNextBit
0030 0331           0091          rrf     txreg
0031 0703           0092          btfss   _c
0032 0A35           0093          goto    Setlo
0033 0565           0094          bsf     _tx
0034 0A26           0095          goto    txbaudwait
```

```
                     0096 Setlo
0035 0465            0097         bcf     _tx
0036 0A26            0098         goto    txbaudwait
                     0099 ;
                     0100 ;
                     0101 receive
0037 0645            0102        btfsc    _rx
0038 0A37            0103        goto     receive         ;wait for receive
                     0104 rxbaudwait
0039 02F2            0105         decfsz  delay
003A 0A39            0106         goto    rxbaudwait
003B 0C88            0107         movlw   baudconst
003C 0032            0108         movwf   delay
003D 02F0            0109         decfsz  count
003E 0A42            0110         goto    RecvNextBit
003F 0C09            0111         movlw   .9
0040 0030            0112         movwf   count
0041 0800            0113         return
                     0114 RecvNextBit
0042 0403            0115         bcf     _c
0043 0645            0116         btfsc   _rx
0044 0503            0117         bsf     _c
0045 0331            0118         rrf     rcreg
0046 0A39            0119         goto    rxbaudwait
                     0120 ;
                     0121 fail
0047 0266            0122         comf    _portb
0048 094A            0123         call    halfsec
0049 0A47            0124         goto    fail
                     0125 halfsec
004A 0070            0126         clrf    hi
004B 0071            0127         clrf    lo
                     0128 hsloop
004C 0000            0129         nop
004D 0000            0130         nop
004E 0000            0131         nop
004F 0000            0132         nop
0050 0000            0133         nop
0051 0000            0134         nop
0052 0000            0135         nop
0053 0000            0136         nop
0054 0000            0137         nop
0055 02F1            0138         decfsz  lo
0056 0A4C            0139         goto    hsloop
0057 02F0            0140         decfsz  hi
0058 0A4C            0141         goto    hsloop
0059 0800            0142         return
                     0143 ;
                     0144 wait
005A 0070            0145         clrf    hi
005B 0071            0146         clrf    lo
                     0147 dly
005C 02F1            0148         decfsz  lo
005D 0A5C            0149         goto    dly
005E 02F0            0150         decfsz  hi
005F 0A5C            0151         goto    dly
0060 0800            0152         return
                     0153
                     0154 ;
                     0155 AllDone
0061 0C55            0156         movlw   0x55
0062 0026            0157         movwf   _portb
0063 0A63            0158         goto    $
                     0159 ;
                     0160 Mclr
0064 0066            0161         clrf    _portb
0065 00E6            0162         decf    _portb
0066 0075            0163         clrf    gpram
                     0164 S3check
```

```
0067 0625           0165         btfsc   _ra1
0068 0A67           0166         goto    S3check
0069 0066           0167         clrf    _portb
006A 0A6A           0168         goto    $
                    0169 ;
                    0170 ;
                    0171         org     0x1ff
01FF 0A00           0172         goto    start
                    0173 ;
                    0174         end
                    0175
                    0176
                    0177
                    0178
                    0179
                    0180
                    0181

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX-- --------

0180 : -------- -------- -------- --------
01C0 : -------- -------- -------- -------X

All other memory blocks unused.


Errors   :    0
Warnings :    0
```


NOTES:

AN510

Implementation of an Asynchronous Serial I/O

INTRODUCTION

The PIC16C5X series from Microchip Technology Inc., are 8-bit, high-speed, EPROM-based microcontrollers. This application note describes the implementation of an Asynchronous serial I/O using Microchip's PIC16C5X series of high-speed 8-bit microcontrollers. These EPROM based microcontrollers can operate at very high speeds with a minimum of 200 ns cycle time @ 20 MHz input clock. Many microcontroller applications require chip-to-chip serial data communications. Since the PIC16C5X series have no on chip serial ports, serial communication has to be performed in software. For many cost-sensitive high volume applications, implementation of a serial I/O through software provides a more cost effective solution than dedicated logic. This application note provides code for PIC16C5X to simulate a serial port using two I/O Pins (one as input for reception and the other as output for transmission).

IMPLEMENTATION

Two programs are provided in this application note. One program simulates a full duplex RS-232 communication and the other provides implementation of half duplex communication. Using Half-Duplex, rates up to 19200 baud can be implemented using an 8 MHz input clock. In case of Full-Duplex, the software can handle up to 9600 baud at 8 MHz and 19200 baud at 20 MHz, one or two stop bits, eight or seven data bits, No Parity and can transmit or receive with either LSB first (normal mode) or MSB first (CODEC like mode). It should be noted that the higher the input clock the better the resolution. These options should be set up during assembly time and not during run time. The user simply has to change the header file for the required communication options. The software does not provide any handshaking protocols. With minor modifications, the user may incorporate software handshaking using XON/XOFF. To implement hardware handshaking, an additional two digital I/O Pins may be used as RTS (ready to send) and CTS (clear to send) lines.

Figure 1 shows a flow chart for serial transmission and Figure 2 shows a flow chart for reception. The flowcharts show cases for transmission/reception with LSB first and eight data bits. For reception, the data receive pin, DR, is polled approximately every B/2 seconds (52 µs in case of 9600 baud) to detect the start bit, where B is the time duration of one bit (B = 1/Baud). If a start bit is found, then the first data bit is checked for after 1.25B seconds. From then on, the other data bits are checked every B seconds (104µs in case of 9600 baud).

In the case of transmission, first a start bit is sent by setting the transmit data pin, DX to zero for B seconds, and from then on the DX pin is set/cleared corresponding to the data bit every B seconds. Assembly language code corresponding to the following flowcharts is given in Figures 3 and 4.

FIGURE 1 - TRANSMISSION FLOW CHART

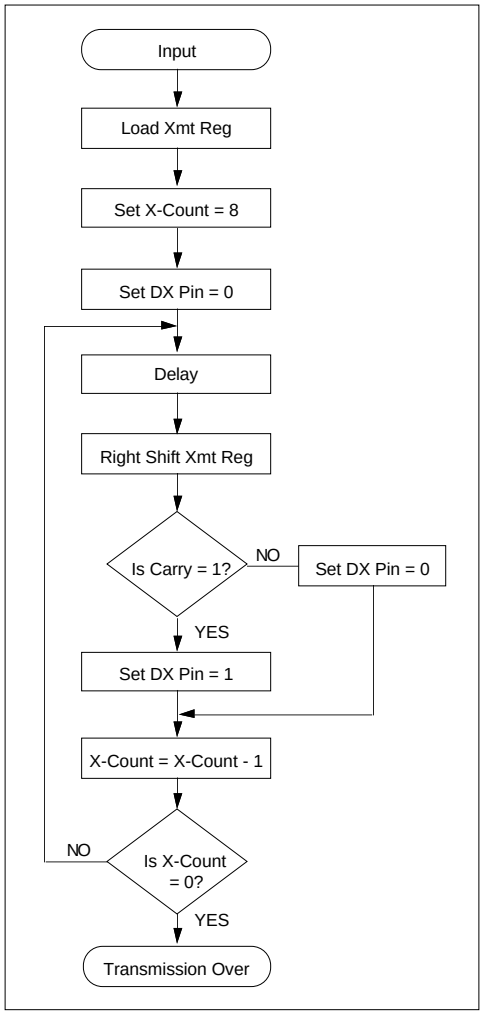


FIGURE 2 - RECEPTION FLOW CHART

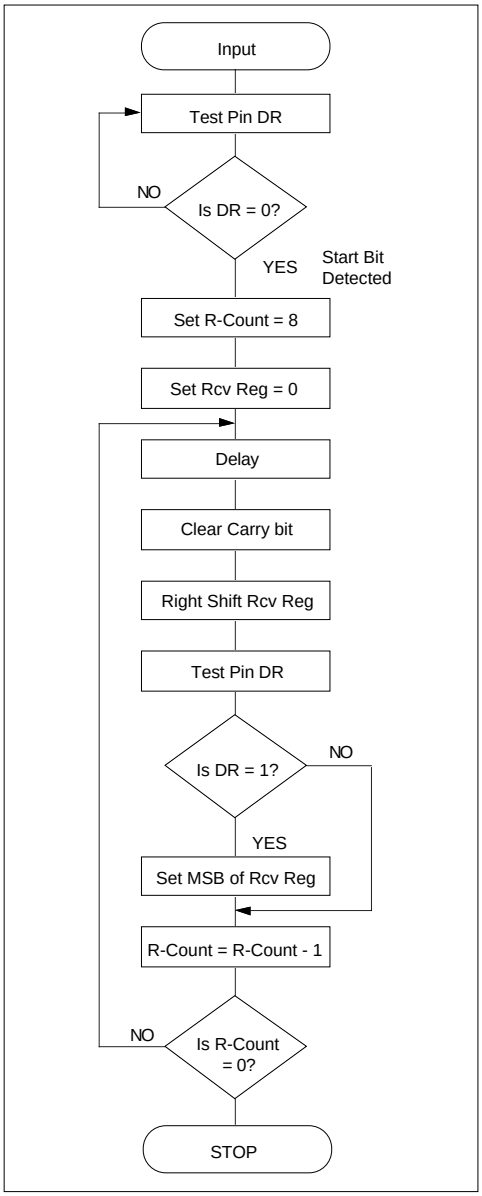


FIGURE 3 - TRANSMIT ASSEMBLY CODE (CORRESPONDING TO FIGURE 1)

```
        ;*********************  Transmitter*****************************
        Xmtr   movlw  8             ; Assume XmtReg contains data to be Xmted
               movwf  XCount        ; 8 data bits
               bcf    Port_A,DX     ; Send Start Bit
        X_next call   Delay         ; Delay for B/2 Seconds
               rrf    XmtReg
               btfsc  STATUS,CARRY  ; Test the bit to be transmitted
               bsf    Port_A,DX     ; Bit is a one
               btfss  STATUS,CARRY
               bcf    Port_A,DX     ; Bit is zero
               decfsz Count         ; If count = 0, then transmit a stop bit
               goto   X_next        ; transmit next bit
        ;
        X_Stop call   Delay
               bsf    Port_A,DX       ; Send Stop Bit
        X_Over goto   X_Over
```

FIGURE 4 - RECEIVE ASSEMBLY CODE (CORRESPONDING TO FIGURE 2)

```
    ;***************    Receiver  *********************
           ;
           Rcvr   btfsc  Port_A,DR     ; Test for Start Bit
                  goto   Rcvr          ; Start Bit not found
                  movlw  8             ; Start Bit Detected
                  movwf  RCount        ; 8 Data Bits
                  clrf   RcvReg        ; Receive Data Register
           R_next call   Delay         ; Delay for B/2 Seconds, B=Time duration of 1
    Bit
                  bcf    STATUS,CARRY  ; Clear CARRY bit
                  rrf    RcvReg        ; to set if MSB first or LSB first
                  btfsc  Port_A,DR     ; Is the bit a zero or one ?
                  bsf    RcvReg,MSB    ; Bit is a one
                  call   Delay
                  decfsz RCount
                  goto   R_next
           R_Over goto   R_Over        ; Reception done
```

The software is organized such that the communication software acts as a Real Time Operating System (RTOS) which gives control to the User routine for a certain time interval. After this predetermined time slot, the user must give back the control to the Operating System. This is true only in the case of full-duplex implementation. Timing considerations are such that the user gets control for approximately half the time of the bit rate and the rest of the half time is used up by the Operating System (and software delays). Please refer to Table 1 for the delay constants and the time the User gets at 8 MHz input clock. Delay constants and the time that the User gets at 20 MHz and 4 MHz input clock speeds are given in the source code listing of the full duplex routine. At frequencies other than 4, 8, or 20 MHz, the delay constants and the time the user gets can be computed from the equations given in Figure 6.

FIGURE 5 - FULL DUPLEX BLOCK DIAGRAM

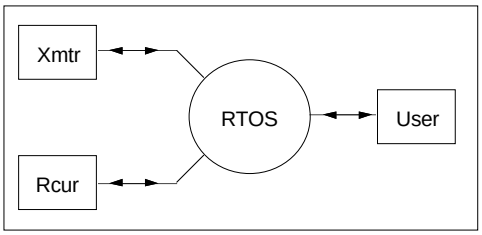


FIGURE 6 - EQUATIONS FOR DELAY CONSTANTS

```
 Baud_Cycles = Clkout/Baud ;
 User_time = Baud_Cycles* (float) 0.5 ;
K0 =  (1.25*Baud_Cycles - 2.0*User_time - 89)/3.0     ; IF (K0 < 0)
   {
   K0 = 0.0 ;
   User_time = 0.50* (1.25*Baud_Cycles - 89.0) ;
   }
 K1 =  (1.25*Baud_Cycles-18 - User_time - 59.0 - 3.K0)/3.0 ;
 K2 =  (Baud_Cycles - User_time - 41.0 - 3.K0)/3.0 ;
 K3 =  (Baud_Cycles - User_time - 61.0 - 3.K0)/3.0 ;
 K4 =  (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 ;
 K5 =  (Baud_Cycles - User_time - 55.0 - 3.K0)/3.0 +1.0 ;
 K6 =  0.0 ;
 K7 =  (1.25*Baud_Cycles - User_time - 39.0 - 3.K0)/3.0 ;
```

TABLE 1 - DELAY CONSTANTS AT 8 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	-	0	5	39	109
K1	-	39	80	150	288
K2	-	27	51	86	155
K3	-	21	44	80	148
K4	-	23	46	82	150
K5	-	24	47	83	151
K6	-	0	0	0	0
K7	-	45	86	156	295
User Cycles	-	86	208	416	832

TABLE 2 - DELAY CONSTANTS AT 20 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	0	13	57	143	317
K1	49	98	184	358	705
K2	34	60	103	191	364
K3	27	53	96	184	357
K4	29	55	98	186	359
K5	30	56	99	187	360
K6	0	0	0	0	0
K7	56	104	190	365	712
User Cycles	118	260	521	1042	2083

For example, if the baud rate selected is 9600 bps (@ 8 MHz) , then the total time frame for one bit is approximately 104 µs. Out of this 104 µs, 61 µs is used by the Operating System and the other 4 µs is available to the User. It is the User's responsibility to return control to the Operating System exactly after the time specified in Table 1. For very accurate timing (with resolution up to one clock cycle) the User may set up the RTCC timer with the Prescaler option for calculating the real time. With RTCC set to increment on internal CLKOUT (500 ns @ 8 MHz CLKIN) and the prescaler assigned to it, very accurate and long timing delay loops may be assigned. This method of attaining accurate delay loops is not used in the RS232 code (RTOS), so that the RTCC is available to the User for other important functions. If the RTCC is not used for other functions, the User may modify the code to replace the software delay loops, by counting the RTCC. For an example of using this method of counting exact timing delays, refer to the "User" routine in Full-Duplex code (Appendix B).

The software uses minimal processor resources. Only six data RAM locations (File Registers) are used. The RTOS uses one level of stack, but it is freed once the control is given back to the user. The watchdog timer and RTCC are not used. The user should clear the watchdog timer at regular intervals, if the WDT is enabled.

The usage of the program is described below. The user should branch to location "Op_Sys" exactly after the time specified in Table 1 or as computed from Equations in Figure 6. Whereas, the transmission is totally under User control, the Reception is under the control of the Operating System. As long as the user does not set the X_flag, no transmission occurs. On the other hand the Operating System is constantly looking for a start bit and the user should not modify either R_done flag or RcvReg.

TRANSMISSION

Transmit Data is output on DX pin (Bit 0 of Port_A). In the user routine, the user should load the data to be transmitted in the XmtReg and Set the X_flag (bsf FlagRX,X_flag). This flag gets cleared after the transmission. The user should check this flag (X_flag) to see if transmission is in progress. Modifying XmtReg when X_flag is set will cause erroneous data to be transmitted.

RECEPTION

Data is received on pin DR (Bit 1 of Port_A). The User should constantly check the "R_done" flag to see if reception is over. If the reception is in progress, R_flag is set. If the reception is over, "R_done" flag is set to 1. The "R_done" flag gets reset to zero when a next start bit is detected. The user should constantly check the R_done flag, and if SET, then the received word is in Register "RcvReg". This register gets cleared when a new start bit is detected. It is recommended that the receive register RcvReg be copied to another register after the R_done flag is set. The R_done flag also gets cleared when the next start bit is detected.

The user may modify the code to implement an N deep buffer (limited to the number of Data RAM locations available) for receive. Also, if receiving at high speeds, and if the N deep buffer is full, an XOFF signal (HEX 13) may be transmitted. When ready to receive more data, an XON signal (HEX 11) should be transmitted.

SUMMARY

PIC16C5X family of microcontrollers allow users to implement half or full duplex RS-232 communication.

Implementation of an Asynchronous Serial I/O

Author: Amar Palacherla
Logic Products Division

2

APPENDIX A: ASSEMBLY LANGUAGE FOR HALF DUPLEX

```
MPASM 01.00.02 Alpha  C:\AP-NOTES\   6-24-1994  8:56:3                    PAGE  1

LOC  OBJECT CODE     LINE  SOURCE TEXT

                     0001 ;
                     0002 ;                    RS-232 Communication With PIC16C54
                     0003 ;
                     0004 ;          Half Duplex Asynchronous Communication
                     0005 ;
                     0006 ;          This program has been tested at Bauds from 1200 to 19200 Baud
                     0007 ;          ( @ 8,16,20 Mhz CLKIN )
                     0008 ;
                     0009 ;          As a test, this program will echo back the data that has been
                     0010 ;          received.
                     0011 ;
                     0012 ;
                     0013          LIST      P=16C54, T=ON
                     0001          INCLUDE   "PICREG.H"
                     0001 ;*****************************  PIC16C5X Header  ********************************
01FF                 0002 PIC54    equ   1FFH    ; Define Reset Vectors
01FF                 0003 PIC55    equ   1FFH
03FF                 0004 PIC56    equ   3FFH
07FF                 0005 PIC57    equ   7FFH
                     0006 ;
0001                 0007 RTCC     equ   1
0002                 0008 PC       equ   2
0003                 0009 STATUS   equ   3     ; F3 Reg is STATUS Reg.
0004                 0010 FSR      equ   4
                     0011 ;
0005                 0012 Port_A   equ   5
0006                 0013 Port_B   equ   6     ; I/O Port Assignments
0007                 0014 Port_C   equ   7
                     0015 ;
                     0016 ;*****************************************************************************
                     0017 ;
                     0018 ;                    ; STATUS REG. Bits
0000                 0019 CARRY    equ   0     ; Carry Bit is Bit.0 of F3
0000                 0020 C        equ   0
0001                 0021 DCARRY   equ   1
0001                 0022 DC       equ   1
0002                 0023 Z_bit    equ   2     ; Bit 2 of F3 is Zero Bit
0002                 0024 Z        equ   2
0003                 0025 P_DOWN   equ   3
0003                 0026 PD       equ   3
0004                 0027 T_OUT    equ   4
0004                 0028 TO       equ   4
0005                 0029 PA0      equ   5
```

```
0006 0030 PA1    equ   6
0007 0031 PA2    equ   7
     0032 ;
0001 0033 Same   equ   1
0000 0034 W      equ   0
     0035 ;
0000 0036 LSB    equ   0
0007 0037 MSB    equ   7
     0038 ;
0001 0039 TRUE   equ   1
0001 0040 YES    equ   1
0000 0041 FALSE  equ   0
0000 0042 NO     equ   0
     0043 ;
     0044 ;*****************************************************************************
     0045
     0013 ;*********************  Communication Parameters   ***************************
     0014 ;
0001 0015 X_MODE  equ   1      ; If ( X_MODE==1) Then transmit LSB first
     0016 ;                      if ( X_MODE==0) Then transmit MSB first ( CODEC like )
0001 0017 R_MODE  equ   1      ; If ( R_MODE==1) Then receive LSB first
     0018 ;                      if ( X_MODE==0) Then receive MSB first ( CODEC like )
0001 0019 X_Nbit  equ   1      ; if (X_Nbit==1) # of data bits ( Transmission ) is 8 else 7
0001 0020 R_Nbit  equ   1      ; if (R_Nbit==1) # of data bits ( Reception ) is 8 else 7
     0021 ;
0000 0022 Sbit2   equ   0      ; if Sbit2 = 0 then 1 Stop Bit else 2 Stop Bits
     0023 ;
     0024 ;*****************************************************************************
0005 0025 X_flag  equ   PA0    ; Bit 5 of F3 ( PA0 )
0006 0026 R_flag  equ   PA1    ; Bit 6 of F3 ( PA1 )
     0027 ;
0000 0028 DX      equ   0      ; Transmit Pin ( Bit 0 of Port A )
0001 0029 DR      equ   1      ; Recive Pin ( Bit 1 of Port A )
     0030 ;
     0031 ;
0044 0032 BAUD_1  equ   .68    ; 3+3X = CLKOUT/Baud
0043 0033 BAUD_2  equ   .67    ; 6+3X = CLKOUT/Baud
0022 0034 BAUD_3  equ   .34    ; 3+3X = 0.5*CLKOUT/Baud
0056 0035 BAUD_4  equ   .86    ; 3+3X = 1.25*CLKOUT/Baud
0042 0036 BAUD_X  equ   .66    ; 11+3X = CLKOUT/Baud
0042 0037 BAUD_Y  equ   .66    ; 9 +3X = CLKOUT/Baud
     0038 ;
     0039 ;*********************  Data RAM Assignments  ********************************
     0040 ;
     0041       ORG   08H     ; Dummy Origin
     0042 ;
0008 0043 RcvReg  RES   1      ; Data received
0001 0044
```

```
0009  0001 0045 XmtReg  RES    1                ; Data to be transmitted
000A  0001 0046 Count   RES    1                ; Counter for #of Bits Transmitted
000B  0001 0047 DlyCnt  RES    1
           0048 ;*****************************************************************************
           0049 ;
           0050         ORG    0
           0051 ;
0000 0068  0052 Talk    clrf   RcvReg           ; Clear all bits of RcvReg
0001 0625  0053         btfsc  Port_A,DR        ; check for a Start Bit
0002 0A30  0054         goto   User             ; delay for 104/2 uS
0003 0923  0055         call   Delay4           ; delay for 104+104/4
           0056 ;*****************************************************************************
           0057 ;       Receiver
           0058 ;
           0059 Rcvr
           0060         IF     R_NBit
0004 0C08  0061         movlw  8                ; 8 Data bits
           0062         ELSE
           0063         movlw  7                ; 7 data bits
           0064         ENDIF
           0065 ;
0005 002A  0066         movwf  Count
0006 0403  0067 R_next  bcf    STATUS,CARRY
           0068         IF     R_MODE
0007 0328  0069         rrf    RcvReg,Same      ; to set if MSB first or LSB first
           0070         ELSE
           0071         rlf    RcvReg,Same
           0072         ENDIF
0008 0625  0073         btfsc  Port_A,DR
           0074 ;
           0075         IF     R_MODE
           0076           IF     R_Nbit
0009 05E8  0077             bsf    RcvReg,MSB    ; Conditional Assembly
           0078           ELSE
           0079             bsf    RcvReg,MSB-1
           0080           ENDIF
           0081         ELSE
           0082         bsf    RcvReg,LSB
           0083         ENDIF
           0084 ;
000A 091F  0085         call   DelayY
000B 02EA  0086         decfsz Count,Same
000C 0A06  0087         goto   R_next
           0088 ;*****************************************************************************
000D 0208  0089 R_over  movf   RcvReg,0         ; Send back What is Just Received
000E 0029  0090         movwf  XmtReg
           0091 ;*****************************************************************************
           0092 ;       Transmitter
```

```
0093 ;
0094 Xmtr
0095                IF      X_Nbit
0096 000F 0C08      movlw   8
0097                ELSE
0098                movlw   7
0099                ENDIF
0100 0010 002A      movwf   Count
0101 ;
0102                IF      X_MODE
0103                ELSE
0104                  IF    X_Nbit
0105                  ELSE
0106                  rlf   XmtReg,Same
0107                  ENDIF
0108                ENDIF
0109 ;
0110 0011 0405      bcf     Port_A,DX       ; Send Start Bit
0111 0012 0925      call    Delay1
0112 0013 0403 X_next bcf   STATUS,CARRY
0113 ;
0114                IF      X_MODE          ; Conditional Assembly
0115 0014 0329      rrf     XmtReg,Same     ; to set if MSB first or LSB first
0116                ELSE
0117                rlf     XmtReg,Same
0118                ENDIF
0119 ;
0120 0015 0603      btfsc   STATUS,CARRY
0121 0016 0505      bsf     Port_A,DX
0122 0017 0703      btfss   STATUS,CARRY
0123 0018 0405      bcf     Port_A,DX       ; Send Stop Bit
0124 0019 0921      call    DelayX
0125 001A 02EA      decfsz  Count,Same
0126 001B 0A13      goto    X_next
0127 001C 0505      bsf     Port_A,DX
0128 001D 0925      call    Delay1
0129 ;
0130                IF      Sbit2
0131                bsf     Port_A,DX
0132                call    Delay1
0133                ENDIF
0134 ;
0135 001E 0A00      goto    Talk            ; Back To Reception & Transmision
0136 ;
0137 ;  End of Transmission
0138 ;
0139 DelayY  movlw  BAUD_Y
0140 001F 0C42      goto    save
0020 0A28
```

```
0021 0C42          0141 DelayX  movlw   BAUD_X
0022 0A28          0142         goto    save
0023 0C56          0143 Delay4  movlw   BAUD_4
0024 0A28          0144         goto    save
0025 0C44          0145 Delay1  movlw   BAUD_1          ; 104 uS for 9600 baud
0026 0A28          0146         goto    save
0027 0C43          0147 Delay2  movlw   BAUD_2
0028 002B          0148 save    movwf   DlyCnt
0029 02EB          0149 redo_1  decfsz  DlyCnt,Same
002A 0A29          0150         goto    redo_1
002B 0800          0151         retlw   0
                   0152 ;
002C 0C0E          0153 main    movlw   0EH             ; Bit 0 of Port A is Output
002D 0005          0154         tris    Port_A          ; Set Port_A.0 as output ( DX )
002E 0525          0155         bsf     Port_A,DR
                   0156 ;
002F 0A00          0157         goto    Talk
                   0158 ;
                   0159 ;
0030 0C22          0160 User    movlw   BAUD_3
0031 002B          0161         movwf   DlyCnt
0032 02EB          0162 redo_2  decfsz  DlyCnt,Same
0033 0A32          0163         goto    redo_2
0034 0A00          0164         goto    Talk            ; Loop Until Start Bit Found
                   0165 ;
                   0166 ;
                   0167         ORG     PIC54
01FF 0A2C          0168         goto    main
                   0169 ;
                   0170         END
                   0171
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX-----------

0040 : ---------------- ---------------- ---------------- ----------------

0180 : ---------------- ---------------- ---------------- ----------------

01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.
Errors   :   0
Warnings :   0
```

APPENDIX B: ASSEMBLY LANGUAGE LISTING FOR FULL DUPLEX

```
MPASM 01.00.02 Alpha  C:\AP-NOTES\   6-24-1994  9:8:43                PAGE  1
RS232 Communication Using PIC16C54

LOC  OBJECT CODE     LINE  SOURCE TEXT

                     0001 ;*************************************************************************
                     0002     TITLE   "RS232 Communication Using PIC16C54"
                     0003 ;
                     0004 ;   Comments :
                     0005 ;                (1) Full Duplex
                     0006 ;                (2) Tested from 1200 to 9600 Baud( @ 8 Mhz )
                     0007 ;                (3) Tested from 1200 to 19200 Baud(@ 16 & 20 Mhz)
                     0008 ;
                     0009 ;          The User gets a total time as specified by the User Cycles
                     0010 ;          in the table ( or from equations ). The user routine has to
                     0011 ;          exactly use up this amount of time. After this time the User
                     0012 ;          routine has to give up the control to the Operating System.
                     0013 ;          If less than 52 uS is used, then the user should wait in a
                     0014 ;          delay loop, until exactly 52 uS.
                     0015 ;
                     0016 ;          Transmission :
                     0017 ;                Transmit Data is output on DX pin (Bit DX of Port_A).
                     0018 ;                In the user routine, the user should load the
                     0019 ;                data to be transmitted in the XmtReg and Set the
                     0020 ;                X_flag ( bsf FlagRX,X_flag ). This flag gets cleared
                     0021 ;                after the transmission.
                     0022 ;
                     0023 ;          Reception :
                     0024 ;                Data is received on pin DR ( bit DR of Port_A ).
                     0025 ;                The User should constantly check the "R_done" flag
                     0026 ;                to see if reception is over. If the reception is
                     0027 ;                in progress, R_flag is set to 1.
                     0028 ;                If the reception is over, "R_done" flag is set to 1.
                     0029 ;                The "R_done" flag gets reset to zero when a next start
                     0030 ;                bit is detected. So, the user should constantly check
                     0031 ;                the R_done flag, and if SET, then the received word
                     0032 ;                is in Register "RcvReg". This register gets cleared
                     0033 ;                when a new start bit is detected.
                     0034 ;
                     0035 ;          Program Memory :
                     0036 ;             Total Program Memory Locations Used ( except initialization
                     0037 ;             in "main" & User routine ) = 132 locations.
                     0038 ;
                     0039 ;          Data Memory :
                     0040 ;             Total Data memory locations (file registers used) = 6
                     0041 ;                   2 File registers to hold Xmt Data & Rcv Data
                     0042 ;                   1 File registers for Xmt/Rcv flag test bits
```

```
0043 ;                  3 File registers for delay count & scratch pad
0044 ;
0045 ;          Stack :
0046 ;             Only one level of stack is used in the Operating System/RS232
0047 ;             routine. But this is freed as soon as the program returns to the
0048 ;             user routine.
0049 ;
0050 ;          RTCC :    Not Used
0051 ;          WDT  :    Not Used
0052 ;
0053                LIST        P=16C54, T=ON
0054           INCLUDE     "PICREG.H"
0001 ;*****************************  PIC16C5X Header    *****************************
0002 PIC54   equ     1FFH    ; Define Reset Vectors
0003 PIC55   equ     1FFH
0004 PIC56   equ     3FFH
0005 PIC57   equ     7FFH
0006 ;
0007 RTCC    equ     1
0008 PC      equ     2
0009 STATUS  equ     3       ; F3 Reg is STATUS Reg.
0010 FSR     equ     4
0011 ;
0012 Port_A  equ     5
0013 Port_B  equ     6       ; I/O Port Assignments
0014 Port_C  equ     7
0015 ;
0016 ;*****************************************************************************
0017 ;
0018 ;                       ; STATUS REG. Bits
0019 CARRY   equ     0       ; Carry Bit is Bit.0 of F3
0020 C       equ     0
0021 DCARRY  equ     1
0022 DC      equ     1
0023 Z_bit   equ     2       ; Bit 2 of F3 is Zero Bit
0024 Z       equ     2
0025 P_DOWN  equ     3
0026 PD      equ     3
0027 T_OUT   equ     4
0028 TO      equ     4
0029 PA0     equ     5
0030 PA1     equ     6
0031 PA2     equ     7
0032 ;
0033 Same    equ     1
0034 W       equ     0
0035 ;
0036 LSB     equ     0
```

```
0007                    0037 MSB     equ    7
                        0038 ;
0001                    0039 TRUE    equ    1
0001                    0040 YES     equ    1
0000                    0041 FALSE   equ    0
0000                    0042 NO      equ    0
                        0043 ;
                        0044 ;*****************************************************************
                        0045
                        0054          INCLUDE     "RS232.H"
                        0055 ;*****************************************************************
                        0001 ;                    RS232 Communication Parameters
                        0002 ;
                        0003 ;
                        0004 ;
0001                    0005 X_MODE  equ    1      ; If ( X_MODE==1) Then transmit LSB first
                        0006 ;                       if ( X_MODE==0) Then transmit MSB first ( CODEC like )
0001                    0007 R_MODE  equ    1      ; If ( R_MODE==1) Then receive LSB first
                        0008 ;                       if ( R_MODE==0) Then receive MSB first ( CODEC like )
0001                    0009 X_Nbit  equ    1      ; if ( X_Nbit==1) # of data bits ( Transmission ) is 8 else 7
0001                    0010 R_Nbit  equ    1      ; if ( R_Nbit==1) # of data bits ( Reception ) is 8 else 7
                        0011 ;
0000                    0012 SB2     equ    0      ; if SB2 = 0 then 1 Stop Bit
                        0013 ;                     ; if SB2 = 1 then 2 Stop Bit
                        0014 ;*****************************************************************
                        0015 ;       Transmit & Receive Test Bit Assignments
                        0016 ;
0000                    0017 X_flag  equ    0      ; Bit 0 of FlagRX
0002                    0018 R_flag  equ    2      ; Bit 1 of FlagRX
0003                    0019 S_flag  equ    3      ; Bit 2 of FlagRX
0004                    0020 BitXsb  equ    4      ; Bit 3 of FlagRX
0005                    0021 A_flag  equ    5
0006                    0022 S_bit   equ    6      ; Xmt Stop Bit Flag( for 2/1 Stop bits )
                        0023 ;
0001                    0024 R_done  equ    1      ; When Reception complete, this bit is SET
0000                    0025 X_done  equ    X_flag ; When Xmission complete, this bit is Cleared
                        0026 ;
0000                    0027 DX      equ    0      ; Transmit Pin ( Bit 0 of Port A )
0001                    0028 DR      equ    1      ; Recive Pin ( Bit 1 of Port A )
                        0029 ;
                        0030 ;*****************************  Data RAM Assignments  ******************************
                        0031 ;
                        0032          ORG    08H    ; Dummy Origin
                        0033 ;
0001                    0034 RcvReg  RES    1      ; Data received
0001                    0035 XmtReg  RES    1      ; Data to be transmitted
0001                    0036 Xcount  RES    1      ; Counter for #of Bits Transmitted
0001                    0037 Rcount  RES    1      ; Counter for #of Bits to be Received
```

```
000C  0001  0038 DlyCnt  RES    1      ; Counter for Delay constant
000D  0001  0039 FlagRX  RES    1      ; Transmit & Receive test flag hold register
            0040 ;_____________________________________________________________________
            0041 ;   Constants      19200   9600    4800    2400    1200
            0042 ; ( @ 20 Mhz )
            0043 ;_____________________________________________________________________
            0044 ;       K0           0      13      57     143     317*
            0045 ;       K1          49      98     184     358*    705*
            0046 ;       K2          34      60     103     191     364*
            0047 ;       K3          27      53      96     184     357*
            0048 ;       K4          29      55      98     186     359*
            0049 ;       K5          30      56      99     187     360*
            0050 ;       K6           0       0       0       0       0
            0051 ;       K7          56     104     190     365*    712*
            0052 ;
            0053 ;    User Cycles   118     260     521    1042    2083
            0054 ; ***************************************************************
            0055 ;
            0056 ;
            0057 ;_____________________________________________________________________
            0058 ;   Constants      19200   9600    4800    2400    1200
            0059 ;   ( @ 8 Mhz )
            0060 ;_____________________________________________________________________
            0061 ;       K0           -       0       5      39     109
            0062 ;       K1           -      39      80     150     288*
            0063 ;       K2           -      27      51      86     155
            0064 ;       K3           -      21      44      80     148
            0065 ;       K4           -      23      46      82     150
            0066 ;       K5           -      24      47      83     151
            0067 ;       K6           -       0       0       0       0
            0068 ;       K7           -      45      86     156     295*
            0069 ;
            0070 ;    User_Cycles     -      86     208     416     832
            0071 ; ***************************************************************
            0072 ;
            0073 ;_____________________________________________________________________
            0074 ;   Constants      19200   9600    4800    2400    1200
            0075 ;   ( @ 4 Mhz )
            0076 ;_____________________________________________________________________
            0077 ;       K0           -       -       0       5      39
            0078 ;       K1           -       -      39      80     150
            0079 ;       K2           -       -      27      51      86
            0080 ;       K3           -       -      21      44      80
            0081 ;       K4           -       -      23      46      82
            0082 ;       K5           -       -      24      47      83
            0083 ;       K6           -       -       0       0       0
            0084 ;       K7           -       -      45      86     156
            0085 ;
```

```
0086 ;   User_Cycles      =      86     208     416
0087 ;*****************************************************************
0088 ;
0089 ;
0090 ; The constants marked " * " are >255. To implement these constants
0091 ; in delay loops, the delay loop should be broken into 2 or more loops.
0092 ; For example, 357 = 255+102. So 2 delay loops, one with 255 and
0093 ; the other with 102 may be used.
0094 ;*****************************************************************
0095 ;       Set Delay Constants for 9600 Baud @ CLKIN = 8 Mhz
0096 ;
0097 K0      EQU     .0
0098 K1      EQU     .39
0099 K2      EQU     .27
0100 K3      EQU     .21
0101 K4      EQU     .23
0102 K5      EQU     .24
0103 K6      EQU     .0
0104 K7      EQU     .45
0105 ;
0106 ;*****************************************************************
0107
0055
0056 ;
0057         ORG     0
0058 ;*****************************************************************
0059 ;
0000 0C01 0060 Delay   movlw   K0+1
0001 002C 0061         movwf   DlyCnt          ; Total Delay = 3K+6
0002 02EC 0062 redo    decfsz  DlyCnt,Same
0003 0A02 0063         goto    redo
0004 0800 0064         retlw   0
0065 ;
0005 002C 0066 Delay1  movwf   DlyCnt
0006 02EC 0067 redo_1  decfsz  DlyCnt,Same     ;
0007 0A06 0068         goto    redo_1
0008 0A8D 0069         goto    User
0070 ;
0009 002C 0071 Delay2  movwf   DlyCnt
000A 02EC 0072 redo_2  decfsz  DlyCnt,Same     ; Delay =   = 260 Cycles
000B 0A0A 0073         goto    redo_2
000C 0A67 0074         goto    User_1
0075 ;
000D 0625 0076 R_strt  btfsc   Port_A,DR       ; check for a Start Bit
000E 0A17 0077         goto    Shelly          ; delay for 104/2 uS
000F 042D 0078         bcf     FlagRX,R_done   ; Reset Receive done flag
0010 054D 0079         bsf     FlagRX,R_flag   ; Set flag for Reception in Progress
0011 078D 0080         btfss   FlagRX,BitXsb
```

```
0081                    bsf     FlagRX,A_flag   ; A_flag is for start bit detected in R_strt
0082                    clrf    RcvReg          ; Clear all bits of RcvReg
0083                    IF      R_Nbit
0084 0C08               movlw   8               ; 8 Data bits
0085                    ELSE
0086                    movlw   7               ; 7 data bits
0087                    ENDIF
0088                    movwf   Rcount
0089                    goto    Shell           ; delay for 104+104/4
0090                ;
0091 ShellY     btfss   FlagRX,BitXsb
0092                    goto    Shell
0093                    bsf     FlagRX,R_flag
0094                    goto    Shell
0095                ;
0096 R_next     bcf     STATUS,CARRY
0097                    IF      R_MODE
0098                    rrf     RcvReg,Same     ; to set if MSB first or LSB first
0099                    ELSE
0100                    rlf     RcvReg,Same
0101                    ENDIF
0102                    btfsc   Port_A,DR
0103                    IF      R_MODE
0104                        IF      R_Nbit
0105                        bsf     RcvReg,MSB  ; Conditional Assembly
0106                        ELSE
0107                        bsf     RcvReg,MSB-1
0108                        ENDIF
0109                    ELSE
0110                    bsf     RcvReg,LSB
0111                    ENDIF
0112                    decfsz  Rcount,Same
0113                    goto    Shell
0114                    bcf     FlagRX,R_flag
0115                    bsf     FlagRX,S_flag
0116                    bsf     FlagRX,R_done
0117                    goto    Shell
0118                ;
0119                ;       Reception Done
0120                ;
0121 X_strt     bcf     Port_A,DX       ; Send Start Bit
0122                    IF      X_Nbit
0123 0C08               movlw   8
0124                    ELSE
0125                    movlw   7
0126                    ENDIF
0127                    movwf   Xcount
0128                    IF      X_MODE

0012 05AD
0013 0068

0014 0C08

0015 002B
0016 0A78

0017 078D
0018 0A78
0019 054D
001A 0A78

001B 0403

001C 0328

001D 0625

001E 05E8

001F 02EB
0020 0A78
0021 044D
0022 056D
0023 052D
0024 0A78

0025 0405

0026 0C08

0027 002A
```

```
                        ELSE
                          IF      X_Nbit
                          ELSE
                            rlf   XmtReg,Same
                          ENDIF
                        ENDIF
0028 0A50               goto    X_SB
                        ;
0029 0403 X_next        bcf     STATUS,CARRY
                        IF      X_MODE
002A 0329               rrf     XmtReg,Same     ; Conditional Assembly
                        ELSE                    ; to set if MSB first or LSB first
                        rlf     XmtReg,Same
                        ENDIF
002B 0603               btfsc   STATUS,CARRY
002C 0505               bsf     Port_A,DX
002D 0703               btfss   STATUS,CARRY
002E 0405               bcf     Port_A,DX
002F 00EA               decf    Xcount,Same
0030 0A52               goto    X_Data
                        ;
0031 040D X_SB_1        bcf     FlagRX,X_flag   ; Xmt flag = 0 - transmission over
0032 0C09               movlw   9
0033 002A               movwf   Xcount
0034 0505               bsf     Port_A,DX       ; Send Stop Bit
0035 0A60               goto    X_Stop
                        ;
0036 0505 X_SB_2        bsf     Port_A,DX
0037 04CD               bcf     FlagRX,S_bit
0038 0A60               goto    X_Stop
                        ;
                        ;  End of Transmission
                        ;
0039 076D R0_X0         btfss   FlagRX,S_flag
003A 0A8D               goto    User
003B 046D               bcf     FlagRX,S_flag
003C 0900               call    Delay
003D 0C2E               movlw   K7+1
003E 0A05               goto    Delay1
                        ;
003F 0900 R1_X0         call    Delay           ; delay for 1st bit is 104+104/4
0040 0C28               movlw   K1+1
0041 002C               movwf   DlyCnt
                        IF      R_Nbit
0042 0C08               movlw   8               ; 8 Data bits
                        ELSE
                        movlw   7               ; 7 data bits
```

```
                        ENDIF
                        xorwf   Rcount,W
                        btfsc   STATUS,Z_bit
                        goto    redo_1
                        movlw   K2+1
                        goto    Delay1
                        ;
                        R1_X1                           ; same as R0_X1
                        R0_X1   movlw   9
                        subwf   Xcount,W
                        btfsc   STATUS,Z_bit
                        goto    X_strt
                        movf    Xcount,Same             ; to check if All data bits Xmted
                        btfss   STATUS,Z_bit
                        goto    X_next
                        IF      SB2
                          btfsc FlagRX,S_bit
                            goto X_SB_2
                          bsf   FlagRX,S_bit
                            goto X_SB_1
                        ELSE
                          goto  X_SB_1
                        ENDIF
                        ;
                        ;
                        X_SB    goto    cycle4
                        cycle4  goto    X_Data
                        ;
                        X_Data  btfsc   FlagRX,A_flag
                        goto    SbDly
                        btfsc   FlagRX,BitXsb
                        goto    ABC
                        call    Delay
                        movlw   K3+1
                        goto    Delay2
                        ;
                        SbDly   bcf     FlagRX,A_flag
                        call    Delay
                        movlw   K4+1
                        goto    Delay2
                        ;
                        ABC     bcf     FlagRX,BitXsb
                        call    Delay
                        goto    User_1
                        ;
                        X_Stop
                        btfsc   FlagRX,A_flag
                        goto    SbDly
```

```
0062 068D          btfsc   FlagRX,BitXsb
0063 0A5D          goto    ABC
0064 0900          call    Delay
0065 0C19          movlw   K5+1
0066 0A09          goto    Delay2
                   ;
0067 064D User_1   btfsc   FlagRX,R_flag          ; Reception already in progress
0068 0A77          goto    Sync_1
0069 066D          btfsc   FlagRX,S_flag
006A 0A74          goto    Sync_3
006B 0625          btfsc   Port_A,DR            ; check for a Start Bit
006C 0A77          goto    Sync_2               ; No Start Bit - goto User routine
006D 042D          bcf     FlagRX,R_done        ; Reset Receive done flag
006E 044D          bcf     FlagRX,R_flag
006F 058D          bsf     FlagRX,BitXsb        ; Set flag for Reception in Progress
0070 0068          clrf    RcvReg               ; Clear all bits of RcvReg
0071 0C08          IF      R_Nbit
0071 0C08          movlw   8                    ; 8 Data bits
0072 002B          ELSE
0073 0A8D          movlw   7                    ; 7 data bits
0074 046D          ENDIF
0075 0C01          movwf   Rcount
0076 0A05          goto    User
                   ;
0077 0A8D Sync_3   bcf     FlagRX,S_flag
0078 064D          movlw   K6+1
0079 0A7D          goto    Delay1
                   ;
007A 060D Sync_1
007B 0A48 Sync_2   goto    User
007C 0A39 ;*****************************************************************
007D 060D ;
007E 0A48 ;*****************************************************************
007F 0A3F ;
                   ;    Operating System
                   ;  The User routine after time = B/2, should branch Here
                   ;
0080 074D Op_Sys   btfss   FlagRX,R_flag
0081 0A82          goto    Chek_X
0082 0A7D          goto    Chek_X
0083 060D          btfsc   FlagRX,X_flag
0084 0A48          goto    R0_X1
0085 0A39          goto    R0_X0                ; Case for R0_X0
0086 060D Chek_X   btfsc   FlagRX,X_flag
0087 0A48          goto    R1_X1
0088 0A3F          goto    R1_X0
                   ;
                   ;*****************************************************************
                   ;
                   ;    Operating System
                   ;  The User routine after time = B/2, should branch Here
                   ;
0089 074D Op_Sys   btfss   FlagRX,R_flag
```

```
0081 0A0D          goto    R_strt
0082 0A1B          goto    R_next
0273 ;
0274 ;*****************************************************************************
0275 ;
0276 ;
0277 ;
0278 main    movlw   0EH             ; Bit 0 of Port A is Output
0279         tris    Port_A          ; Set Port_A.0 as output ( DX )
0280 ;                                & Port_A.1 is input ( DR )
0281         bsf     Port_A,DX
0282         movlw   9
0283         movwf   Xcount          ; If Xcount == 9, Then send start bit
0284         clrf    FlagRX          ; Clear All flag bits.
0285         IF      SB2
0286           bsf   FlagRX,S_bit    ; Set Xmt Stop bit flag(2 Stop Bits)
0287         ELSE
0288           bcf   FlagRX,S_bit    ; Clear Xmt Stop bit flag
0289         ENDIF
0290         movlw   1FH             ; Prescaler = 4
0291         OPTION                  ; Set RTCC increment on internal Clock
0292         goto    Op_Sys
0293 ;
0294 ;*****************************************************************************
0295 ;
0296 ;       **********************  User Routine  ***************
0297 ;  The User routine should use up time exactly = User time as given
0298 ;  in the Constants Table ( or by Equations for constants ).
0299 ;  At 9600, this 86 Clock Cycles. RTCC timer is used here to count
0300 ;  upto 86 cycles ( From 128-86 To 0 ) by examining Bit 7 of RTCC.
0301 ;
0030 K_user  equ     .128+.6-.86
0303 ;
0304 User    movlw   K_user
0305         movwf   RTCC
0306         btfsc   FlagRX,R_done
0307         goto    ErrChk
0308 SetXmt  btfsc   FlagRX,X_flag
0309         goto    Op
0310         movlw   41H
0311         movwf   XmtReg
0312         bsf     FlagRX,X_flag   ; Enable Xmission
0313         goto    Op
0314 ;
0315 ErrChk
0316         movlw   "Z"
0317         xorwf   RcvReg,W
0318         btfsc   STATUS,Z_bit
0319         goto    SetXmt
0320 error   goto    error           ; Received word is not "Z"
```

```
                        0321 ;
009C 07E1               0322 Op     btfss   RTCC,MSB        ; Test for RTCC bit 7
009D 0A9C               0323        goto    Op              ; If Set, Then RTCC has incremented
009E 0A80               0324 Oflow  goto    Op_Sys          ; to 128.
                        0325 ;
                        0326 ;*****************************************************************
                        0327 ;
                        0328        ORG     PIC54
01FF 0A83               0329        goto    main
                        0330
                        0331        END
                        0332
                        0333

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ________ ________
00C0 : ________ ________ ________ ________

0180 : ________ ________ ________ ________
01C0 : ________ ________ ________ _______X

All other memory blocks unused.

Errors   :   0
Warnings :   0
```

NOTES:

Using a PIC16C5X as a Smart I²C™ Peripheral

Author: Don Lekei - NII Norsat International Inc.

INTRODUCTION

The PIC16C5X microcontrollers from Microchip are ideally suited for use as smart peripheral devices under the control of the main processors in systems due to their low cost and high speed. They are capable of performing tasks which would simply overload a conventional microprocessor, or require considerable logic circuitry, at a cost competitive with lower mid-range PLDs. To minimize the engineering overhead of adding multiple controllers to a product, it is convenient for the auxiliary controllers to emulate standard I/O peripherals.

A common interface found in existing products is the I²C bus. This efficient, two-wire, bi-directional interface allows the designer to connect multiple devices together, with the microprocessor able to send data to and receive data from any device on the bus. This interface is found on a variety of components, such as PLLs, DACs, video controllers, and EEPROMs. If a product already contains one or more I²C devices, it is simple to add a PIC16C5X emulating a compatible component.

This application note describes the implementation of a standard slave device with multiple, bi-directional registers. A subset of the full I²C specification is supported, which can be controlled by the same software which would talk to a Microchip 24LCXX series EEPROM.

THE I²C BUS

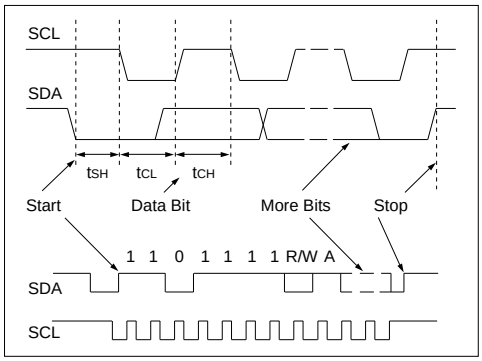
The I²C bus is a master-slave two-wire interface, consisting of a clock line (SCL) and a data line (SDA). Bi-directional communication (and in a full, multi-master system, collision detection and clock synchronization) is facilitated through the use of a wire-and (ie. active-low, passive high) connection.

The standard-mode I²C bus supports SCL clock frequency up to 100 KHz. The newly released fast-mode I²C bus supports clock rate up to 400 KHz. This application note will support 100 KHz (standard-mode) clock rate.

Each device has a unique seven bit address, which the master uses to access each individual slave device.

During normal communication, SDA is only permitted to change while SCL is low, thus providing two violation conditions (see Figure 1) which are used to signal a start condition (SDA drops while SCL is high) and a stop condition (SDA rises while SCL is high), which frame a message.

FIGURE 1 - I²C TIMING



Each byte of a transfer is 9-bits long (see timing chart in the program listing). The talker sends 8 data bits followed by a "1" bit. The listener acknowledges the receipt of the byte and permission to send the next byte by inserting a "0" bit over the trailing "1". The listener may indicate "not ready for data" by leaving the acknowledge bit as a "1".

The clock is generated by the master only. The slave device must respond to the master within the timing specifications of the I²C definition otherwise the master would be required to operate in slow mode, which most software implementations of I²C masters do not actually support. The specified (standard-mode) t_{CL} is 4.7 µs, and t_{CH} is only 4 µs, so it would be extremely difficult to achieve the timing of a hardware slave device with a conventional microcontroller.

MESSAGE FORMAT

A message is always initiated by the master, and begins with a start condition, followed by a slave address (7 MSBs) and direction bit (LSB = 1 for READ, 0 for WRITE). The addressed slave must acknowledge this byte if it is ready to communicate any data. If the slave fails to respond, the master should send a stop and retry.

If the direction bit is "0" the next byte is considered the sub-address (this is an extension to I²C used by most multi-register devices). The sub-address selects which "register" or function subsequent read or write operations will affect. Any additional bytes will be received and stored in consecutive locations until a stop is sent. If the slave is unable to process more data, it could terminate transfer by not acknowledging the last byte.

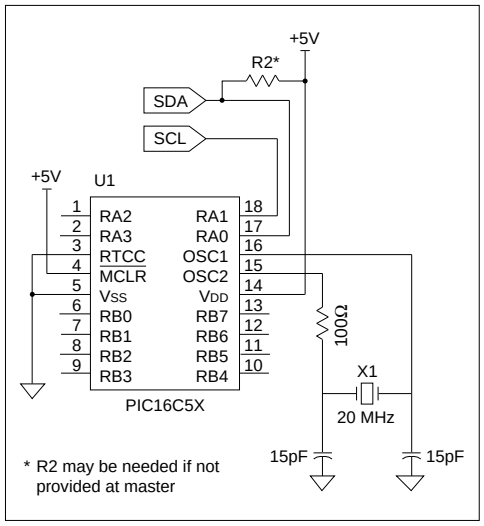
If the direction bit is "1", the slave will transfer successive bytes to the master (while the master holds the line at '1'), while the master acknowledges each byte with a "0" in the ninth bit. The master can terminate the transfer by not acknowledging the last byte, while the slave can stop the transfer by generating a stop condition.

The start address of a read operation is set by sending a write request with a sub-address only (no data bytes). For a detailed set of timing diagrams and different communication modes, consult any of the Microchip 24LCXX EEPROM specifications. This program communicates using the same formats.

IMPLEMENTATION

The chip will respond to slave address "DEVICE_ADDRESS", which by default is $D6_{16}$ ($D7_{16}$ for read). This address was chosen because it is the fourth optional address of a Philips PCF8573 clock/calender or a TDA8443 tipple video switch (unlikely that a product would contain four of those).

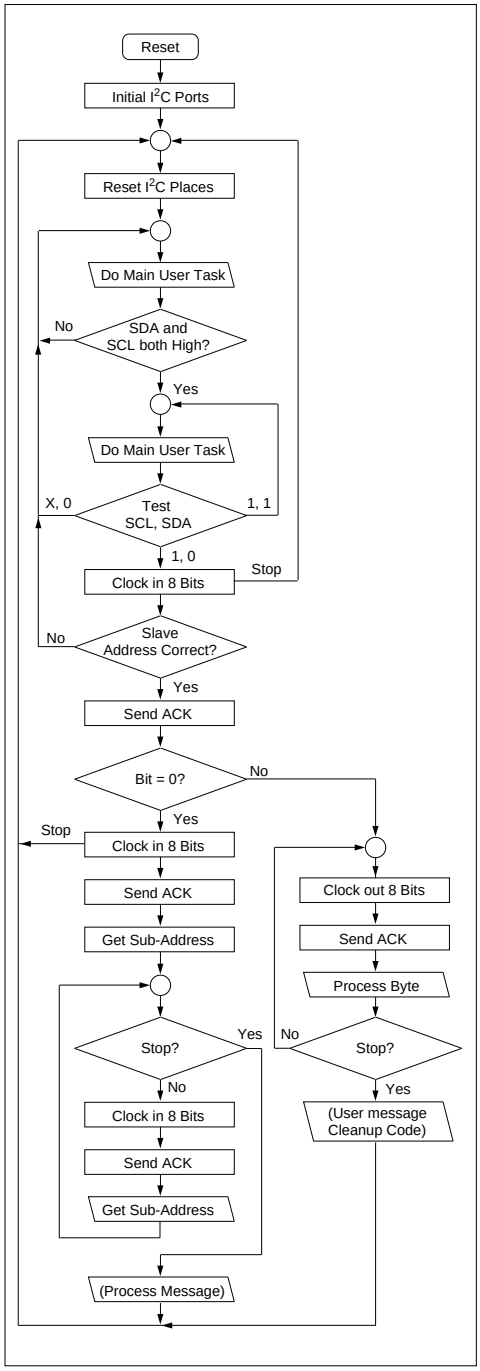
FIGURE 2 - SCHEMATIC OF I²C CONNECTIONS



The connections to the device are shown in Figure 2. The use of RA0 for data in is required. Data is shifted directly out of the port. The code could be modified to make it port independent, but the loss of efficiency may hinder some real-time applications.

This application emulates an I²C device with 8 registers, accessed as sub-addresses 1 through 8 (modulo 7), plus a data channel (0). The example code returns an ID string when the data channel is accessed. When bytes are written to sub-addresses other than 0, they are stored in I2CR0-I2CR7 (I2CR0 gets data written to sub-address 8).

FIGURE 3 - I²C DEVICE FLOWCHART



When the initial sub-address is 0, the flag B:ID is set. This is used to indicate access to a special channel. In this case, the data channel is used to return an ID message, or output data to port B, however the natural extension would be to use this as a data I/O channel.

To make the basic device routines easily adaptable to a variety of uses, macros are used to implement the application specific code. This allows the developer the option of using subroutine calls, or in-line code to avoid the 4 clock cycle overhead and use of the precious stack.

Macro	User code function
USER_MAIN	Code to execute in the main loop while not in a message. If this code takes too long, tSH of 4µs will be violated (see Fig. 1). The slave will simply miss the address, not acknowledge, and the master will retry.
USER_Q	This would be quick user code to implement real-time processes. In most applications, this macro would be empty. If used, this routine should be kept under 4µs if possible.
USER_MSG	This would be user code to process a message. It is inserted after a message is successfully received.
USER_RECV	This would be user code to process a received byte. It allows the user to add extra code to implement special purpose sub-addresses such as FIFOs.
USER_XMIT	This would be user code to prepare an output byte. In the default routine, it traps sub-address 0 and calls the ID string function.

References:

I²C Bus Specification, Phillips Components, December 1988.

The I²C bus and how to use it (including specification), Signetics/Philips Semiconductors, January 1992.

Fenger, Carl, "The Inter-Integrated Circuit (I2C) Serial Bus: Theory and Practical Consideration", *Application Note 168*, Philips Components, December 1988.

"24C16 16K CMOS Serial Electrically Erasable PROM", *Microchip Data Book (1992).*

About the Author:

Don Lekei has been designing microprocessor based products over 14 years. He has developed many software and hardware products for a wide variety of applications. Mr. Lekei is Manager of Advanced Technologies at NII Norsat International Inc. at their Canadian headquarters in Surrey, British Columbia. Norsat designs and manufactures products to receive broadcast communications from satellites, terrestrial broadcasting systems and optical fibre. Norsat develops technologies and products for satellite entertainment television, broadcast music and data networks.

APPENDIX A:

```
MPASM B0.54                                                    PAGE  1



                        LIST    P=16C54, C=80, N=0, R=DEC

0676                CPU     EQU     1654
0000                SIM     EQU     0               ;Change timing constants for simulator


                        IF      (CPU==1654) || (CPU==1655)
01FF                _RESVEC EQU     01FFH           ;16c54 start address
                        ENDIF

                        IF      CPU==1656
                    _RESVEC EQU     03FFH           ;16C56 start address
                        ENDIF

                        IF      CPU==1657
                    _RESVEC EQU     07FFH           ;16C57 start address
                        ENDIF

                    ;*** Reset Vector *****************************************


                        ORG     _RESVEC         ;
                    RESVEC                      ;
01FF 0A0B                   GOTO    INIT            ;

                    ;*********************************************************


                    ;*********************************************************

                    ;* Macros to set/clear/branch/skip on bits
                    ;* These macros define and use synthetic “bit labels”
                    ;* Bit labels contain the address and bit of a location
                    ;*
                    ;*********************************************************

                    ;*      Usage                   Description
                    ;*      ___________  _____________

                    ;*      BIT     label,bit,file  ;Define a bit label
                    ;*      SEB     label           ;set bit using bit label
                    ;*      CLB     label           ;clear bit using bit label

                    ;*      SKBS    label           ;SKIP on bit set
                    ;*      SKBC    label           ;SKIP on bit clear
                    ;*      BBS     label,address   ;BRANCH on bit set
                    ;*      BBC     label,address   ;BRANCH on bit clear
                    ;*      CBS     label,address   ;CALL on bit set
                    ;*      CBC     label,address   ;CALL on bit clear
                    ;*
                    ;*********************************************************


                    BIT     MACRO   label,bit,file  ;Define a bit label
                    label   EQU     file<<8|bit
                        ENDM                    ;

                    SEB     MACRO   label           ;Set bit
                        BSF     label>>8,label&7;(macro)
                        ENDM                    ;
```

```
                    CLB     MACRO   label                   ;Clear bit
                            BCF     label>>8,label&7        ;(macro)
                            ENDM                            ;

                    SKBS    MACRO   label                   ;Skip on bit set
                            BTFSS   label>>8,label&7        ;(macro)
                            ENDM

                    SKBC    MACRO   label                   ;Skip on bit clear
                            BTFSC   label>>8,label&7        ;(macro)
                            ENDM

                    BBS     MACRO   label,address           ;Branch on bit set
                            BTFSC   label>>8,label&7        ;(macro)
                            GOTO    address                 ;(macro)
                            ENDM                            ;

                    BBC     MACRO   label,address           ;Branch on bit clear
                            BTFSS   label>>8,label&7        ;(macro)
                            GOTO    address                 ;(macro)
                            ENDM

                    CBS     MACRO   label,address           ;Call on bit set
                            CALL    label>>8,label&7        ;(macro)
                            ENDM                            ;

                    CBC     MACRO   label,address           ;Call on bit clear
                            CALL    label>>8,label&7        ;(macro)
                            ENDM


                    ;For Assembler portability

0000                W       EQU     0                       ;For file,W
0000                w       EQU     0                       ;For file,W
0001                F       EQU     1                       ;For file,F
0001                f       EQU     1                       ;For file,F

                    ;*********************************************************

                    ;* REGISTER DECLARATIONS
                    ;*********************************************************


                            ORG     0                       ;ORG for register declaration


0000  0001          ind     RES     1                       ;0=pseudo-reg 0 for indirect

0001  0001          RTCC    RES     1                       ;1=real time counter
0002  0001          PC      RES     1                       ;2=PC
0003  0001          STATUS  RES     1                       ;3=status reg

                    ;* Status reg bits

                             BIT    B_C,0,STATUS            ;Carry
0300                B_C      EQU    STATUS<<8|0


                             BIT    B_DC,1,STATUS           ;Half carry
0301                B_DC     EQU    STATUS<<8|1


                             BIT    B_Z,2,STATUS            ;Zero
0302                B_Z      EQU    STATUS<<8|2
```

```
                        BIT     B_PD,3,STATUS   ;Power down
0303                B_PD    EQU     STATUS<<8|3


                        BIT     B_TO,4,STATUS   ;Timeout
0304                B_TO    EQU     STATUS<<8|4


                        BIT     B_PA0,5,STATUS  ;Page select (56/57 only)
0305                B_PA0   EQU     STATUS<<8|5


                        BIT     B_PA1,6,STATUS  ;Page select (56/57 only)
0306                B_PA1   EQU     STATUS<<8|6


                        BIT     B_PA2,7,STATUS  ;GP flag
0307                B_PA2   EQU     STATUS<<8|7



0004  0001          FSR     RES     1               ;4=file select reg 0-4=indirect address
0005  0001          PORTA   RES     1               ;5=port A I/O register (4 bits)
0006  0001          PORTB   RES     1               ;6=port B I/O register

                        IF      (CPU==1655)|||(CPU==1657)
                    PORTC   RES     1               ;7=I/O port C on 16C54/56 only
                        ENDIF

                                                    ;registers used by this code

0007  0001          I2CFLG  RES     1               ;I2C flag reg
                                                    ;—i2c flags————————————————————

                        BIT     B_RD,0,I2CFLG   ;Flag: 1=read
0700                B_RD    EQU     I2CFLG<<8|0


                        BIT     B_UA,1,I2CFLG   ;Flag: 0=reading unit address
0701                B_UA    EQU     I2CFLG<<8|1


                        BIT     B_SA,2,I2CFLG   ;Flag: 1=reading subabbress
0702                B_SA    EQU     I2CFLG<<8|2


                        BIT     B_ID,3,I2CFLG   ;Flag: 1=reading id
0703                B_ID    EQU     I2CFLG<<8|3


                    ;————————————————————————

0008  0001          I2CREG  RES     1               ;I2C I/O register
0009  0001          I2CSUBA RES     1               ;Subaddress
000A  0001          I2CBITS RES     1               ;I2C xmit bit counter


                    ;*********************************************************
                    ;* 8 Pseudo registers accessed by sub-addresses 1-8
                    ;* (address 0 accesses the ID string)
                    ;* these are read-write registers
                    ;*********************************************************



000B                I2CR0   EQU     $               ;Sub-address 8
000B  0001                  RES     1               ;8 pseudo registers

000C                I2CR1   EQU     $               ;Sub-address 1
```

```
000C  0001                    RES     1

000D                  I2CR2   EQU     $               ;Sub-address 2
000D  0001                    RES     1

000E                  I2CR3   EQU     $               ;Sub-address 3
000E  0001                    RES     1

000F                  I2CR4   EQU     $               ;Sub-address 4
000F  0001                    RES     1

0010                  I2CR5   EQU     $               ;Sub-address 5
0010  0001                    RES     1

0011                  I2CR6   EQU     $               ;Sub-address 6
0011  0001                    RES     1

0012                  I2CR7   EQU     $               ;Sub-address 7
0012  0001                    RES     1

                      ;Constants used by program

00D6                  DEVICE_ADDRESS EQU      0D6H    ;I2C device address


                      ;*******************************************************
                      ;
                      ;** PORTA DEFINITIONS
                      ;** I2C interface uses PORTA
                      ;** note SDA goes to A0 for code efficiency
                      ;**
                      ;*******************************************************



00F7                  TAREAD   EQU    B'11110111'     ;TRISA register for SDA rea

00F6                  TAWRITE  EQU    B'11110110'     ;TRISA register for SDA wri

00F7                  TAINIT   EQU    TAREAD          ;Initial TRISA value

                               BIT    B_SDA,0,PORTA   ;I2C SDA (data) This must be bit 0!
0500                  B_SDA    EQU    PORTA<<8|0


                               BIT    B_SCL,1,PORTA   ;I2C SCL (clock)
0501                  B_SCL    EQU    PORTA<<8|1


                      ;spare           B_???,2,PORTA   ;not used
                      ;spare           B_???,3,PORTA   ;not used

                      ;*******************************************************
                      ;**
                      ;** Port B definition (Parallel out)
                      ;**
                      ;*******************************************************

0000                  TBINIT  EQU     B'00000000'     ;Port B tris (all output)
00FF                  PBINIT  EQU     B'11111111'     ;Port B init


                      ;*******************************************************
                      ;
                      ;* Macros to contain user POLL loop code.
                      ;* These are implimented as macros to allow ease of modification,
                      ;* especially in real-time applications. The functions could be coded as
                      ;* in-line code or as subroutines depending on ROM/time tradeoffs.
                      ;*
                      ;* USER_MAIN:   Decision or code to perform at idle time
                      ;*
```

```
                        ;* USER_Q:      'Quick' code for use during transfer - max
                        ;*              I˝C Spec. More than 4 Ês may result in I˝C
                        ;*              full spec speed.
                        ;*
                        ;* USER_MSG:    Code to execute at receipt of I˝C command.
                        ;*
                        ;*************************************************************

                        USER_MAIN       MACRO
                        ;*** This would be user code for idle loop
                                        ENDM

                        USER_Q          MACRO
                        ;*** This would be quick user code
                                        ENDM

                        USER_MSG        MACRO
                        ;*** This would be user code to process a message
                                        ENDM

                        USER_RECV       MACRO
                        ;*** This would be user code to process a received byte
                        ;*** example code sends sub-address 0 to port b
                                       BBC     B_ID,_NXI_notid              ;Channel 0! Bit set if

                                       MOVFW   I2CREG                       ;get received byte
                                       MOVWF   PORTB                        ;and write it on portb
                                       GOTO    IN_CONT
                        _NXI_notid
                                       ENDM

                        USER_XMIT       MACRO
                        ;*** This would be user code to prepare an output byte
                        ;*** example code sends id string to output
                                        BBC     B_ID,_NXO_notid              ;Channel 0! Bit set if

                                        CALL    GETID                        ;get next byte from ID
                                        GOTO    OUT_CONT                     ;and send it
                        _NXO_notid
                                        ENDM

                        ;*************************************************************
                        ; START OF CODE
                        ;*************************************************************
                                ORG     0
                        ;*************************************************************

                        ;* Device ID Table (must be at start)
                        ;* TABLE FOR UNIT ID returns next char in W
                        ;*************************************************************

                        GETID
0000 0209                       MOVFW   I2CSUBA                              ;W=I2CSUBA
0001 0E07                       ANDLW   07H                                  ;Limit to 8 locations
0002 01E2                       ADDWF   PC,F

                        ;*************************************************************

                        ;* Device ID text: read starting at sub-address 0
                        ;*************************************************************


0003 0850                       RETLW   'P'
0004 0849                       RETLW   'I'
0005 0843                       RETLW   'C'
0006 0849                       RETLW   'I'
0007 0832                       RETLW   '2'
0008 0843                       RETLW   'C'
0009 0800                       RETLW   0
```

```
000A 0800                   RETLW   0

                    ;*******************************************************
                    ;* I2C Device routines
                    ;*
                    ;* Enable must be HIGH, else state goes to 0
                    ;* write is to me, read is from me.
                    ;*
                    ;*            <============== first byte / subsequant write
                    ;*
                    ;* SDA   -|   X--X--X--X--X--X--X--X-
                    ;*
                    ;*         |-X--X--X--X--X--X--X--X-
                    ;* (bit)   s    7     6     5     4     3     2     1     0
                    ;* SCL  --|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |
                    ;*           |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
                    ;*
                    ;* STATE: 0 1 2  3  2  3  2  3  2  3  2  3  2  3  2  3  2
                    ;*
                    ;*            <============== subsequant reads ============
                    ;*
                    ;* SDA     X--X--X--X--X--X--X--X--X
                    ;*        -X--X--X--X--X--X--X--X--X
                    ;* (bit)ack   7     6     5     4     3     2     1     0
                    ;* SCL  -|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
                    ;*        |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |
                    ;*
                    ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8
                    ;*
                    ;*            <============== Final READ ==================
                    ;*
                    ;* SDA     X--X--X--X--X--X--X--X--X
                    ;*        -X--X--X--X--X--X--X--X--X
                    ;* (bit)ack   7     6     5     4     3     2     1     0
                    ;* SCL  -|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|
                    ;*        |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |-|  |
                    ;*
                    ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8
                    ;*
                    ;* STATE B is an ignore bit state for non-addressed bits
                    ;* STATE C indicates last sample had ENA low
                    ;* on rising edge of ENA, DATA LOW = low voltage, DATA&CLOC
                    ;*******************************************************
                    ;I2C interface uses PORTA
                    ;note SDA must be on PORTA,0 for code efficiency
                    ;*******************************************************
                    ;** INIT
                    ;** Hardware reset entry point
                    ;**
                    ;*******************************************************
                    INIT    ;Power-on entry
                    ;*******************************************************
                    ;** RESET
                    ;** software reset entry point
                    ;**
                    ;*******************************************************
                    RESET                                   ;Soft reset

000B 0CF7                   OVLW   TAINIT                   ;Init ports
000C 0005                   TRIS    PORTA
000D 0C00                   MOVLW   TBINIT
000E 0006                   TRIS    PORTB
000F 0CFF                   MOVLW   PBINIT
0010 0026                   MOVWF   PORTB
                    ;*****************************************************
                    ; Main wait loop while idle. POLL loop should be called here
                    ;
```

```
                    ;*****************************************************
                    I2CWAIT
0011 0004                   CLRWDT                          ;Clear watchdog timer
                            CLB     B_UA                    ;Init state flags
0012 0427                   BCF     B_UA>>8,B_UA&7
                            CLB     B_SA                    ;Init state flags
0013 0447                   BCF     B_SA>>8,B_SA&7
                            CLB     B_RD                    ;Init state flags
0014 0407                   BCF     B_RD>>8,B_RD&7
                    loop1
0015 0004                  CLRWDT                           ;Clear watchdog timer
                    USER_MAIN                               ;Call user code while in idle state
                    ;*** This would be user code for idle loop
                            SKBC    B_SDA                   ;Wait for SDA&SCL=H
0016 0605                   BTFSC   B_SDA>>8,B_SDA&7
                    loop2
                            SKBS    B_SCL                   ;
0017 0725                   BTFSS   B_SCL>>8,B_SCL&7
0018 0A15                   GOTO    loop1                   ; No longer valid to wait f
0019 0004                   CLRWDT                          ;Clear watchdog timer
                            USER_MAIN                       ;Call user code while in idle state
                    ;*** This would be user code for idle loop
                    ;** wait for start **
                            SKBC    B_SCL                   ;Clock has dropped
001A 0625                   BTFSC   B_SCL>>8,B_SCL&7
                            SKBC    B_SDA                   ;Data dropped... Start!
001B 0605                   BTFSC   B_SDA>>8,B_SDA&7
001C 0A17                   GOTO    loop2
                    ;** START RECEIVED! — wait for first bit!
                    loop3
                            BBS     B_SDA,I2CWAIT           ;Data raised before clock dropped —
001D 0605                   BTFSC   B_SDA>>8,B_SDA&7
001E 0A11                   GOTO    I2CWAIT
                            BBS     B_SCL,loop3             ;Wait for clock low
001F 0625                   BTFSC   B_SCL>>8,B_SCL&7
0020 0A1D                   GOTO    loop3
                    NEXTBYTE
0021 0004                   CLRWDT                          ;Clear watchdog timer
0022 0C01                   MOVLW   1                       ;Init receive byte so bit f
0023 0028                   MOVWF   I2CREG
                    ;** Shift bits! — external poll may be executed during low
                    ;* ENABLE line is checked for loss of enable ONLY during HI
                    ;*** CLOCK IS LOW — DATA MAY CHANGE HERE
                    ;*** We have at least 4 Ês before any change can occur
                    loop4
                            USER_Q
                    ;*** This would be quick user code
                    loop4A
                           BBC      B_SCL,loop4A            ;Wait for clock high
0024 0725                  BTFSS    B_SCL>>8,B_SCL&7
0025 0A24                  GOTO     loop4A
                    ;*** CLOCK IS HIGH — SHIFT BIT - then watch for change
0026 0305                  RRF     PORTA,W                 ;Move RA0 into C
0027 0368                  RLF     I2CREG,F                ;Shift in bit
0028 0603                  SKPNC                           ;Skip if not done
0029 0A36                  GOTO    ACK_I2C                 ;Acknowledge byte
002A 0608                  BTFSC   I2CREG,0                ;Skip if data bit was 0
002B 0A31                  GOTO    ii_1                    ;This bit was set
                    ii_0
                           BBC      B_SCL,loop4            ;Wait for clock  low
002C 0725                  BTFSS    B_SCL>>8,B_SCL&7
002D 0A24                  GOTO     loop4
                           SKBS     B_SDA                  ;Data low-high == stop
002E 0705                  BTFSS    B_SDA>>8,B_SDA&7
002F 0A2C                  GOTO     ii_0
                           I2CSTOP
                           USER_MSG                       ;process completed message!
                    ;*** This would be user code to process a message
0030 0A11                  GOTO     I2CWAIT                ;back to main loop
```

```
                    ii_1    BBC     B_SDA,I2CWAIT           ;Data high-low == start
0031 0705                   BTFSS   B_SDA>>8,B_SDA&7
0032 0A11                   GOTO    I2CWAIT
                            BBC     B_SCL,loop4             ;Wait for clock  low
0033 0725                   BTFSS   B_SCL>>8,B_SCL&7
0034 0A24                   GOTO    loop4
0035 0A31                   GOTO    ii_1
                    ACK_I2C
                            BBC     B_UA,ACK_UA             ;Not addressed - check unit address
0036 0727                   BTFSS   B_UA>>8,B_UA&7
0037 0A8B                   GOTO    ACK_UA
                            BBS     B_SA,ACK_SA             ;Reading secondary address
0038 0647                   BTFSC   B_SA>>8,B_SA&7
0039 0A97                   GOTO    ACK_SA
                    ;****
                    ;** Do what must be done with new data bytes here (before A

                    ;** Don't ack if byte can't be processed!
                    ;****
                    ;————
                    USER_RECV
                    ;*** This would be user code to process a received byte
                    ;*** example code sends sub-address 0 to port b
003A 0767                   BTFSS   B_ID>>8,B_ID&7
003B 0A3F                   GOTO    _NXI_notid
003C 0208                   MOVFW   I2CREG                  ;get received byte
003D 0026                   MOVWF   PORTB                   ;and write it on portb
003E 0A47                   GOTO    IN_CONT
                    _NXI_notid
003F 0C07                   MOVLW   07H                     ;Register count
0040 0169                   ANDWF   I2CSUBA,f               ;Limit register count
0041 0C0B                   MOVLW   I2CR0                   ;Pseudo-registers
0042 01C9                   ADDWF   I2CSUBA,W               ;Offset from buffer start
0043 02A9                   INCF    I2CSUBA                 ;Next sub-address
0044 0024                   MOVWF   FSR                     ;Indirect address
0045 0208                   MOVFW   I2CREG
0046 0020                   MOVWF   ind                     ;Put data into register
                    IN_CONT                                 ;continue point for interce
                    ACKloop
                            BBS     B_SCL,ACKloop           ;Wait for clock low
0047 0625                   BTFSC   B_SCL>>8,B_SCL&7
0048 0A47                   GOTO    ACKloop
                            CLB     B_SDA                   ;Set ACK
0049 0405                   BCF     B_SDA>>8,B_SDA&7
004A 0CF6                   MOVLW   TAWRITE
004B 0005                   TRIS    PORTA
                            CLB     B_SDA                   ;Set ACK (just in case docs are wrong)
004C 0405                   BCF     B_SDA>>8,B_SDA&7
                    ACKloop2
                           USER_Q
                    ;*** This would be quick user code
                            BBC     B_SCL,ACKloop2          ;Wait for clock high
004D 0725                   BTFSS   B_SCL>>8,B_SCL&7
004E 0A4D                   GOTO    ACKloop2
                    ACKloop3
                           USER_Q
                    ;*** This would be quick user code
                            BBS     B_SCL,ACKloop3          ;Wait for clock low
004F 0625                   BTFSC   B_SCL>>8,B_SCL&7
0050 0A4F                   GOTO    ACKloop3
0051 0CF7                   MOVLW   TAREAD                  ;End ACK
0052 0005                   TRIS    PORTA
                            BBC     B_RD,NEXTBYTE           ;Skip if read (we were acking address on
0053 0707                   BTFSS   B_RD>>8,B_RD&7
0054 0A21                   GOTO    NEXTBYTE
                    ;*********************************************************
                    ; I2C Readback (I2C read request)
                    ; Application specific code to get bytes to send may be add
```

```
                        ; This routine gets data from location pointed to by I2CSUB
                        ; sends it to I2C. Subsequent reads get sequential addresse
                        ; AND's the register # with 7 to limit to 8 registers (for
                        ; could be modified to do a comparison to an ablolute numbe
                        ;
                        ;*****************************************************
                        NEXTOUT
                        ;*** <<< PUT NEXT BYTE INTO I2CREG HERE NOW! >>> ***
                        USER_XMIT
                        ;*** This would be user code to prepare an output byte
                        ;*** example code sends id string to output
0055 0767                       BTFSS  B_ID>>8,B_ID&7
0056 0A59                       GOTO   _NXO_notid
0057 0900                       CALL    GETID                   ;get next byte from ID chan
0058 0A60                       GOTO    OUT_CONT                ;and send it
                        _NXO_notid
0059 0C07                       MOVLW   07H                     ;Register count
005A 0169                       ANDWF   I2CSUBA,f               ;Limit register count
005B 0C0B                       MOVLW   I2CR0                   ;Pseudo-registers
005C 01C9                       ADDWF   I2CSUBA,W               ;Offset from buffer start
005D 02A9                       INCF    I2CSUBA                 ;Next sub-address
005E 0024                       MOVWF   FSR                     ;Indirect address
005F 0200                       MOVFW   ind                     ;Get data from register
                        OUT_CONT
0060 0028                       MOVWF   I2CREG
                        ;— add code here to init I2CREG! when B_ID is clear!
0061 0C08                       MOVLW   8                       ;Bit counter
0062 002A                       MOVWF   I2CBITS
                        ;** OUT bits! — external poll may be executed during low c
                        ;               may also be executed during high cycle if
                        ;* ENABLE line is checked for loss of enable ONLY during HI
                        ;*** CLOCK IS LOW — CHANGE DATA HERE FIRST!
                        ;*** loop 1: data was 1
                        iiOUT_loop_1
0063 0368                       RLF     I2CREG,F                ;Shift data out, MSB first
0064 0603                        SKPNC                          ;1->0: change
0065 0A79                        GOTO    iiOUT_1                ;Output another 1!
                                 CLB     B_SDA                  ;Output 0
0066 0405                        BCF     B_SDA>>8,B_SDA&7
0067 0CF6                        MOVLW   TAWRITE
0068 0005                        TRIS    PORTA
                                 CLB     B_SDA                  ;Set data (just in case docs are
0069 0405                        BCF     B_SDA>>8,B_SDA&7
                        iiOUT_0
006A 0004                        CLRWDT                         ;Clear watchdog timer
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_02
                                 BBC     B_SCL,iiOUT_loop_02    ;Wait for clock high
006B 0725                        BTFSS  B_SCL>>8,B_SCL&7
006C 0A6B                        GOTO   iiOUT_loop_02
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_03
                                 BBS    B_SCL,iiOUT_loop_03     ;Wait for clock low
006D 0625                        BTFSC  B_SCL>>8,B_SCL&7
006E 0A6D                        GOTO   iiOUT_loop_03
006F 02EA                        DECFSZ  I2CBITS                ;Count bits
0070 0A74                        GOTO    iiOUT_loop_0           ;Loop for last bit 0
0071 0CF7                        MOVLW   TAREAD                 ;Done with last bit 0... Se
0072 0005                        TRIS    PORTA
0073 0A80                        GOTO    iiOUT_ack              ;Get ACK
                        iiOUT_loop_0
0074 0368                        RLF     I2CREG,F               ;Shift data out, MSB first
0075 0703                        SKPC                           ;0->1: change
0076 0A6A                        GOTO    iiOUT_0                ;Output another 0!

0077 0CF7                        MOVLW   TAREAD                 ;Set to 1
0078 0005                        TRIS    PORTA
```

```
                        iiOUT_1
0079 0004                CLRWDT                         ;Clear watchdog timer
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_12
                                BBC     B_SCL,iiOUT_loop_12    ;Wait for clock high
007A 0725                       BTFSS  B_SCL>>8,B_SCL&7
007B 0A7A                       GOTO    iiOUT_loop_12
                        USER_Q
                        ;*** This would be quick user code
                        iiOUT_loop_13
                                BBS     B_SCL,iiOUT_loop_13    ;Wait for clock low
007C 0625                       BTFSC   B_SCL>>8,B_SCL&7
007D 0A7C                       GOTO    iiOUT_loop_13
007E 02EA                       DECFSZ  I2CBITS                ;Count bits
007F 0A63                       GOTO    iiOUT_loop_1           ;Loop for last bit 1
                        iiOUT_ack                              ;Get acknowledge
0080 02A9                       INCF    I2CSUBA                ;Next sub-address
                        iiOUT_loop_a2
                                BBC     B_SCL,iiOUT_loop_a2    ;Wait for clock high
0081 0725                       BTFSS   B_SCL>>8,B_SCL&7
0082 0A81                       GOTO    iiOUT_loop_a2
                                BBS     B_SDA,I2CWAIT          ;No ACK – wait for restart!
0083 0605                       BTFSC   B_SDA>>8,B_SDA&7
0084 0A11                       GOTO    I2CWAIT
                        ;– prepare next character here!
                        iiOUT_loop_a3
                                BBC     B_SCL,NEXTOUT          ;Wait for clock low - output next!
0085 0725                       BTFSS   B_SCL>>8,B_SCL&7
0086 0A55                       GOTO    NEXTOUT
                                BBS     B_SDA,iiOUT_loop_a3    ;Watch out for new start condition!
0087 0605                       BTFSC   B_SDA>>8,B_SDA&7
0088 0A85                       GOTO    iiOUT_loop_a3
0089 0A11                       GOTO    I2CWAIT                ;Stop received!
008A 0A11                       GOTO    I2CWAIT
                        ;*********************************************************
                        ;* Unit address received - check for valid address
                        ;*
                        ;*********************************************************
                        ACK_UA
                                 SEB     B_UA                  ;Flag unit address received
008B 0527                        BSF     B_UA>>8,B_UA&7
008C 0608                        BTFSC   I2CREG,0              ;Skip if data coming in
                                 SEB     B_RD                  ;Flag - reading from slave
008D 0507                        BSF     B_RD>>8,B_RD&7
008E 0208                        MOVF    I2CREG,W              ;Get address
008F 0EFE                        ANDLW   0FEH                  ;Mask direction flage before compare
0090 0FD6                        XORLW   DEVICE_ADDRESS        ;Device address
0091 0743                        BNZ     I2CWAIT               ;Not for me! (skip rest of message)
0092 0A11                        BBS     B_RD,ACKloop          ;Read - no secondary address
0093 0607                        BTFSC   B_RD>>8,B_RD&7
0094 0A47                        GOTO    ACKloop
                                 SEB     B_SA                  ;Next is secondary address
0095 0547                        BSF     B_SA>>8,B_SA&7
0096 0A47                        GOTO     ACKloop              ;Yes! ACK address and continue
                        ;*********************************************************
                        ;* Secondary address received - stow it!
                        ;* SA = 0 is converted to 128 to facilitate ID read
                        ;*********************************************************
                                ACK_SA
                                 CLB     B_SA                  ;Flag second address received
0097 0447                        BCF     B_SA>>8,B_SA&7
                                 CLB     B_ID
0098 0467                        BCF     B_ID>>8,B_ID&7
0099 0208                        MOVFW    I2CREG               ;Get subaddress
009A 0643                        SKPNZ                         ;Not 0
                                 SEB     B_ID                  ;Flag - id area selected
009B 0567                        BSF     B_ID>>8,B_ID&7
```

```
009C 0029                       MOVWF   I2CSUBA             ;Set subaddress
009D 0A47                       GOTO    ACKloop
                                END


Errors   :    0
Warnings :    0
```

PIC16C54A EMI Results

INTRODUCTION

This paper discusses the EMI results of the PIC16C54A. These measurements were taken by an independent consulting firm that specializes in electromagnetic testing. These results are for a specific system design, each design will have its own results.

DEVICES USED

These tests were done on a random PIC16C54A device, and should be considered as typical results. Initial testing was done on three boards / devices. The device frequency of the boards were, respectively, 32 KHz, 4 MHz, and 20 MHz. As would be expected there was a substantial difference (decrease) at the low frequency as compared to the higher frequencies. The difference between the 4 MHz operation and the 20 MHz operation was marginal. The final testing of the device was done at 4 MHz, and was according to the FCC measurement procedure MP-4.

SYSTEM USED

The PIC16C54A device was tested in the Microchip OHMMETER board. This board is a three layer board, with a ground plane. Power and ground planes greatly help in the compliance of designs to the FCC part 15 subpart B testing. This board had minimal other external components, so that the electromagnetic measurements could mostly be attributed to the device and design of the system. To reduce the noise that comes from a power supply, a 10 nF bypass capacitor was attached to the power / ground of the input jack.

EMI TESTING

The testing of electromagnetic noise (EM) on a system has two types of commonly used testing environments, an internal and external environment. The internal test is done is a screen room to reduce the amount of stray EMI. The indoor tests is useful in determining the source of EMI radiation. The external test is done outdoors. This places the equipment under test in an environment to measure radiated emissions (EMI). The FCC only requires the outdoor testing of devices. The equipment under test (EUT) was positioned to maximize the emissions.

INTERNAL TESTS

The equipment under test was performed on a wooden test bench, inside a screen room, 0.8m above the earth ground plane (see Figure 1). The EUT was powered through the Line Impedance Stabilization Network (LISN) bonded to the ground plane. The LISN power was filtered and the filter was bonded to the ground plane. The EUT was positioned on the table with the minimum distance from any conductive surface, as specified in MP-4. The excess power cord was wrapped in a figure-8 pattern to form a bundle approximately 8 cm in length. The EUT configuration was set for the highest emission frequency, and data was collected under the program control of the host computer. The spectrum analyzer collected the maximum peak readings over each spectrum. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 1.

TABLE 1: CONDUCTED EMISSIONS RESULTS

Frequency (MHz)	Emission Level dBuV	Emission Level uV	Specification Limit uV
0.5345	33.2	46	250
7.085	33.1	45	250
10.08	33.1	45	250
10.42	33.3	46	250
11.62	33.4	47	250
13.41	33.8	49	250

PIC16C54A EMI Results

While in the screen room additional analyses on the device was done. First a local probe was used to test the emission levels around the board and device. There were no measurable emissions at the I/O pins or their corresponding trace lines. Emissions were measured at the jack to the power supply. The power supply and cord were the greatest source of emissions for the EUT. This is due to the power cord being an antenna which emitted the noise from the power supply. Designers should attempt to minimize antennas, which emit EMI. Antennas could be the power supply cord, as well as traces on the system board.

Second the PIC16C54A was monitored for susceptibility, following the IEC 801-3 specification. This test was measured from 27 MHz to 500 MHz in 10 KHz steps. The device did not display any signs of susceptibility (see Table 2).

EXTERNAL TESTS

The open field site used for radiated emission testing was setup according to the FCC bulletin OST 55. Figure 2 shows the layout of the open field test site. The EUT was mounted on a turntable. The position of the turntable is remote controlled to determine the highest emission levels. Initial testing was done with a broad band mounted on the antenna mast distance of 3 meters. Further investigation was done to determine the EUT positions that produced the maximum level of emissions. The receiving antenna was mounted on the antenna mast. The antenna height was varied to find the highest level of radiated emissions at each frequency. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 3. Figures 3 and 4 show the dBuV vs. MHz graphs.

TABLE 2: RADIATED SUSCEPTIBILITY

Frequency (MHz)	SPEC V/M	Threshold V/M	Modulation	Comments
27.0 - 500.0 †	3.0	> 3.0	80%	No Susceptibility

† Frequency incrementented in 10 KHz steps

TABLE 3: RADIATED EMISSIONS RESULTS

Frequency	Meter Reading (dBuV)	Antenna Factor dB	Effective Gain dB	Distribution Factor dB	Corr. Rdg dBuV/m	Corr. Rdg. uV/m	Spec Limit uV/m
36.01	53.1	11.8	34.0	0	30.9	35	100
40.04	53.0	11.3	34.2	0	30.1	32	100
44.04	56.1	11.1	34.0	0	33.2	46	100
48.04	55.1	11.0	33.7	0	32.4	42	100
64.05	55.2	9.0	33.9	0	30.3	33	100
112.05	56.0	10.6	33.4	0	33.2	46	150

FIGURE 1

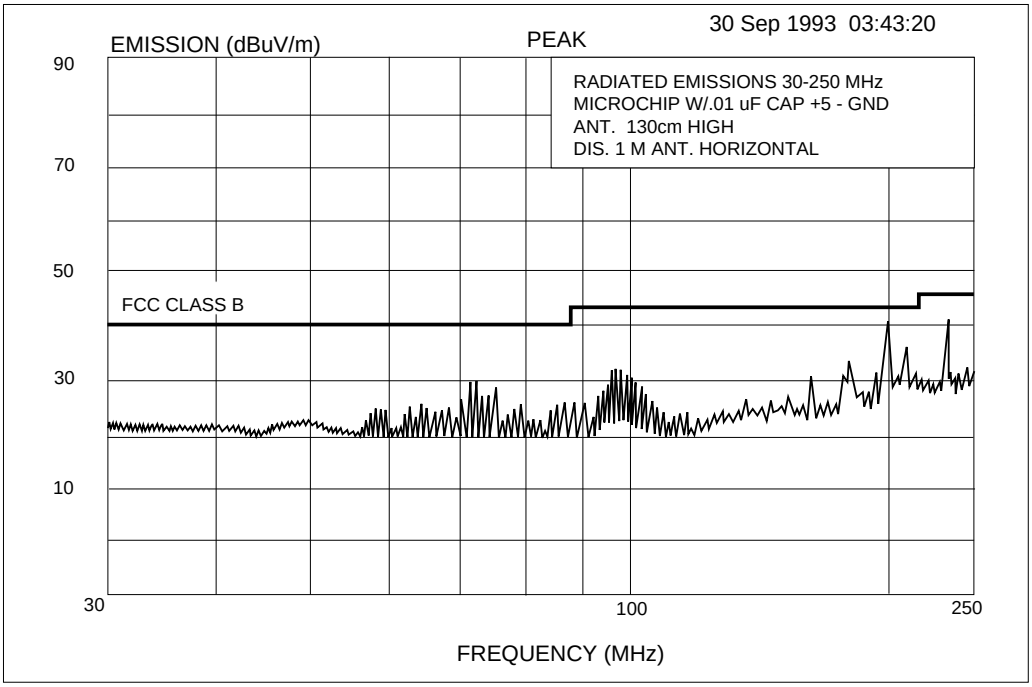
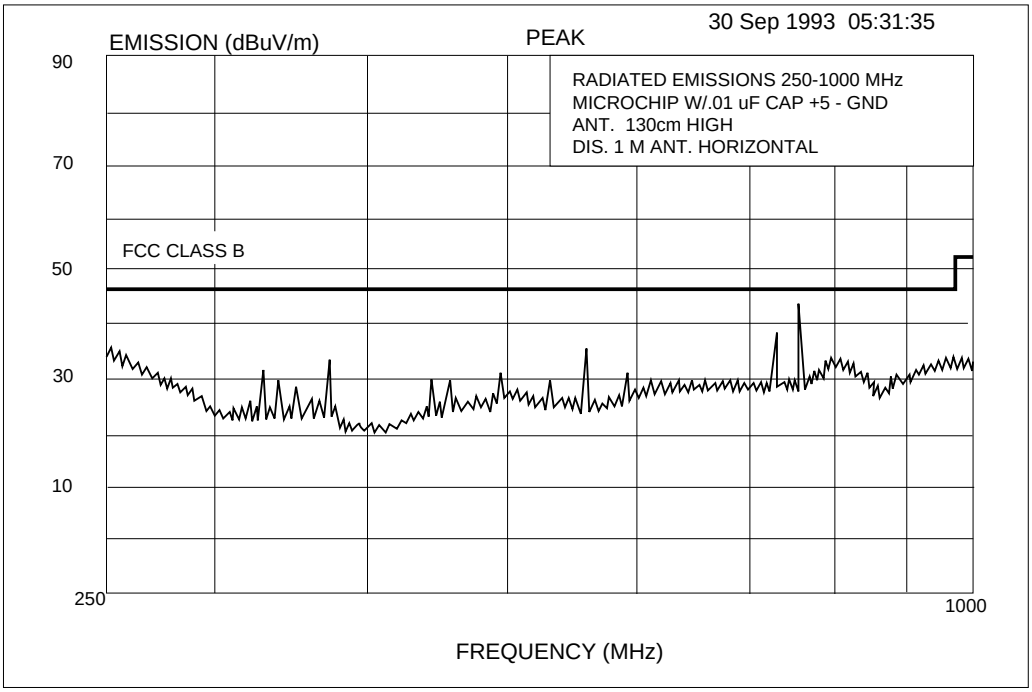


FIGURE 2



PIC16C54A EMI Results

CONCLUSION

The PIC16C54A device can be implemented into system designs that are required to be certified to the FCC Class B specification limits as defined by the FCC Title 47, Part 15 Subpart B and IEC 801-3 susceptibility.

APPENDIX A: TEST SETUPS

FIGURE 3: CONDUCTED EMISSIONS TEST SETUP FOR SITE "A"

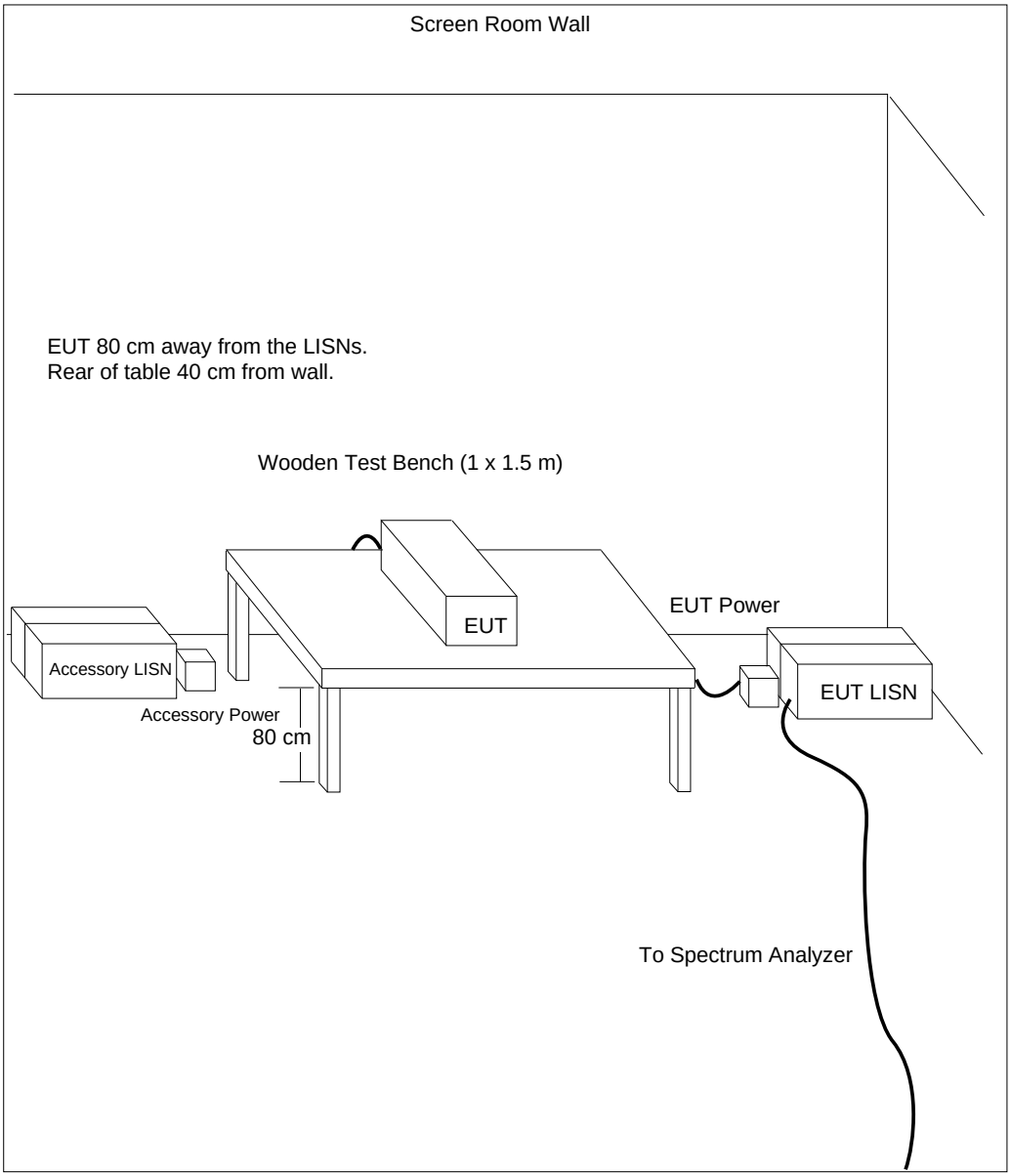
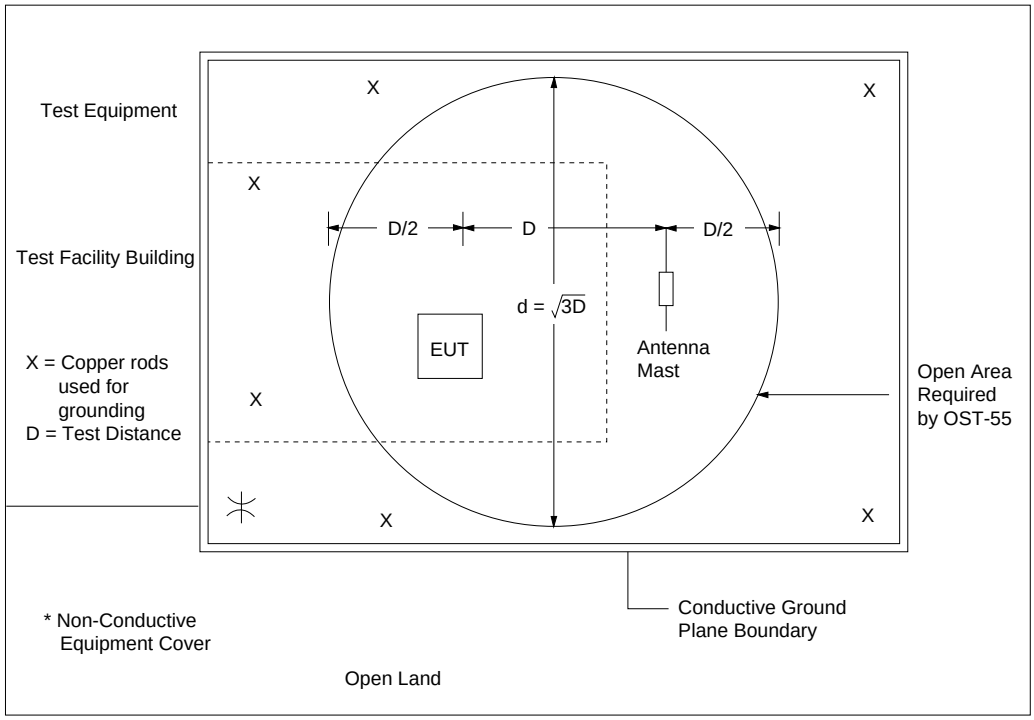


FIGURE 4: RADIATED EMISSIONS TEST SETUP FOR SITE "A"



OHMMETER Board Setup:	*Stan D'Souza*
Written By:	*Mark Palmer*
Testing By:	*Compatible Electronics*

PIC16C54A EMI Results

NOTES:

SECTION 3
PIC16CXX APPLICATION NOTES

AN566

Using the PortB Interrupt on Change as an External Interrupt

INTRODUCTION

The PIC16/17 family of RISC-like microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. To address these applications, there is the PIC16CXX microcontroller family of products. This family has numerous peripheral and special features to better address user applications.

One feature is the interrupt on change of the PORTB pins. This “interrupt on change” is caused when any of the RB<7:4> pin, configured as input, changes levels. When used in conjunction with the software programmable weak internal pull-ups, a direct interface to a keypad is possible. This is shown in application note AN552 (Implementing Wake-up on Key Stroke). Another way to use the “interrupt on change” feature would be as additional external interrupt sources. This allows the PIC16CXX devices to support multiple external interrupts, in addition to the INT pin.

This application note will discuss some of the issues in using PortB as additional external interrupt pins, and will show some examples. These examples can be easily modified to suit your particular needs.

USING A PORTB INPUT FOR AN EXTERNAL INTERRUPT

The interrupt source(s) cannot simply be directly connected to the PortB pins, and expect the interrupt to function the same as the interrupt (INT) pin. The characteristic of the interrupt signal must also be known to develop the microcontrollers hardware/software. After we know this, we can determine the best way to structure the program to handle this signal. These characteristics include:

1. Trigger interrupt on rising, falling, or both edges
2. What is the pulse width of the interrupt (high time / low time)

It is easy to understand the need of knowing which edge triggers the external interrupt service routine. This allows one to ensure that the interrupt service routine is only entered for the desired edge, with all other edges ignored. Not so clear is pulse width of the interrupt. This determines the amount of additional overhead that the software routine may need.

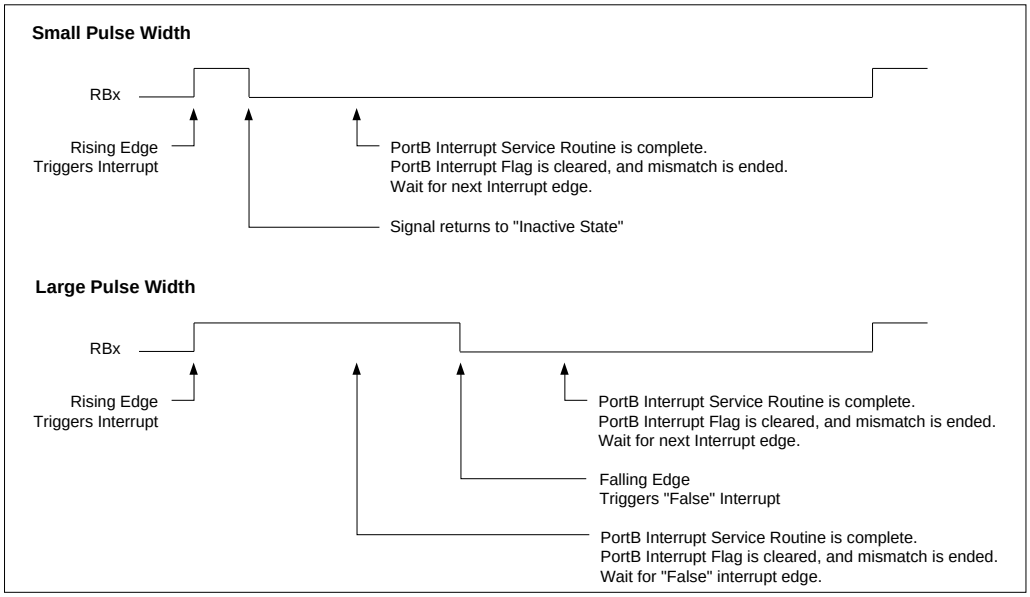
Using the PortB Interrupt on Change as an External Interrupt

Figure 1 shows the two cases for the interrupt signal verses the time to complete the interrupt service routine. The first waveform is when the signal makes the low-to-high-to-low transitions before the interrupt service routine has completed (interrupt flag cleared). When the interrupt flag has been cleared the interrupt signal has already returned to the inactive level. The next transition of the signal is due to another interrupt request. An interrupt signal with this characteristic will be called a small pulse width signal. The second waveform is when the signal only makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). The next transition (high-to-low) will return the interrupt signal to the inactive level. This will generate a “false” interrupt, that will need to be cleared. Then the following transition (low-to-high) will be a “true” interrupt. An interrupt signal with this characteristic will be called a wide pulse width signal.

An interrupt pulse with a small pulse width requires less overhead than a wide pulse width. A small pulse width signal must be less then the minimum execution time of the interrupt service routine, while a wide pulse width must be greater then the maximum time through the interrupt service routine.

Example 1 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a small pulse width. In this case since the interrupt pulse width is small, the pulse has gone high and then low again before PortB is read to end the mismatch condition. So when PortB is read it will read a low signal and will again be waiting for the rising edge transition.

FIGURE 1 - INTERRUPT STEPS FOR SMALL AND WIDE PULSE WIDTHS



Example 1: Single Interrupt with a Small Pulse Width

```
PER_INT         BTFSS   INTCON, RBIF          ; PortB interrupt?
                GOTO    OTHER_INT             ; Other interrupt
                :                             ; Do task for INT on RB7
                :                             ;
CLR_RBINTF      MOVF    PORTB, 1              ; Read PortB (to itself) to end
                                              ; mismatch condition
                BCF     INTCON, RBIF          ; Clear the RB interrupt flag.
                RETFIE                        ; Return from interrupt
OTHER_INT               :                     ; Do what you need to here
                        :
                RETFIE                        ; Return from interrupt
```

Using the PortB Interrupt on Change as an External Interrupt

Example 2 shows a single interrupt source on PortB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a wide pulse width. In this case since the interrupt pulse width is large, the pulse is still high before PortB is read to end the mismatch condition. So when PortB is read it will read a high signal and will generate an interrupt on the next falling edge transition (which should be ignored).

Example 3 shows a interrupt on change with the interrupt source on PortB (RB7). This executes the interrupt service routine on a both edges. The interrupt source must have a minimum pulse width to ensure that both edges can be "seen". The minimum pulse width is the maximum time from the interrupt edge to the reading of PortB and clearing the interrupt flag.

Example 2: Single Interrupt with a Wide Pulse Width

```
PER_INT         BTFSS   INTCON, RBIF          ; PortB interrupt?
                GOTO    OTHER_INT             ; Other interrupt
                BTFSC   PORTB, RB7            ; Check for rising edge
                GOTO    CLR_RBINTF            ; Falling edge, clear PortB int
                :                             ;   flag
                        :                     ; Do task for INT on RB7
                        :
CLR_RBINTF      MOVF    PORTB, 1              ; Read PortB (to itself) to end
                                              ; mismatch condition
                BCF     INTCON, RBIF          ; Clear the RB interrupt flag.
                RETFIE                        ; Return from interrupt
OTHER_INT               :                     ; Do what you need to here
                        :
                RETFIE                        ; Return from interrupt
```

Example 3: Interrupt on Change

```
PER_INT         BTFSS   INTCON, RBIF          ; PortB interrupt?
                GOTO    OTHER_INT             ; Other interrupt
CLR_RBINTF      MOVF    PORTB, 1              ; Read PortB (to itself) to end
                                              ; mismatch condition
                BCF     INTCON, RBIF          ; Clear the RB interrupt flag.
                        :                     ; Do task for INT on RB7
                        :                     ;
                RETFIE                        ; Return from interrupt
OTHER_INT               :                     ; Do what you need to here
                        :
                RETFIE                        ; Return from interrupt
```

Using the PortB Interrupt on Change as an External Interrupt

Using PortB Inputs for Multiple Interrupts

The previous examples have been for a single external interrupt on PORTB. This can be extended to support up to 4 external interrupts. To do this requires additional software overhead, to determine which of the PortB pins (RB<7:4>) caused the interrupt. Care should be taken in the software to ensure that no interrupts are lost.

In this example, the interrupt sources on RB7, RB5, and RB4 have a small pulse width, while the interrupt source on pin RB6 is wide and should cause a trigger on the rising edge.

SUMMARY

The PortB interrupt on change feature is both a very convenient method for direct interfacing to an external keypad, with no additional components, but is also versatile in its uses. The ability to add up to four additional external interrupt. Of course hybrid solutions are also possible. That is, for example, using PORTB<6:1> as a 3x3 keypad, with PORTB7 as an external interrupt and PORTB0 as a general purpose I/O. The flexibility of this feature allows the user to implement a best fit design for the application.

Example 4: Multiple Interrupts with Different Pulse Widths

```
PER_INT         BTFSS   INTCON, RBIF          ; PortB interrupt?
                GOTO    OTHER_INT             ; Other interrupt
;
; PortB change interrupt has occurred. Must determine which pin caused
; interrupt and do appropriate action. That is service the interrupt,
; or clear flags due to other edge.
;
                MOVF    PORTB, 0              ; Move PortB value to the W register
                                              ;   This ends mismatch conditions
                MOVWF   TEMP                  ; Need to save the PortB reading.
                XORWF   LASTPB, 1             ; XOR last PortB value with the new
                                              ;   PortB value.
CK_RB7          BTFSC   LASTPB, RB7           ; Did pin RB7 change
                CALL    RB7_CHG               ; RB7 changed and caused the interrupt
CK_RB6          BTFSC   LASTPB, RB6           ; Did pin RB6 change
                CALL    RB6_CHG               ; RB6 changed and caused the interrupt
CK_RB5          BTFSC   LASTPB, RB5           ; Did pin RB5 change
                CALL    RB5_CHG               ; RB5 changed and caused the interrupt
CK_RB4          BTFSC   LASTPB, RB4           ; Did pin RB4 change
                GOTO    RB4_CHG               ; RB4 changed and caused the interrupt
;
RB7_CHG              :                        ; Do task for INT on RB7
                     :                        ;
                RETURN
RB6_CHG         BTFSC   PORTB, RB6            ; Check for rising edge
                RETURN                        ; Falling edge, Ignore
                     :                        ; Do task for INT on RB6
                     :
                RETURN
RB5_CHG              :                        ; Do task for INT on RB5
                     :                        ;
                RETURN
RB4_CHG              :                        ; Do task for INT on RB4
                     :                        ;
CLR_RBINTF      MOVF    TEMP, 0               ; Move the PortB read value to the
                MOVWF   LASTPB                ;   register LASTPB
                BCF     INTCON, RBIF          ; Clear the RB interrupt flag.
                RETFIE                        ; Return from interrupt
;
OTHER_INT            :                        ; Do what you need to here
                     :
                RETFIE                        ; Return from interrupt
```

Author: Mark Palmer, Logic Products Division

AN552

Implementing Wake-up on Key Stroke

INTRODUCTION

Microchip's PIC16CXX family of microcontrollers are ideally suited to directly interface to a keypad. The high 4-bits of PortB (RB4 - RB7) have internal pull-ups and can trigger a "change on port state" interrupt. This interrupt, if enabled, will wake the microcontroller from sleep. In most battery powered applications, a microcontroller is exercised when a key is pressed, e.g. in a remote keyless entry system. The life of the battery can be extended by using PIC16CXX microcontrollers. This can be done by putting the PIC16CXX microcontroller into sleep mode for most of the time and wake-up only when a key is pressed.

IMPLEMENTATION

Figure 1 depicts an application where four keys are connected to RB4 - RB7. Internal pull-ups are used to maintain a high level on these inputs. In this example, LEDs are connected to RB0 - RB3. When SW1 is pressed, LED1 is turned on and when SW2 is pressed, LED2 is turned on and so on. The PIC16CXX is normally in sleep mode with the "change on port state" interrupt enabled. When SW1 is pressed, RB4 goes low and triggers an interrupt. Since the PIC16CXX is in sleep, it first wakes up and starts executing code at the interrupt vector. Note that if the global interrupt is enabled, the program execution after an interrupt is at the interrupt vector, if the global interrupt is not enabled, the program starts executing right after the sleep instruction.

After waking up, a 20 - 40 msec. de-bounce delay is executed which checks the port for a key hit and, depending on which key is hit, its associated LED is turned on. The LEDs are used purely for demonstration purposes. In a remote keyless entry application, the remote code would be transmitted when the appropriate key is hit.

Figure 2 depicts a 4x4 keypad interface to the PIC16CXX. When using the PIC16CXX in a keypad application, the internal pull-ups on RB4 - RB7 can be enabled eliminating the need for external pull-up resistors. The series 100Ω resistors are used for ESD protection, and are recommended in keypad interface applications.

Author: Stan D'Souza, Logic Products Division

SUMMARY

The PIC16CXX is ideally suited to interface directly to a Keypad application. Built in pull-up resistors and very low sleep current make it a very good candidate for battery powered remote operations and applications.

Performance:

Code Size:	64 words
RAM Locations Used:	0 bytes

FIGURE 1 - 4 KEY INTERFACE TO PIC16CXX

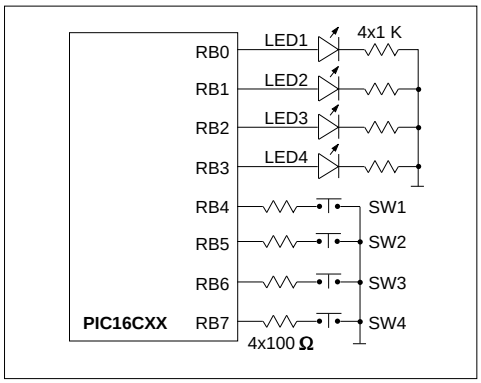
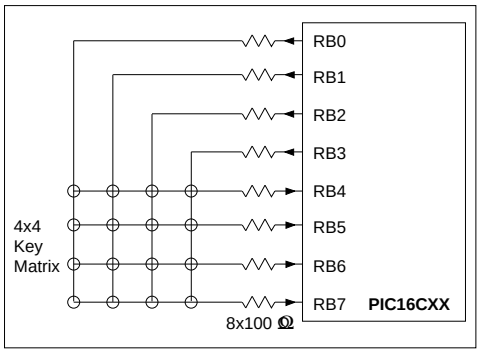


FIGURE 2 - 4X4 KEYPAD INTERFACE TO PIC16CXX



Implementing Wake-up on Key Stroke

```
MPASM 1.00 Released     WAKUP.ASM   7-15-1994  13:24:29                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;This program demonstrates the wake-up on Keystroke feature of the
                     0002 ;PIC16C71. Port B pins RB4 - RB7 can be configured as inputs with
internal
                     0003 ;pull up resistors, also the interrupt associated with the change on
input
                     0004 ;on RB4 - RB7 can be set up to wake the chip from sleep. If the
                     0005 ;global interrupt is enabled just before sleep, the program will vector
to
                     0006 ;the interrupt vector (0004). If not the chip will continue execution
                     0007 ;just after the next instruction following sleep.
                     0008 ;In this example code, the port B is initalized to input 4 push-buttons
at
                     0009 ;RB4 - RB7. RB0 - RB3 are configured to drive LEDs corresponding to
                     0010 ;which push-button is hit (LED on RB0 when RB4 is hit and so on).
                     0011 ;Sleep is executed. When any keys is hit, the processor wakes
                     0012 ;up and jumps to the interrupt vector. The corresponding LED is
                     0013 ;turned on and after the key is released, the whole process is re-
peated.
                     0014 ;
                     0015         LIST P=16C71, F=INHX8M
                     0016 ;
0002                 0017 z       equ     2
0007                 0018 RBPU    equ     7
0010                 0019 temp    equ     10h
0001                 0020 OptionReg  equ     1h
                     0021     include “picreg.equ”
                     0083
                     0084
                     0021
                     0022 ;
                     0023         org     0
0000 2805            0024         goto    start
                     0025 ;
                     0026         org     4
0004 2808            0027         goto    ServiceInterrupt
                     0028 ;
                     0029 ;
                     0030 start
0005 2024            0031         call    InitPortB       ;initalize port B
                     0032 loop
                     0033 ;        sleep                    ;sleep till key is hit
0006 0000            0034         nop
0007 2806            0035         goto    loop
                     0036 ;
                     0037 ServiceInterrupt
0008 180B            0038         btfsc   INTCON,RBIF     ;change on rb int?
0009 280D            0039         goto    ServiceWakup    ;yes then service
000A 128B            0040         bcf     INTCON,RTIE     ;clear RTCC int mask
000B 110B            0041         bcf     INTCON,RTIF     ;clear flag
000C 0008            0042         return
                     0043 ;
                     0044 ;This routine checks which keys is hit and lights up the
                     0045 ;corresponding LED associated with it. eg. RB0’s LED when
                     0046 ;RB4’s key is pressed. Finally it waits till all keys have
                     0047 ;been released before returning form the service routine.
                     0048 ServiceWakup
000D 118B            0049         bcf     INTCON,RBIE     ;clear mask
000E 0906            0050         comf    PORT_B,w        ;read PORT_B

000F 100B            0051         bcf     INTCON,RBIF     ;clear flag
0010 2035            0052         call    delay16         ;do de-bounce for 16mSecs
0011 0906            0053         comf    PORT_B,w        ;read port B again
0012 39F0            0054         andlw   B’11110000'     ;mask outputs
0013 0090            0055         movwf   temp            ;save in temp
```

```
0014 0E10           0056         swapf   temp,w          ;switch low and high
0015 0086           0057         movwf   PORT_B          ;send as outputs.
0016 2018           0058         call    KeyRelease      ;check for key release
0017 0009           0059         retfie
                    0060 ;
                    0061 ;This sub-routine, waits till all key have been released
                    0062 ;In order to save power, the chip is in sleep mode till
                    0063 ;all keys are released.
                    0064 KeyRelease
0018 2035           0065         call    delay16         ;do debounce
0019 0906           0066         comf    PORT_B,w        ;read PORT_B
001A 100B           0067         bcf     INTCON,RBIF     ;clear flag
001B 158B           0068         bsf     INTCON,RBIE     ;enable mask
001C 39F0           0069         andlw   B'11110000'     ;clear outputs
001D 1903           0070         btfsc   STATUS,z        ;key still pressed?
001E 0008           0071         return                  ;no then return
001F 0063           0072         sleep                   ;else save power
0020 118B           0073         bcf     INTCON,RBIE     ;on wake up clear mask
0021 0906           0074         comf    PORT_B,w
0022 100B           0075         bcf     INTCON,RBIF     ;clear flag
0023 2818           0076         goto    KeyRelease      ;try again
                    0077 ;
                    0078 ;
                    0079 ;This sub-routine, initializes PortB.
                    0080 InitPortB
0024 1683           0081         bsf     STATUS,RP0      ;select bank 1
0025 3003           0082         movlw   B'00000011'     ;Port_A digital I/O
0026 0088           0083         movwf   ADCON1          ;               /
0027 3000           0084         movlw   0               ;
0028 0085           0085         movwf   PORT_A          ;set port a as outputs
0029 30F0           0086         movlw   B'11110000'     ;RB0-RB3 outputs
002A 0086           0087         movwf   PORT_B          ;RB4-RB7 inputs
002B 1381           0088         bcf     OptionReg,RBPU  ;enable pull up
002C 1283           0089         bcf     STATUS,RP0      ;select page 0
002D 0186           0090         clrf    PORT_B          ;init port B
002E 0185           0091         clrf    PORT_A          ;make port a all low
002F 1405           0092         bsf     PORT_A,0        ;make first bit high
0030 118B           0093         bcf     INTCON,RBIE     ;disable mask
0031 0806           0094         movf    PORT_B,w        ;read port
0032 100B           0095         bcf     INTCON,RBIF     ;clear flag
0033 158B           0096         bsf     INTCON,RBIE     ;enable mask
0034 0009           0097         retfie                  ;enable global and return
                    0098 ;
                    0099 ;delay16 waits for approx 16.4mSecs using RTCC interrupts
                    0100 ;fosc speed is 4Mhz.
                    0101 delay16
0035 1683           0102         bsf     STATUS,RP0      ;select page 1
0036 3007           0103         movlw   B'00000111'     ;fosc/256 -> RTCC
0037 0081           0104         movwf   OptionReg       ;       /
0038 1283           0105         bcf     STATUS,RP0      ;select page 0
0039 0181           0106         clrf    RTCC
003A 110B           0107         bcf     INTCON,RTIF     ;clear flag
003B 168B           0108         bsf     INTCON,RTIE     ;enable mask
                    0109 CheckAgain
003C 1D0B           0110         btfss   INTCON,RTIF     ;timer overflowed?
003D 283C           0111         goto    CheckAgain      ;no check again
003E 128B           0112         bcf     INTCON,RTIE     ;else clear mask
003F 110B           0113         bcf     INTCON,RTIF     ;clear flag
0040 0008           0114         return
                    0115 ;
                    0116         end
                    0117
                    0118
                    0119
                    0120
                    0121
                    0122
                    0123
                    0124
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X—XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : X——————— ———————— ———————— ————————

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

AN579

Using the 8-Bit Parallel Slave Port

INTRODUCTION

The PIC16C64/C74 microcontrollers from Microchip Technology Inc., can be interfaced in a multi-microprocessor environment with ease using the built-in Parallel Slave Port. With their very high operating speeds (cycle times as low as 200ns with a clock rate of 20MHz), and an array of on-chip peripherals, they make ideal smart interfaces to the real world.

IMPLEMENTATION

PortD operates as an 8-bit wide parallel slave port, with PortE providing the control signals when bit PSPMODE (TRISE<4>) is set. In parallel slave mode, PortD is asynchronously readable and writable by the external world through the CS (RE2/CS), RD (RE0/RD), and WR (RE1/WR) control inputs.

In order to use the parallel slave port, the data direction bits in the TRISE register corresponding to RD, WR, and CE (TRISE<2:0>) must be configured as inputs (set =1).

The port pins are connected to two 8-bit latches, one for data output (from the PIC16CXX) and one for data input. The PIC16CXX sends data by writing to the output latch, and receives data by reading the input latch (note that the input and output latches are at the same address). In this mode the TRISD register is ignored, since the external device connected to the slave port controls the direction of data flow.

When the external device performs either a read or a write operation to the PIC16CXX, the interrupt flag, PSPIF (PIR1<7>), will be set and the processor interrupted if PSPIE (PIE1<7>) is set and interrupts are enabled (GIE and PEIE, (INTCON<7:6>) set). When the interrupt is serviced, PSPIF must be cleared by software.

The read-only status flag IBF, Input Buffer Full (TRISE<7>), is set if a received word is waiting to be read. IBF is cleared upon read of the input buffer latch. If another word is received prior to the first being read, status flag IBOV (TRISE<5>) is set. IBOV can be cleared by software.

The Output Buffer Full status bit, OBF (TRISE<6>), is set if a word written to PortD latch is waiting to be read by the external bus.

When not in PSPMODE the IBF and OBF bits are cleared. If the IBOV flag was previously set, however, it must be cleared by software.

FIGURE 1: SUMMARY OF PARALLEL SLAVE PORT REGISTERS

Register Name	Function	Address	Power-On Reset Value
PORTD	Parallel slave port Read/Write Data	08h	X X X X X X X X
TRISD	PortD data direction register	88h	1 1 1 1 1 1 1 1
PORTE	Parallel slave port Read/Write/Chip Select signals	09h	- - - - - - X X
TRISE	Control bits for PortD slave port	89h	0 0 0 0 - 1 1 1
INTCON	Global Interrupt Enable	0Bh	0 0 0 0 0 0 0 X
PIR1	Interrupt register (PSPIF bit)	0Ch	0 0 0 0 0 0 0 0
PIE1	Interrupt Enable register (PSPIE bit)	8Ch	0 0 0 0 0 0 0 0

FIGURE 2: PORT E FUNCTIONS

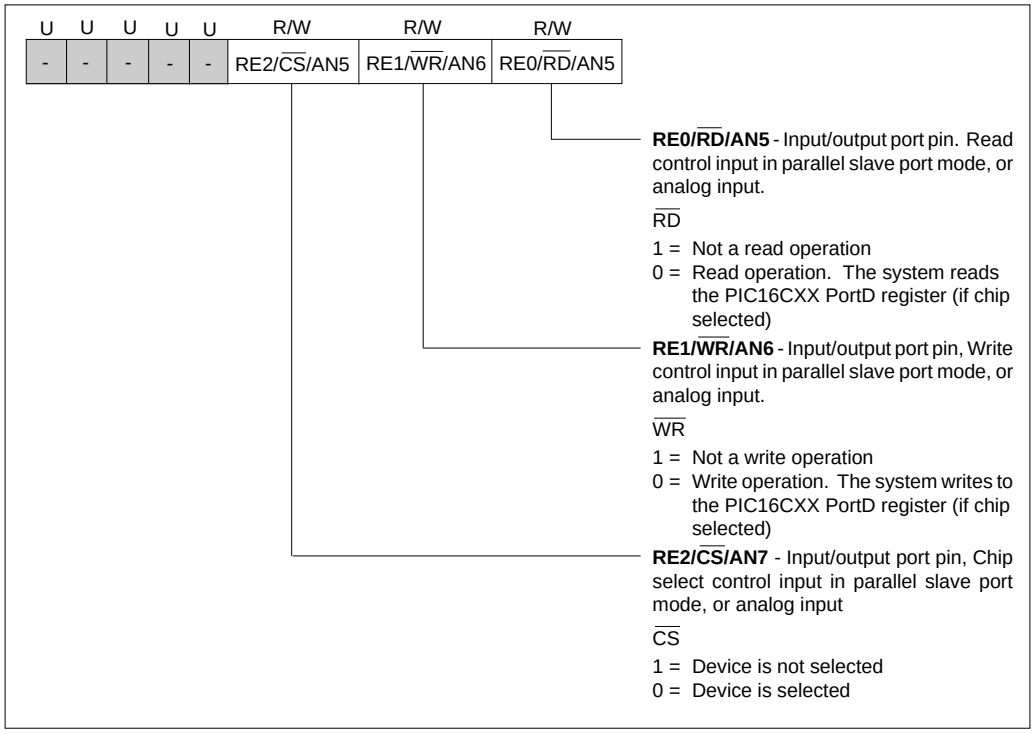
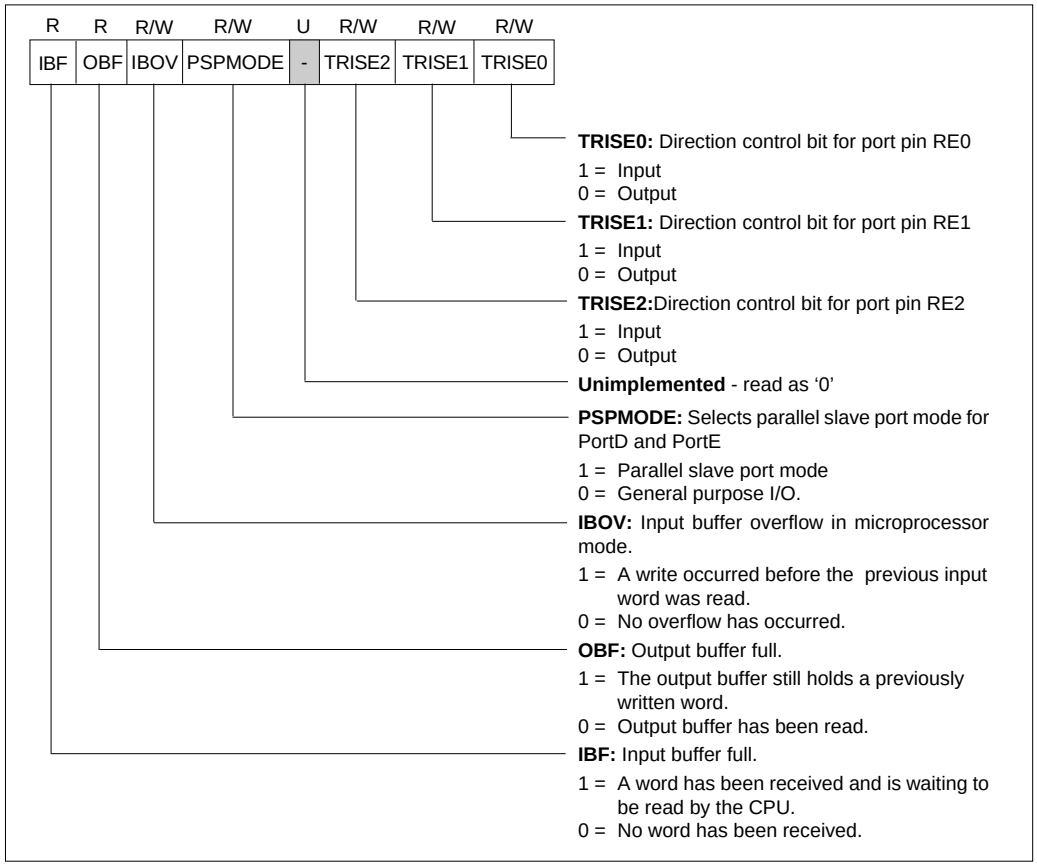


FIGURE 3: TRISE REGISTER



3

FIGURE 4: PIE1 REGISTER

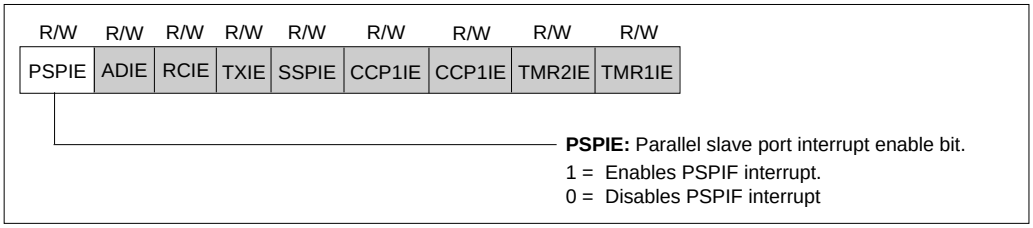


FIGURE 5: PIR1 REGISTER

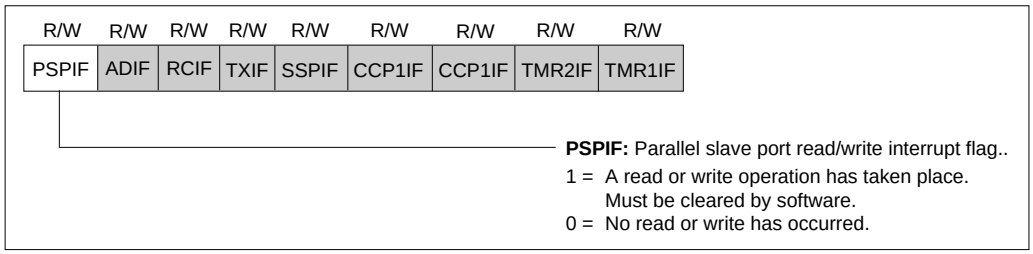
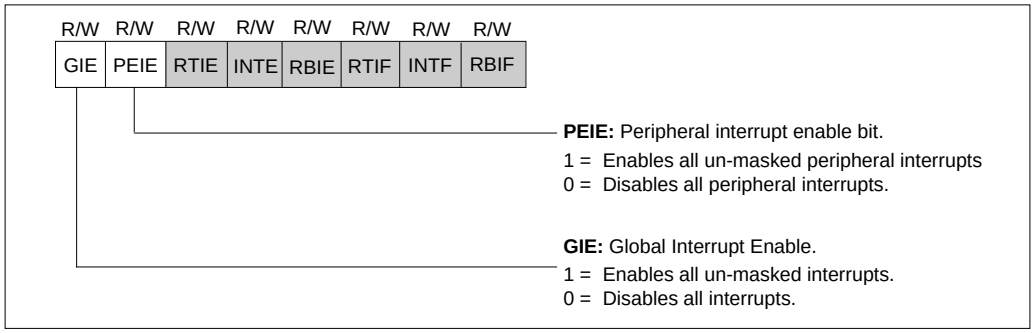


FIGURE 6: INTCON REGISTER



AUTHOR: Scott Fink, Logic Products Division

```
;***********************************************************************
;* 16C64/74 Parallel Slave port
;*             This program demonstrates the Parallel Slave Port function of
;*             the PIC16C64/74.  The program is interrupt driven, when the PIC16CXX
;*             is either read from or written to, an interrupt is generated.  If the
;*             interrupt was caused by a read, a register is incremented, and
;*             the new count is placed in an output queue.  If the interrupt was
;*             caused by a write, the data is put on the Port B pins
;***********************************************************************
               list p=16c64,f=inhx8m
;
               include “c:\16C74.inc”

;Register definitions
FLAGREG        equ        20h              ;Flag bit register
OUTDATA        equ        21h              ;Output data
INDATA         equ        22h              ;Input data
COUNT          equ        23h              ;Count of times output register read

;Bit definitions for flag register
ERROR          equ        00h              ;Error flag bit
OUTRDY         equ        01h              ;Output data ready flag
INFULL         equ        02h              ;Input data received flag

               org        0000h            ;Reset Vector
               goto       Start

               org        0005h            ;Interrupt Vector
               goto       Service_Int

Start
               clrf       OUTDATA          ;Clear data registers
               clrf       INDATA
               bsf        STATUS,RP0       ;Select register page 1
               movlw      b’00010111'      ;Set RD, WR, and CS as
               movwf      TRIS_E           ; inputs, Enable Parallel Slave port
               movlw      0FFh
               movwf      TRIS_B           ;Set Port_B to all outputs
               movlw      b’10000000'      ;
               movwf      PIE1             ;Enable Parallel Slave Port interrupt
               bcf        STATUS,RP0       ;Select register page 0

               movf       OUTDATA,W        ;Set output Data in PORTD
               movwf      PORT_D
               movlw      b’11000000'      ;Set GIE, PEIE (enable interrupts)
               movwf      INTCON

Loop
               btfsc      FLAGREG,INFULL   ;Check if input data received
               goto       Checkout         ;No data ready, check output
               bcf        FLAGREG,INFULL   ;Clear input data ready flag
               movf       Indata,W         ;Get Input data
               movwf      PORT_B           ;Output input data to Port_B
Checkout
               btfsc      FLAGREG,OUTRDY   ;Check if data output already
               goto       Loop             ;Not output yet, loop
               incf       COUNT            ;Increment output data
               movf       COUNT,W          ;Get output data
               movwf      OUTDATA          ;Put data in output queue
               bsf        FLAGREG,OUTRDY   ;Set flag for interrupt routine
               goto       Loop
```



```
;*******************************************************************************
;*Interrupt Service Routine
;*             Inputs:    FLAGREG - Flag register to/from the main routine:
;*                                     Bit 1: OUTRDY - To Service_Int, indicates data
;*                                                     ready in output queue
;*                        OUTDATA - Output data queue
;*                        PIR1           - Interrupt flag register
;*                        TRIS_E         - Parallel slave port flag register
;*                        PORT_D         - Input data from slave port
;*
;*             Outputs:
;*                        PORT_D         - Output data to slave port
;*                        INDATA         - Input data queue
;*                        FLAGREG - Flag register to/from the main routine:
;*                                     Bit 0: ERROR - From Service_Int, indicates input
;*                                                    buffer overflow
;*                                     Bit 2: INFULL- From Service_Int, indicates data
;*                                                    received and in INDATA
;*******************************************************************************

Service_Int
               btfss      PIR1,PSPIF         ;Test for Peripheral interrupt
               goto       Intout             ;Not a Peripheral interrupt, exit
               bcf        PIR1,PSPIF         ;Clear Peripheral interrupt
               bsf        STATUS,RP0         ;Select Page 1
               btfss      TRIS_E,IBF         ;Check if input data ready
               goto       Notinput           ;No input, check output
               bcf        STATUS,RP0         ;Input ready, select Page 0
               bsf        FLAGREG,INFULL     ;Set flag for main routine
               movf       PORT_D,W           ;Get input data
               movwf      INDATA             ;Put byte in input queue
Notinput
               btfsc      TRIS_E,OBF         ;Check if output data read
               goto       Intout             ;Not read, exit
               bcf        STATUS,RP0         ;Select Page 0
               btfss      FLAGREG,OUTRDY     ;Check if data in output queue
               goto       Intout             ;Output not read, exit
               movf       OUTDATA,W          ;Get data from queue
               movf       PORT_D             ;Put data in output buffer
               bcf        FLAGREG,OUTRDY     ;Clear flag for main routine
Intout
               bsf        STATUS,RP0         ;Select Page 1
               btfsc      TRIS_E,IBOV        ;Check input buffer overflow flag
               goto       Interror           ;If not clear, error
               bcf        STATUS,RP0         ;Select Page 0
               retfie                        ;Re-enable GIE and return
Interror
               bcf        STATUS,RP0         ;Select Page 0
               bsf        FLAGREG,ERROR      ;Set error flag for main routine
               retfie                        ;Re-enable GIE and return

               end
```

MICROCHIP

AN589

A PC-Based Development Programmer for the PIC16C84

Author: Robert Spur - Analog Design Specialist, Inc.

PROGRAMMING THE PIC16C84 MICROCONTROLLER

This application note describes the construction of a low cost serial programmer for the PIC16C84 microcontroller which is controlled using a PC with a parallel (Centronix printer) port. This programmer has the capability of programming the PIC16C84 microcontroller, and reading back internal data without removing the device form the target circuit.

This feature is very useful in applications where changes in program code or constants are necessary to compensate for other system features. For example, an embedded control system may have to compensate for variances in a mechanical actuator performance or loading. The basic program can be programmed and tested in design. The final program and control constants can be easily added later in the production phase without removing the microcontroller from the circuit.

Automatic software and performance upgrades can also be implemented with an in-system. Upon receiving new system software via disk or modem, a control processor with the included programming code could perform an in circuit reprogramming of other microcontrollers in the system.

This programmer can load program code, part configuration, and EEPROM data into the PIC16C84 part. In read back mode, it can verify all verify all data entries.

FUNCTION DESCRIPTION

The PIC16C84 microcontroller is put into programming mode by forcing a low logic level on the RB7 (pin 13) and RB6 (pin 12) while the MCLR (pin 4) is first brought low to reset the part, and then brought to the program/verify voltage of 12 to 14 volts. The MCLR pin remains at the program/verify voltage for the remainder of the programming or verification time.

After entering programming mode, RB7 is used to serially enter programming modes and data into the part. A high to low transition on RB6, the clock input, qualifies each bit of the data applied on RB7. The serial command-data format is specified in Figure 1.2.1.3 of the Microchip PIC16C84 Programming Specification (DS30189D). The first 6 bits form the command field, and the last 16 bits form the data field. Notice that the data field is composed of one zero starting bit, 14 actual data bits, and one zero stop bit. The increment address command, shown in Figure 1.2.1.5 (see PIC16C84 Data Sheet, DS30189D), is comprised of only the command field. Table 1.2.1.1 (see DS30189D) summarizes the available commands and command codes for serial programming mode.

The read mode is similar to programming mode with the exception that the data direction of RB7 is reversed after the 6-bit command to allow the requested data to be returned to the programmer. Figure 1.2.1.4 (see DS30189D) shows this sequence which starts by shifting the 6-bit command into the part. After the read command is issued, the programmer tri-states its buffer to allow the part to serially shift its internal data back to the programmer. The rising edge of RB6, the clock input, controls the data flow by sequentially shifting previously programmed or data bits from the part. The programmer qualifies this data on the falling edge of RB6. Notice that 16 clock cycles are necessary to shift out 14 data bits.

Accidental in circuit reprogramming is prevented during normal operation by the MCLR voltage which should never exceed the maximum circuit supply voltage of 6V DC and the logic levels of port bits RB7 and RB8.

After program/verification the MCLR pin is brought low to reset the target microcontroller and electrically released. The target circuit is then free to activate the MCLR signal. In the event MCLR is not forced by the target circuit, R4 (a 2K Ohm pull up resistor in the programmer) provides a high logic level on the target microcontroller which enables execution of its program independent of the programmer connection. Provisions should be made to prevent the target circuit from resetting the target microcontroller with MCLR or effecting the RB6 and RB6 during the programming process. In most cases this can be done without jumpers.

DETAILED CIRCUIT DESCRIPTION

A logic high on PC parallel interface latch bit D4 turns on Q3 causing the $\overline{MCLR}$ pin to go low which places the target part in reset mode. The reset condition is then removed and the program/verify voltage is applied by placing a logic high on D3 and a logic low on D4 which turns off Q3 and turns on Q2 and Q1. Circuit protection of Q1 and Q3 is obtained from connecting the emitter of Q2 to latch bit D4 which prevents a simultaneous reset and program/verify voltage mode. Q2, a 2N3904, has a reverse emitter base break down voltage of 6 volts which will not be exceeded when 5 volt logic is used on the parallel interface.

The resistors R1, R2, R3, and the diode D1 provide a logic level interface to the analog circuitry. R4 provides a $\overline{MCLR}$ (master clear) pull up function during target circuit run mode. The programming voltage is supplied and adjusted by an external lab supply. This supply should have a current limit in the 100 ma range. 5 volts for U2 (LS244) is locally regulated from programming supply voltage by U1. R5 (750 ohm resistor) is connected to the regulator output to insures proper 5 volt regulation when the 13.5 volt programming voltage is applied through the pull up resistor R4.

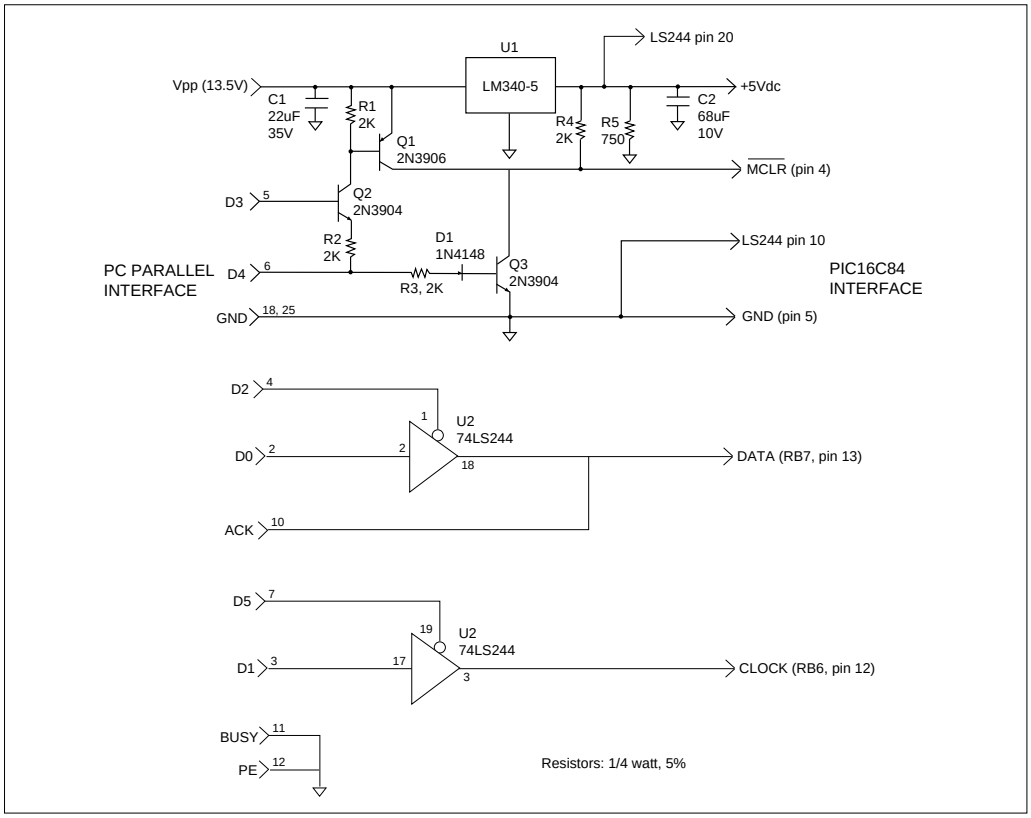
Data and clock are connected to the part via a tri-state buffer U2. PC parallel port interface bit D0 is used for data and port bit D1 is used for clock. During programming mode both clock and data buffers are enabled by port bits D2 and D5. During read mode, the data buffer is tri-stated via D2 and the printer data acknowledge signal line is used to receive verification data from the part.

After program/verification mode both the data and clock lines are tri-stated via D2 and D5 allowing the these lines to be used by the target circuit. This allows the programmer to remain physically, but not electrically connected to the target system.

An optional 5 volt line was included in the 3-foot programming interconnect cable for convince. Short interconnection leads and good grounding are always good construction practice.

To meet the programming verification specification, the target part's supply voltage should be first set to the maximum specified supply voltage and a program/data read back should be preformed. This process is then repeated at the lowest specified supply voltage.

FIGURE 1: PROGRAMMER SCHEMATIC



SOFTWARE DESCRIPTION

The listed code provides a hardware-software interface to a standard PC parallel (Centronix) interface port. The code can be adapted to a microprocessor parallel interface port by substituting an output command for the “biosprint” command.

Control software can transfer the PIC16C84 program, configuration bits, and EEPROM data from a standard PROM interface file into the target system by reading the file and calling the function in Figure 2 using the appropriate command name in the definition table, and the data to be programmed. The command names are repeated here for reference.

LOAD_CONFIG	Sets PIC16C84 data pointer to configuration.
LOAD_DATA	Loads, but does not program, data.
READ_DATA	Reads data at current pointer location.
INC_ADDR	Increments PIC16C84 data pointer.
BEGIN_PROG	Programs data at current data pointer location.
PARALLEL_MODE	Puts PIC16C84 into parallel mode. (not used)
LOAD_DATA_DM	Loads EEPROM data.
READ_DATA_DM	Reads EEPROM data.

Function “int ser_pic16c84(<command>,<data [or 0]>) is called to preform command. Function returns internal data after read commands.

Do not forget to initiate the programming mode before programming, increment the addresses after each byte is programmed, and put the programmer in run mode after programming.

Designed by: Analog Design Specialist, Inc.
P.O. Box 26-0846
Littleton, CO 80126

3

EXAMPLE 1: PUT TARGET SYSTEM INTO PROGRAM MODE.

```
.. program code..
ser_pic16c84(PROGRAM_MODE,0);
.. program code..
```

EXAMPLE 2: READ DATA FROM THE TARGET SYSTEM

```
.. program code..
data = ser_pic16c84(READ_DATA,0); // read data
// transfers data from target part to variable “data”.
.. more program code ..
```

EXAMPLE 3: PROGRAM DATA INTO THE TARGET SYSTEM

```
.. program code..
ser_pic16c84(LOAD_DATA,data);    // load data into target
ser_pic16c84(BEGIN_PROG,0);      // program loaded data
ser_pic16c84(INC_ADDR,0);        // increment to next address
 // transfers data from program variable “data” to target
     part.
.. more program code ..
```

EXAMPLE 4: PUT TARGET SYSTEM INTO RUN MODE

```
.. program code..
ser_pic16c84(RUN,0);
.. program code..
```

```
//************************** FIGURE #2 ********************************
//**                                                                 **
//**   SERIAL PROGRAMMING ROUTINE FOR THE PIC16C84 MICROCONTROLLER   **
//**                                                                 **
//**                    Analog Design Specialists                    **
//**                                                                 **
//*********************************************************************

//FUNCTION PROTOTYPE: int ser_pic16c84(int cmd, int data)

// cmd: LOAD_CONFIG   -> part configuration bits
//      LOAD_DATA     -> program data, write
//      READ_DATA     -> program data, read
//      INC_ADDR      -> increment to the next address (routine does not auto increment)
//      BEGIN_PROG    -> program a previously loaded program code or data
//      LOAD_DATA_DM  -> load EEPROM data regesters (BEGIN_PROG must folow)
//      READ_DATA_DM  -> read EEPROM data
//
// data: 1) 14 bits of program data or
//       2)  8 bits of EEPROM data (least significant 8 bits of int)

// Additional programmer commands (not part of PIC16C84 programming codes)
//
// cmd: RESET         -> provides 1 ms reset pulse to target system
//      PROGRAM_MODE  -> initializes PIC16C84 for programming
//      RUN           -> disconnects programmer from target system
//
// function returns:1) 14 or 8 bits read back data for read commands
//                  2) zero                       for write data commands
//                  3) PIC_PROG_EROR = -1 for programming function errors
//                  4) PROGMR_ERROR  = -2 for programmer function errors


#include <bios.h>

#define LOAD_CONFIG    0
#define LOAD_DATA      2
#define READ_DATA      4
#define INC_ADDR       6
#define BEGIN_PROG     8
#define PARALLEL_MODE 10  // not used
#define LOAD_DATA_DM   3
#define READ_DATA_DM   5
#define MAX_PIC_CMD   63  // division between pic and programmer commands

#define RESET         64  // external reset command, not neded for programming
#define PROGRAM_MODE  65  // initialize program mode
#define RUN           66  // electrically disconnect programmer

#define PIC_PROG_EROR -1
#define PROGMR_ERROR  -2

#define PTR            0  // use device #0

// parallel port bits
//      d0: data output to part to be programmed
//      d1: programming clock
//      d2: data dirrection, 0= enabel tri state buf -> send data to part
//      d3: Vpp control 1= turn on Vpp
//      d4: ~MCLR =0, 1 = reset device with MCLR line
//      d5: clock line tri state control, 0 = enable clock line

int ser_pic16c84(int cmd, int data)          // custom interface for pic 16c84
  {
  int i, s_cmd;

  if(cmd <=MAX_PIC_CMD)                      // all programming modes
    {
    biosprint(0,8,PTR);                      // set bits 001000, output mode, clock & data low
```

```
    s_cmd = cmd;                                    // retain command “cmd”
    for (i=0;i<6;i++)                               // output 6 bits of command
      {
      biosprint(0,(s_cmd&0x1) +2+8,PTR);            // set bits 001010, clock hi
      biosprint(0,(s_cmd&0x1)   +8,PTR);            // set bits 001000, clock low
      s_cmd >>=1;
      }

    if((cmd ==INC_ADDR)||(cmd ==PARALLEL_MODE)      // command only, no data cycle
      return 0;

    else if(cmd ==BEGIN_PROG)                       // program command only, no data cycle
      {
      delay(10);                                    // 10 ms PIC programming time
      return 0;
      }

    else if((cmd ==LOAD_DATA)||(cmd ==LOAD_DATA_DM)||(cmd ==LOAD_CONFIG)) // output 14 bits of
data
      {
      for (i=200;i;i—) ;                            // delay between command & data
      biosprint(0,2+8,PTR);                         // set bits 001010, clock hi; leading bit
      biosprint(0,  8,PTR);                         // set bits 001000, clock low

      for (i=0;i<14;i++)                            // 14 data bits, lsb first
       {
       biosprint(0,(data&0x1) +2+8,PTR);            // set bits 001010,  clock hi
       biosprint(0,(data&0x1)   +8,PTR);            // set bits 001000,  clock low
       data >>=1;
       }
      biosprint(0,2+8,PTR);                         // set bits 001010, clock hi; trailing bit

   // *****************  Analog Design Specialists  ******************


      biosprint(0,  8,PTR);                         // set bits 001000, clock low

      return 0;
      }

    else if((cmd ==READ_DATA)||(cmd ==READ_DATA_DM)) //read 14 bits from part, lsb first
      {
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low, tri state data
                                                           buffer
      for (i=200;i;i—) ;                            // delay between command & data
      biosprint(0,2+4+8,PTR);                       // set bits 001110, clock hi, leading bit
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low

      data =0;
      for (i=0;i<14;i++)                            // input 14 bits of data, lsb first
       {
       data >>=1;                                   // shift data for next input bit
       biosprint(0,2+4+8,PTR);                      // set bits 001110,  clock hi
       biosprint(0,  4+8,PTR);                      // set bits 001100,  clock low
       if(!(biosprint(2,0,0)&0x40)) data += 0x2000; //use printer acknowledge line for input,
                                                           data lsb first
       }
      biosprint(0,2+4+8,PTR);                       // set bits 001110, clock hi, trailing bit
      biosprint(0,  4+8,PTR);                       // set bits 001100, clock low
      return data;
      }

    else return PIC_PROG_EROR;                      // programmer error

    }
  else if(cmd == RESET)                             // reset device
    {
    biosprint(0,32+16+4,PTR);                       // set bits 110100, MCLR = low (reset
                                                           PIC16C84), programmer not conected
```

```
  delay(1);                                  // 1ms delay
  biosprint(0,32   +4,PTR);                  // set bits 100100, MCLR = high
  return 0;
  }

else if(cmd ==PROGRAM_MODE)                  // enter program mode
  {
  biosprint(0,32+16+4,PTR);                  // set bits 110100, Vpp off, MCLR = low
                                                  (reset PIC16C84)
  delay(10);                                 // 10 ms, allow programming voltage to
                                                  stabelize

  biosprint(0,8,PTR);                        // set bits 001000, Vpp on , MCLR = 13.5
                                                  volts, clock & data connected
  delay(10);                                 // 10 ms, allow programming voltage to
                                                  stablize

  return 0;
  }

else if(cmd ==RUN)                           // disconects programmer from device
  {
  biosprint(0,32+4,PTR);                     // set bits 100100
  return 0;
  }
else return PROGMR_ERROR;                    // command error
}
```

AN594

Using the CCP Modules

This application note discusses the operation of a Capture Compare and PWM (CCP) module, and the interaction of multiple CCP modules with the timer resources.

The Capture Compare and PWM (CCP) module is software programmable to operate in one of three modes:

1. A Capture input
2. A Compare output
3. A Pulse Width Modulation (PWM) output

For the CCP module to function, Timer resources must be used in conjunction with the CCP module. The desired CCP mode of operation determines which timer resources are required. Table 1 shows the CCP mode with the corresponding timer resource required. Both the Capture and Compare modes require that Timer 1 be operating in timer mode or synchronized counter mode.

Note: Capture and Compare modes may not operate if Timer1 is operated in asynchronous counter mode.

TABLE 1: CCP MODE - TIMER RESOURCE

CCP Mode	Timer Resource
Capture	Timer 1
Compare	Timer 1
PWM	Timer 2

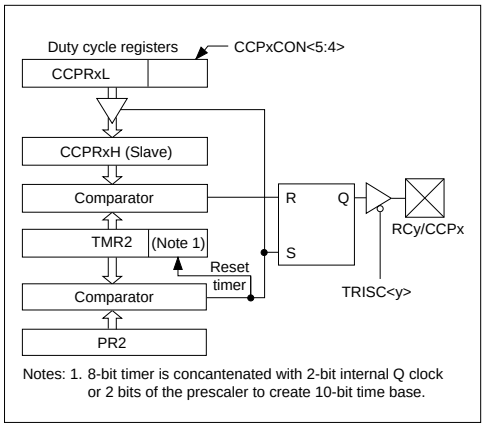
CCP OPERATION

The following three sections discuss the operation of the CCP module in each of its modes of operation. There is a simple example program for each mode of operation. The software example for the capture mode, also uses a second CCP module in compare mode to generate the signal to capture.

PWM Mode

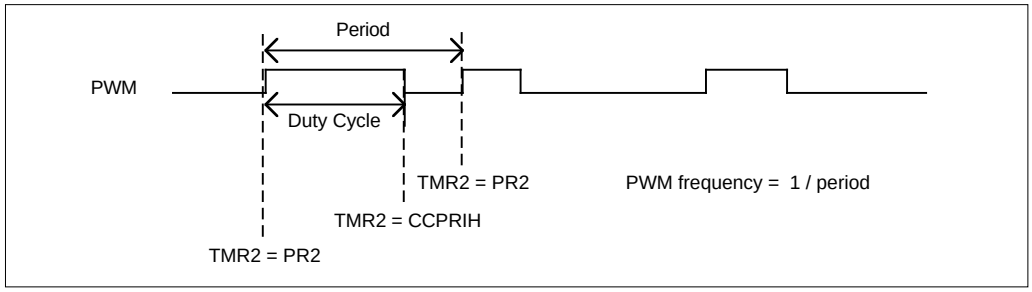
A Pulse Width Modulation output (shown in Figure 1) is a signal that has a timebase (period) and a time that the output stays high (duty cycle). The period is the duration after which the PWM rising edge repeats itself. The resolution of the PWM output is the granularity with which the duty cycle can be varied. The frequency of a PWM is simply the inverse of the period (1 / period).

FIGURE 2: PWM MODE BLOCK DIAGRAM



Notes: 1. 8-bit timer is concantenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time base.

FIGURE 1: PWM OUTPUT



Using the CCP Modules

Each CCP module can support one Pulse Width Modulation (PWM) output signal, with minimal software overhead. This PWM signal can attain a resolution of up to 10-bits, from the 8-bit Timer 2 module. This gives 1024 steps of variance from an 8-bit overflow counter. This gives a maximum accuracy of Tosc (50 ns, when the device is operated at 20 MHz). Figure 2 shows a block diagram of the CCP module in PWM mode. When the Timer 2 overflows (timer = Period Register), the value in the duty cycle registers (CCPRxL:CCPRxCON<5:4>) is latched into the 10-bit slave latch. A new duty cycle value can be loaded into the duty cycle register(s) at any time, but is only latched into the slave latch when Timer 2 = Timer 2 Period Register (PR2).

The period of Timer 2 (and PWM) is determined by the frequency of the device, the Timer 2 prescaler value (1, 4 or 16), and the Timer 2 Period Register. Equation 1 shows the calculation of the PWM period, duty cycle, and the minimum and maximum frequncies.

EQUATION 1: PWM PERIOD, DUTY CYCLE, AND FREQUNCIES

$$\text{PWM Period} = [(\text{PR2}) + 1) \bullet 4\ T_{OSC} \bullet (\text{Timer 2 prescale value})$$

$$\text{PWM Duty Cycle} = [\text{CCPRxL:CCPRxCON<5:4>}] \bullet 4\ T_{OSC} \bullet (\text{Timer 2 prescale value})$$

PWM maximum frequency

$$(\text{High Resolution mode}) = 4 / (\text{PR2} \bullet T_{CY})$$

$$(\text{Low Resolution mode}) = 1 / (\text{PR2} \bullet T_{CY})$$

PWM minimum frequency

$$(\text{High Resolution mode}) = 4 / (\text{PR2} \bullet 16 \bullet T_{CY})$$

$$(\text{Low Resolution mode}) = 1 / (\text{PR2} \bullet 16 \bullet T_{CY})$$

Table 2 shows the minimum and maximum PWM frequency for different device frequencies. The Timer2 prescaler will be selected to give either the minimum or maximum frequencies as shown.

Appendix A is a program which generates up to a 10-bit PWM output. The PWM period and duty cycle are updated after the overflow of Timer1. Upon the overflow of Timer1, ports A, B and D are read. The 10-bit duty cycle is specified by the value on PORTB:PORTA<1:0>, while the period is specified by the value on PORTD. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

Since the PWM duty cycle is double buffered, the duty cycle registers are only loaded when there is sufficient time to complete the update the 10-bit value before the Timer2 = PR2 match occurs. After the duty cycle has been updated and the Timer2 = PR2 match has occurred, the period (stored in the PR2 register) is updated. The operation of the CCP module in PWM mode is similar to the PIC17C42's PWM. Additional concepts of PWM operation can be found in Application Notes AN564 and AN539.

TABLE 2: PWM FREQUENCY FOR DIFFERENT DEVICE FREQUENCIES

	20 MHz		10 MHz		2 MHz		
PWM Resolution	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Units**
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	KHz
9-Bit	1.22	39.06	0.613	9.77	0.123	3.92	KHz

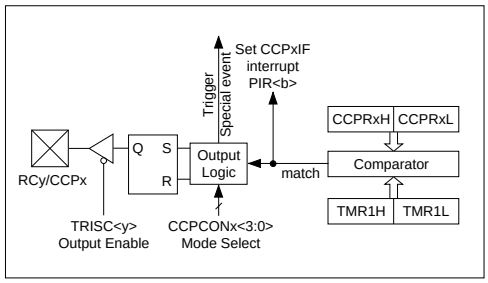
Compare Mode

In compare mode, the 16-bit value of Timer1 is compared to the CCPRxH:CCPRxL registers. When these registers match, the S/W configured event occurs on the CCPx pin. The events that can be S/W selected are:

- Clear CCPx pin on match
- Set CCPx pin on match
- Generate S/W interrupt (CCPx pin unchanged)
- Trigger special event (CCPx pin unchanged)
  - CCP1 clears Timer1
  - CCP2 clears Timer1 and sets the A/D's GO bit

The CCPxM<3:0> control bits, in register CCPxCON, configures the operation of the CCP module. The compare function must have the data direction of the CCPx pin configured as an output, if the compare event is to control the state of the CCPx pin.

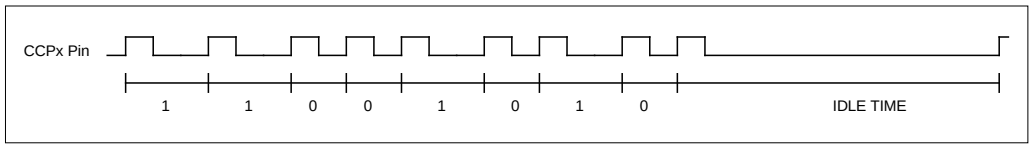
FIGURE 3: COMPARE MODE BLOCK DIAGRAM



When the CCP module is in the OFF state (CCPxM<3:0> = 0h), the CCPx output latch is forced to a low level, though the level on the CCPx pin will be determined by the value in the data latch of the port. Figure 3 shows the block diagram of the CCP module in Compare mode.

Appendix B is a program which uses the CCP module to transmit a pulse train dependent on the data byte. Timer1 is used as a free running timer, with each "new" compare value being an offset added to the present CCP compare latch value. The data is transmitted every 600 µs. Each data bit has a sync pulse (High level) of 8.8 µs. Then the data is transmitted as a low pulse. The time duration of the low pulse determines the value of the data bit. A '0' bit is low for 18.8 µs while a '1' bit is low for 37.6 µs. After the last data bit has been transmitted, another sync pulse is transmitted and the output remains low (idle time) until the 600 µs data period has completed. An example of the pulse train for the a data byte of CAh is shown in Figure 4, and has an idle time of 224 µs. These pulse times are based off the device operational frequency. The program header file, COMP.H, calculates the values to loaded into the compare registers from the specified Device_freq. The data to be transmitted is read from PORTB, during the idle time. By setting the conditional assemble flag`PICMaster` to TRUE, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

FIGURE 4: TRANSMIT PULSE TRAIN (DATA = 0X0CA)



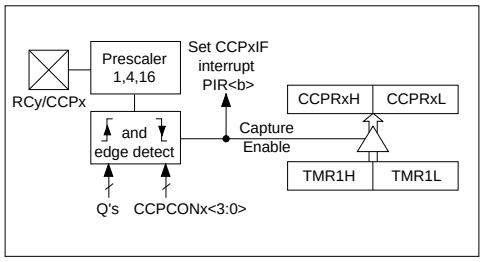
Capture Mode

In capture mode, the 16-bit value of Timer1 is latched into the CCPRxH:CCPRxL registers, when the S/W configured event occurs on the CCPx pin. The events that can cause a capture are:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

The CCPxM<3:0> control bits, in register CCPxCON, configures the operation of the CCP module. The capture function works regardless of the data direction of the CCPx pin (input or output). With the CCPx pin is configured as an output, a write to the CCPx pin (in PORTC) will cause a capture when the capture requirement is met.

FIGURE 5: CAPTURE MODE BLOCK DIAGRAM

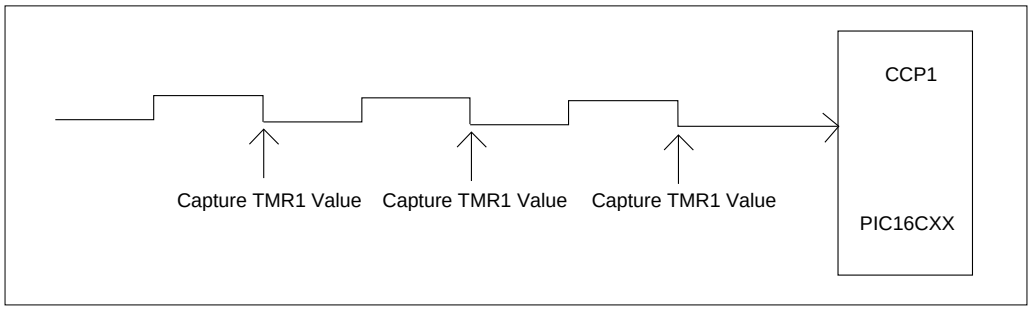


The changing of the capture mode, via the CCPxM<3:0> bits, may cause the CCPxIF bit to be set. This "false" interrupt should be cleared (ignored) after changing between capture modes. The CCP prescaler is only cleared by configuring the CCP module into the OFF state (CCPxM<3:0> = 0h). Figure 5 shows the block diagram of the CCP module in Capture mode. The utilization of the CCP module in capture mode is similar to the PIC17C42's capture. Additional concepts of capture operation can be found in Application Note AN545.

Appendix C is a program which implements a 16-bit capture from a free running timer (TMR1). The capture event is configured as each rising edge. The 16-bit capture value is the "new" 16-bit capture value minus the "old" 16-bit capture value. If the time between captures is greater than 2^{16} Timer1 increments, an invalid result will occur. This invalid result is not indicated by the software. After the capture period result is calculated, the "new" capture value is loaded into the "old" register.

The waveform that is captured is generated from a second CCP module in compare mode. The value that is loaded in to the CCPR2H:CCPR2L is read from the PORTB and PORTD registers. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus. Figure 6 shows an input into the CCPx pin, and the capture measurement points.

FIGURE 6: EXAMPLE CAPTURE WAVE FORM



INTERACTION OF CCP MODULES

Due to the modularity of the PIC16CXX peripherals, future devices with two or more CCP modules on a device are possible. Each CCP module operates independently from the others, though their interaction with the timer resources must be taken into account.

When two or more CCP modules exist on a device, there can be an interaction between the CCP modules. This interaction is shown in Table 3. These interactions do NOT include any interaction (S/W) caused by the main program nor the interrupt service routines of the CCP sources.

Interaction of Two Capture Modes

When two CCP modules are in a Capture mode, Timer1 is the timebase for both captures. This means that they will have the same capture resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the device.

Interaction of One Capture Mode and One Compare Mode

When one CCP module is in a Capture mode and a second CCP module is in Compare mode, Timer1 is the timebase for both the captures and the compare. This means that the capture and the compare will have the same resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Also, care must be taken in that the compare can be configured to clear TMR1 (when in special Trigger mode). Care must be taken in system design to ensure that this clearing of the TMR1 does not have any negative impact on the capture function.

Interaction of Two Compare Modes

When two CCP modules are in a Compare mode, Timer1 is the timebase for both compares. This means that they will have the same compare resolution, as determined by the TMR1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Since the compare modules can be configured to clear TMR1 (when in special Trigger mode), care must be taken in system design to ensure that this clearing of the TMR1 does not have any negative impact on the compare function. If both compares are configured with a special trigger, which clears the TMR1, then the compare register that is closest to (but greater than) the TMR1 value is the compare value that will reset TMR1. Example 1 shows a possible case.

TABLE 3: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 timebase.
Capture	Compare	The compare could be configured for trigger special event, which clears TMR1.
Compare	Compare	The compare(s) could be configured for trigger special event, which clears TMR1.
PWM	PWM	The PWMs will have the same frequency, and update rate (TMR2 interrupt).
PWM	Capture	None
PWM	Compare	None

EXAMPLE 1:

ACTION	TIMER 1 STATE	COMMENT
CCPR1H:CCPR1L = 0x0465	0x????	
CCP1CON = 0x?B	0x????	CCP1 in Compare - Special
:		Trigger Mode
:	0x0232	
:		
CCPR2H:CCPR2L = 0x0165	0x0333	
CCP2CON = 0x?B	0x0334	CCP2 in Compare - Special
:		Trigger Mode
	0x0465	CCP1 resets TMR1 and CCP1 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		

Interaction of Two PWM Modes

When two CCP modules are in a PWM mode, Timer2 is the timebase for both PWM outputs. This means that they will have the same PWM frequency and update rates, as determined by the TMR2 prescaler and frequency of the device. The resolution of the two PWMs may be different, since each CCP module has its own CCPxX:CCPxY bits for high resolution mode. These bits are found in the CCPxCON<5:4> register.

CONCLUSION

The Capture / Compare / PWM modules offer enormous flexibility in the use of the device timer resources. As with all resources, care must be taken to ensure that no adverse system complications can occur with the interaction between multiple CCP modules. The programs for simple operation of the various CCP modes should be a good foundation for modifications to suite your particular needs.

Written by: Mark Palmer - Sr. Application Engineer
Logic Products Division

APPENDIX A: PWM_1.LST

```
MPASM 01.01 Released   PWM_1.ASM  7-13-1994  14:26:3          PAGE 1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST   P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program outputs a PWM signal on the CCP1 pin. The duty cycle and period
                     0006 ; of the PWM is read every time TMR1 overflows.
                     0007 ;       PERIOD     = PORTB
                     0008 ;       DUTY CYCLE = PORTD and PORTE<1:0>
                     0009 ;
                     0010 ; The prescaler of TMR2 is selected by the state of PORTA<1:0> after reset
                     0011 ;       RA1:RA0    Prescaler multiplies Tcyc by
                     0012 ;         0 0          1
                     0013 ;         0 1          4
                     0014 ;         1 x         16
                     0015 ;
                     0016 ;
                     0017 ;       Program = PWM_1.ASM
                     0018 ;       Revision Date:   7-13-94
                     0019 ;
                     0020 ;*****************************************************************************
                     0021 ;
                     0022 ;
                     0023 ; HARDWARE SETUP
                     0024 ;       PORTA<1:0>   -  Prescaler to TMR2, read only after reset
                     0025 ;       PORTB        -  Period of PWM
                     0026 ;       PORTD        -  Duty Cycle high of PWM (8-bits)
                     0027 ;       PORTE<1:0>   -  Duty Cycle low of PWM (2-bits)
                     0028 ;
                     0029 ;
                     0030                INCLUDE <C74_reg.h>
                     0247
                     0030
                     0031                INCLUDE <PWM.h>
                     0047
                     0031
                     0032 ;
                     0033 PICMaster      EQU     TRUE            ; A Debugging Flag
                     0034 Debug          EQU     TRUE            ; A Debugging Flag
                     0035 Debug_PU       EQU     TRUE            ; A Debugging Flag
                     0036 ;
                     0037 ;
                     0038 ; Reset address. Determine type of RESET
```

```
0039 ;
0040         org     RESET_V                 ; RESET vector location
0041 RESET   BSF     STATUS, RP0             ; Bank 1
0042         BTFSC   PCON, POR               ; Power-up reset?
0043         GOTO    START                   ; YES
0044         GOTO    OTHER_RESET             ; NO, a WDT or MCLR reset
0045 ;
0046 ; This is the Periperal Interrupt routine. Need to determine the type
0047 ; of interrupt that occurred. The following interrupts are enabled:
0048 ;   1.  CCP Capture Occured
0049 ;
0051         org     ISR_V                   ; Interrupt vector location
0052 PER_INT_V
0053         BCF     STATUS, RP0             ; Bank 0
0054         BTFSC   PIR1, TMR1IF            ; TMR1 Overflow Interrupt occured?
0055         GOTO    T1OVFL                  ; YES, Service the TMR1 Interrupt
0056 ERROR1                                 ; NO, Error Condition - Unknown Interrupt
0057         BSF     PORTA, 2                ; Toggle a PORT pin
0058         BCF     PORTA, 2
0059         GOTO    ERROR1
0060 ;
0061 ERROR2                                 ; NO, Error Condition - Unknown Interrupt
0062         BSF     PORTA, 3                ; Toggle a PORT pin
0063         BCF     PORTA, 3
0064         GOTO    ERROR2
0065 ;
0066 T1OVFL
0067         BCF     PIR1, TMR1IF            ; Clear T1 Overflow Interrupt Flag
0068       if (PICMaster )
0069         MOVF    DUMMY_PD, W             ;
0070       else
0071         MOVF    PORTD, W                ;
0072       endif
0073         MOVWF   DC_HI                   ;
0074       if (PICMaster )
0075         MOVF    DUMMY_PE, W             ;
0076       else
0077         MOVF    PORTE, W                ;
0078       endif
0079         MOVWF   DC_LO                   ;
0080       if (PICMaster )
0081         MOVF    DUMMY_PB, W             ;
0082       else
0083         MOVF    PORTB, W                ;
0084       endif
0085         BSF     STATUS, RP0             ; Bank 1
0086         MOVWF   T2_PERIOD               ;
0087         BCF     STATUS, RP0             ; Bank 0
```

```
0000 1683
0001 188E
0002 2832
0003 285C

0004 1283
0005 180C
0006 280D

0007 1505
0008 1105
0009 2807

000A 1585
000B 1185
000C 280A

000D 100C

000E 0853

000F 00D5

0010 0854

0011 00D6

0012 0851

0013 1683
0014 00A0
0015 1283
```

```
0088 ;
0089 WAIT_DC
0090                    MOVF    TMR2, W         ; Read present TMR2 register value
0091                    SUBWF   PR2, W          ; How close is the timer to rolling over
0092                    ANDLW   0x0F            ; Does this make it zero?
0093                    BTFSC   STATUS, Z       ; If Z is set, near rollover
0094                    GOTO    WAIT_DC         ; loop until rolled over
0095                    MOVF    DC_HI, W        ; else losd the duty cycle values
0096                    MOVWF   CCPR1L          ; Load DC high
0097                    MOVLW   0x0F            ;
0098                    ANDWF   CCP1CON, F      ; Set the DC low bits
0099                    BTFSC   DC_LO, 1        ;
0100                    BSF     CCP1CON, CCP1X  ;
0101                    BTFSC   DC_LO, 0        ;
0102                    BSF     CCP1CON, CCP1Y  ;
0103                    BCF     PIR1, TMR2IF    ; Clear the TRM2 = PR2 flag
0104 ;
0105 WAIT_PR
0106                    BTFSS   PIR1, TMR2IF    ; LOOP waiting for TRM2 = PR2
0107                    GOTO    WAIT_PR         ; Need to wait until TMR2 = PR2 so that
0108                                            ;   Duty Cycle is latched
0109                    BSF     STATUS, RP0     ; Bank 1
0110                    MOVLW   0x0F            ; Load TMR2 period with minimum value Fh
0111                    MOVWF   PR2             ;
0112                    MOVLW   0xF0            ;
0113                    ANDWF   T2_PERIOD, W    ; Determine if period needs to be greater
0114                    BTFSC   STATUS, Z       ;
0115                    GOTO    NO_OFFSET       ; NO, Period is the minimum
0116 PR_OFFSET
0117                    MOVLW   0x0F            ; Yes, calculate additional offset
0118                    SUBWF   T2_PERIOD, W    ;
0119                    ADDWF   PR2, F          ; ADD Period offset
0120 ;
0121 NO_OFFSET
0122                    BCF     STATUS, RP0     ; Bank 0
0123                    RETFIE                  ; Return / Enable Global Interrupts
0124 ;
0126 ;
0127 ;*****************************************************************************
0128 ;*****     Start program here, Power-On Reset occurred.
0129 ;*****************************************************************************
0130 ;
0131 START                                       ; POWER_ON Reset (Beginning of program)
0132                    BCF     STATUS, RP0     ; Bank 0
0133                    CLRF    TMR1H           ;
0134                    CLRF    TMR1L           ;
0135 ;
0136 MCLR_RESET                                  ; A Master Clear Reset
```

```
0035 0183                       CLRF    STATUS                  ; Do initialization (Bank 0)
0036 018B                       CLRF    INTCON
0037 018C                       CLRF    PIR1
0038 1683                       BSF     STATUS, RP0             ; Bank 1
0039 3080                       MOVLW   0x80                    ;
003A 0081                       MOVWF   OPTION_R                ;
003B 018C                       CLRF    PIE1                    ; Disable all peripheral interrupts
003C 30FF                       MOVLW   0xFF                    ;
003D 009F                       MOVWF   ADCON1                  ; Port A is Digital.
                        ;               
                        ;               
003E 1283                       BCF     STATUS, RP0             ; Bank 0
003F 0185                       CLRF    PORTA                   ; ALL PORT output should output Low.
0040 0186                       CLRF    PORTB
0041 0187                       CLRF    PORTC
0042 0188                       CLRF    PORTD
0043 0189                       CLRF    PORTE
                        ;               
0044 1683                       BSF     STATUS, RP0             ; Select Bank 1
0045 30FF                       MOVLW   0xFF                    ;
0046 0085                       MOVWF   TRISA                   ; RA5 - 0 inputs
0047 0086                       MOVWF   TRISB                   ; RB7 - 0 inputs
0048 0187                       CLRF    TRISC                   ; RC Port are outputs
0049 0088                       MOVWF   TRISD                   ; RD Port are inputs
004A 0089                       MOVWF   TRISE                   ; RE Port are inputs
004B 0092                       MOVWF   PR2                     ; Default PWM period
004C 140C                       BSF     PIE1, TMR1IE            ; Enable TMR1 Interrupt
004D 1283                       BCF     STATUS, RP0             ; Select Bank 0
                        ;               
004E 300C                       MOVLW   0X0C                    ; CCP module is in
004F 0097                       MOVWF   CCP1CON                 ; PWM output mode
                        ;               
                        ; Initialize the Special Function Registers (SFR) interrupts
                        ;               
0050 018C                       CLRF    PIR1                    ;
0051 0190                       CLRF    T1CON                   ;
0052 0192                       CLRF    T2CON                   ;
                            if (PICMaster )
0053 1850                       BTFSC   DUMMY_PA, 0             ;
                            else
                                BTFSC   PORTA, 0                ;
                            endif
0054 1412                       BSF     T2CON, 0                ;
                        ;               
                            if (PICMaster )
0055 18D0                       BTFSC   DUMMY_PA, 1             ;
                            else
                                BTFSC   PORTA, 1                ;
```

```
                        endif
0056 1492                   BSF     T2CON, 1        ;
0187 ;
0057 170B                   BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0058 178B                   BSF     INTCON, GIE     ; Enable all Interrupts
0059 1410                   BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
005A 1512                   BSF     T2CON, TMR2ON   ; Turn Timer 2 ON
0191 ;
0192 ;
005B 285B lzz       goto    lzz             ; Loop waiting for TMR1 interrupt
0194 ;
0195 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
0196 ;
005C 1E03 OTHER_RESET   BTFSS  STATUS, TO    ; WDT Time-out?
005D 2807 WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
0199                    if ( Debug_PU )
005E 2832                   goto    START           ; MCLR reset, Goto START
0201                    else
0202                        GOTO   MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
0203                    endif
0204 ;
0205                    if (Debug )
005F 0000 END_START     NOP                    ; END lable for debug
0207                    endif
0208 ;
0209 ;
0210                    org     PMEM_END       ; End of Program Memory
07FF 2807                   GOTO    ERROR1         ; If you get here your program was lost
0212
0213                end
0214
0215


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ________________ ________________

0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ _______X

All other memory blocks unused.

Errors   :     0
Warnings :    17
```

APPENDIX A2: PWM.H

```
       nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:       CLOCK.H
;       Revision:      7-13-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq              EQU           D'10000000'                  ; Device Frequency is 4 MHz
PULSE_TIME      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
;

DB_HI_BYTE            EQU           (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY EQU           (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR            EQU           40                                    ; RAM Location
OUTER_CNTR            EQU           41                                    ; RAM Location
;
T1OSO                 EQU           0                            ; The RC0 / T1OSO / T1CKI
;
RESET_V               EQU           0x0000               ; Address of RESET Vector
ISR_V                 EQU           0x0004               ; Address of Interrupt Vector
PMEM_END              EQU           0x07FF               ; Last address in Program Memory
TABLE_ADDR            EQU           0x0400               ; Address where to start Tables
;
COUNTER         EQU     0x021          ;
;
XMIT_DATA       EQU     0x30
DATA_CNT        EQU     0x31
ONES_CNT        EQU     0x32
CCP1_INT_CNT    EQU     0x33
CCPREG_HI       EQU     0x40
CCPREG_LO       EQU     0x41
DUMMY_PA        EQU     0x50
DUMMY_PB        EQU     0x51
DUMMY_PC        EQU     0x52
DUMMY_PD        EQU     0x53
DUMMY_PE        EQU     0x54
DC_HI           EQU     0x55
DC_LO           EQU     0x56
T2_PERIOD       EQU     0xA0
;
       list
```

APPENDIX B: COMP_1.LST

```
MPASM 01.01 Released    COMP_1.ASM   7-13-1994  9:7:22                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program outputs a pulse train on the CCP1 pin, as specified by the
                     0006 ; values in the CCPR1H:CCPR1L.
                     0007 ;
                     0008 ;
                     0009 ;  Pulse Train     ___|   |_____|   |_____|
                     0010 ;  Data Value          <-- 1 --> <-- 0 -->
                     0011 ;                      ^       ^       ^
                     0012 ;                      |       |       |
                     0013 ;                      | T_ONE_BIT     | T_ZERO_BIT
                     0014 ;                      |               |
                     0015 ;                      PULSE_TIME      PULSE_TIME
                     0016 ;
                     0017 ;
                     0018 ;       Program = COMP_1.ASM
                     0019 ;       Revision Date:  7-13-94
                     0020 ;
                     0021 ;*****************************************************************************
                     0022 ;
                     0023 ;
                     0024 ; HARDWARE SETUP
                     0025 ;    PORTB       -  Data to serial transmit on CCP pin
                     0026 ;
                     0027 ;
                     0028         INCLUDE <C74_reg.h>
                     0247
                     0028
                     0029         INCLUDE <COMP.h>
                     0053
                     0029
                     0030 ;
0001                 0031 PICMaster       EQU     TRUE            ; A Debugging Flag
0001                 0032 Debug           EQU     TRUE            ; A Debugging Flag
0001                 0033 Debug_PU        EQU     TRUE            ; A Debugging Flag
                     0034 ;
                     0035 ;
                     0036 ; Reset address. Determine type of RESET
                     0037 ;
                     0038         org     RESET_V                 ; RESET vector location
```

```
0039 RESET          BSF     STATUS, RP0     ; Bank 1
0040                BTFSC   PCON, POR       ; Power-up reset?
0041                GOTO    START           ; YES
0042                GOTO    OTHER_RESET     ; NO, a WDT or MCLR reset
0043 ;
0044 ; This is the Periperal Interrupt routine. Need to determine the type
0045 ; of interrupt that occurred. The following interrupts are enabled:
0046 ;   1.  CCP Capture Occured
0047 ;
0049        org     ISR_V               ; Interrupt vector location
0050 PER_INT_V
0051        if ( Debug )
0052            bsf     PORTA, 0        ; Turn on strobe
0053        endif                       ;
0054            BCF     STATUS, RP0     ; Bank 0
0055            BTFSC   PIR1, CCP1IF    ; Compare Interrupt occured?
0056            GOTO    CCP1_INT        ; YES, Service the TMR1 Interrupt
0057 ERROR1                             ; NO, Error Condition - Unknown Interrupt
0058            BSF     PORTA, 2        ; Toggle a PORT pin
0059            BCF     PORTA, 2
0060            GOTO    ERROR1
0061 ;
0062 ERROR2                             ; NO, Error Condition - Unknown Interrupt
0063            BSF     PORTA, 3        ; Toggle a PORT pin
0064            BCF     PORTA, 3
0065            GOTO    ERROR2
0066 ;
0067 ;*****************************************************************************
0068 ; In the CCP interrupt.
0069 ; Since timer1 is not cleared on a CCP match, the value in the
0070 ; CCPR1H:CCPR1L register pair must be updated. This is done with
0071 ; a 16-bit add. Also after the 1st CCP1 match (CCP1 pin goes high)
0072 ; the next match will force it low. Depending on the value of the data bit
0073 ; determines the value add to the CCPR1H:CCPR1L register pair.
0074 ;
0075 ; After the data has been transmitted, the pin will have a sync pulse and
0076 ; then remain low for 300 us.
0077 ;*****************************************************************************
0078 ;
0079 ;
0080
0081 CCP1_INT
0082            BCF     PIR1, CCP1IF    ; Clear CCP1 Interrupt Flag
0083            INCF    CCP1_INT_CNT    ;
0084            BTFSS   CCP1_INT_CNT, 0 ;
0085            GOTO    SYNC_PULSE
0086 DATA_PULSE
0087            DECF    DATA_CNT        ; Decrement the Count of data bits

0000 1683
0001 188E
0002 287C
0003 28BA

0004 1405

0005 1283
0006 190C
0007 280E

0008 1505
0009 1105
000A 2808

000B 1585
000C 1185
000D 280B

000E 110C
000F 0AB3
0010 1C33
0011 2833

0012 03B1
```

```
0013 1903           BTFSC   STATUS, Z           ; Have we transmitted all the Data Bits?
0014 2827           GOTO    PERIOD_DELTA        ; YES, Delay to 300 us
0015 0DB0           RLF     XMIT_DATA, F        ; NO, get next bit to transmit
0016 1803           BTFSC   STATUS, C           ; Is the bit to transmit a '1'?
0017 281F           GOTO    ONE_DATA            ; YES, Stay low for 17.6 us
0093 ZERO_DATA
0094                MOVLW   LOW ( T_ZERO_BIT )  ; NO, Stay low for 8.8 us
0095                ADDWF   CCPR1L, F           ; Update Compare register pair latch
0096                BTFSC   STATUS, C           ;
0097                INCF    CCPR1H, F           ;
0098                MOVLW   HIGH (T_ZERO_BIT )  ;
0099                ADDWF   CCPR1H, F           ;
0100                GOTO    RET_FIE             ;
0101 ;
0102 ONE_DATA
0103                MOVLW   LOW ( T_ONE_BIT )   ; Stay low for 17.6 us
0104                ADDWF   CCPR1L, F           ; Update Compare register pair latch
0105                BTFSC   STATUS, C           ;
0106                INCF    CCPR1H, F           ;
0107                MOVLW   HIGH ( T_ONE_BIT )  ;
0108                ADDWF   CCPR1H, F           ;
0109                INCF    ONES_CNT            ; Increment the number of '1's in the byte
0110                GOTO    RET_FIE             ;
0111 ;
0112 PERIOD_DELTA
0113                MOVF    ONES_CNT, W         ;
0114                ANDLW   0x0F                ; Only want 9 states (0 1s to 8 1s)
0115                ADDWF   PCL, F              ;
0116                GOTO    ZERO_1              ; There was 0 ones in the data byte
0117                GOTO    ONE_1               ; There was 1 one in the data byte
0118                GOTO    TWO_1               ; There was 2 ones in the data byte
0119                GOTO    THREE_1             ; There was 3 ones in the data byte
0120                GOTO    FOUR_1              ; There was 4 ones in the data byte
0121                GOTO    FIVE_1              ; There was 5 ones in the data byte
0122                GOTO    SIX_1               ; There was 6 ones in the data byte
0123                GOTO    SEVEN_1             ; There was 7 ones in the data byte
0124                GOTO    EIGHT_1             ; There was 8 ones in the data byte
0125 ;
0126 SYNC_PULSE
0127                MOVLW   LOW ( PULSE_TIME )  ; Update Compare register pair latch
0128                ADDWF   CCPR1L, F           ;
0129                BTFSC   STATUS, C           ;
0130                INCF    CCPR1H, F           ;
0131                MOVLW   HIGH ( PULSE_TIME ) ;
0132                ADDWF   CCPR1H, F           ;
0133                BSF     CCP1CON, 0          ; On Compare match, CCP1 pin = L
0134                RETFIE
0135 ;
```

```
0136 ZERO_1
0137
0138
0139
0140
0141
0142
0143 ;
0144 ONE_1
0145
0146
0147
0148
0149
0150
0151
0152 ;
0153 TWO_1
0154
0155
0156
0157
0158
0159
0160
0161 ;
0162 THREE_1
0163
0164
0165
0166
0167
0168
0169
0170 ;
0171 FOUR_1
0172
0173
0174
0175
0176
0177
0178
0179 ;
0180 FIVE_1
0181
0182
0183

003B 30EC    MOVLW   LOW ( ZERO_1S )   ; Update Compare register pair latch
003C 0795    ADDWF   CCPR1L, F         ;
003D 1803    BTFSC   STATUS, C         ;
003E 0A96    INCF    CCPR1H, F         ;
003F 3002    MOVLW   HIGH ( ZERO_1S )  ;
0040 0796    ADDWF   CCPR1H, F         ;
0041 287A    GOTO    RET_FIE

0042 30BD    MOVLW   LOW ( ONE_1S )    ; Update Compare register pair latch
0043 0795    ADDWF   CCPR1L, F         ;
0044 1803    BTFSC   STATUS, C         ;
0045 0A96    INCF    CCPR1H, F         ;
0046 3002    MOVLW   HIGH ( ONE_1S )   ;
0047 0796    ADDWF   CCPR1H, F         ;
0048 287A    GOTO    RET_FIE

0049 308E    MOVLW   LOW ( TWO_1S )    ; Update Compare register pair latch
004A 0795    ADDWF   CCPR1L, F         ;
004B 1803    BTFSC   STATUS, C         ;
004C 0A96    INCF    CCPR1H, F         ;
004D 3002    MOVLW   HIGH ( TWO_1S )   ;
004E 0796    ADDWF   CCPR1H, F         ;
004F 287A    GOTO    RET_FIE

0050 305F    MOVLW   LOW ( THREE_1S )  ; Update Compare register pair latch
0051 0795    ADDWF   CCPR1L, F         ;
0052 1803    BTFSC   STATUS, C         ;
0053 0A96    INCF    CCPR1H, F         ;
0054 3002    MOVLW   HIGH ( THREE_1S ) ;
0055 0796    ADDWF   CCPR1H, F         ;
0056 287A    GOTO    RET_FIE

0057 3030    MOVLW   LOW ( FOUR_1S )   ; Update Compare register pair latch
0058 0795    ADDWF   CCPR1L, F         ;
0059 1803    BTFSC   STATUS, C         ;
005A 0A96    INCF    CCPR1H, F         ;
005B 3002    MOVLW   HIGH ( FOUR_1S )  ;
005C 0796    ADDWF   CCPR1H, F         ;
005D 287A    GOTO    RET_FIE

005E 3001    MOVLW   LOW ( FIVE_1S )   ; Update Compare register pair latch
005F 0795    ADDWF   CCPR1L, F         ;
```

```
0060 1803               BTFSC   STATUS, C               ;
0061 0A96               INCF    CCPR1H, F               ;
0062 3002               MOVLW   HIGH ( FIVE_1S )        ;
0063 0796               ADDWF   CCPR1H, F               ;
0064 287A               GOTO    RET_FIE
0065                    ;
0066 SIX_1
0067 30D2               MOVLW   LOW ( SIX_1S )          ; Update Compare register pair latch
0068 0795               ADDWF   CCPR1L, F               ;
0069 1803               BTFSC   STATUS, C               ;
006A 0A96               INCF    CCPR1H, F               ;
006B 3001               MOVLW   HIGH ( SIX_1S )         ;
006C 0796               ADDWF   CCPR1H, F               ;
006D 287A               GOTO    RET_FIE
006E                    ;
006F SEVEN_1
0070 30A3               MOVLW   LOW ( SEVEN_1S )        ; Update Compare register pair latch
0071 0795               ADDWF   CCPR1L, F               ;
0072 1803               BTFSC   STATUS, C               ;
0073 0A96               INCF    CCPR1H, F               ;
0074 3001               MOVLW   HIGH ( SEVEN_1S )       ;
0075 0796               ADDWF   CCPR1H, F               ;
0076 287A               GOTO    RET_FIE
0077                    ;
0078 EIGHT_1
0079 3074               MOVLW   LOW ( EIGHT_1S )        ; Update Compare register pair latch
007A 0795               ADDWF   CCPR1L, F               ;
007B 1803               BTFSC   STATUS, C               ;
007C 0A96               INCF    CCPR1H, F               ;
007D 3001               MOVLW   HIGH ( EIGHT_1S )       ;
007E 0796               ADDWF   CCPR1H, F               ;
007F 287A               GOTO    RET_FIE
0080                    ;
0081 RET_FIE
0082 1017               BCF     CCP1CON, 0              ; On Compare match, CCP1 pin = H
0083 0009               RETFIE                          ; Return / Enable Global Interrupts
0084                    ;
0085                    ;*****************************************************************
0086                    ;*****   Start program here, Power-On Reset occurred.
0087                    ;*****************************************************************
0088                    ;
0089 START
008A 1283               BCF     STATUS, RP0             ; POWER_ON Reset (Beginning of program)
008B 018F               CLRF    TMR1H                   ; Bank 0
008C 018E               CLRF    TMR1L                   ;
008D                    ;                               ;
008E MCLR_RESET                                         ; A Master Clear Reset
```

```
007F 1283               0233         BCF     STATUS, RP0     ; Bank 0
0080 0183               0234         CLRF    STATUS          ; Do initialization (Bank 0)
0081 018B               0235         CLRF    INTCON
0082 018C               0236         CLRF    PIR1
0083 1683               0237         BSF     STATUS, RP0     ; Bank 1
0084 3080               0238         MOVLW   0x80            ; Disable PORTB weak pull-ups
0085 0081               0239         MOVWF   OPTION_R        ;
0086 018C               0240         CLRF    PIE1            ; Disable all peripheral interrupts
0087 30FF               0241         MOVLW   0xFF            ;
0088 009F               0242         MOVWF   ADCON1          ; Port A is Digital.
                        0243 ;
                        0244 ;
0089 1283               0245         BCF     STATUS, RP0     ; Bank 0
008A 0185               0246         CLRF    PORTA           ; ALL PORT output should output Low.
008B 0186               0247         CLRF    PORTB
008C 0187               0248         CLRF    PORTC
008D 0188               0249         CLRF    PORTD
008E 0189               0250         CLRF    PORTE
008F 1010               0251         BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                        0252 ;
0090 1683               0253         BSF     STATUS, RP0     ; Select Bank 1
0091 0185               0254         CLRF    TRISA           ; RA5 -  0 outputs
0092 30FF               0255         MOVLW   0xFF            ;
0093 0086               0256         MOVWF   TRISB           ; RB Port are inputs
0094 0187               0257         CLRF    TRISC           ; RC Port are outputs
0095 0188               0258         CLRF    TRISD           ; RD Port are outputs
0096 0189               0259         CLRF    TRISE           ; RE Port are outputs
0097 150C               0260         BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
0098 1283               0261         BCF     STATUS, RP0     ; Select Bank 0
                        0262 ;
                        0263 ;
                        0264 ;
                        0265 ; Initialize the Special Function Registers (SFR) interrupts
                        0266 ;
                        0267 ;
0099 018C               0268         CLRF    PIR1            ;
009A 0190               0269         CLRF    T1CON           ; Timer mode
009B 170B               0270         BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
009C 178B               0271         BSF     INTCON, GIE     ; Enable all Interrupts
                        0272 ;
                        0273 ; Set-up timer and compare latches and then turn timer1 on.
                        0274 ;
009D 1010               0275         BCF     T1CON, TMR1ON   ; Turn OFF timer1
009E 3041               0276         MOVLW   CCPREG_HI       ; TMR1 = CCPR1H:CCPR1L - 1
009F 008F               0277         MOVWF   TMR1H           ;
00A0 3042               0278         MOVLW   CCPREG_LO       ;
00A1 008E               0279         MOVWF   TMR1L           ;
00A2 038E               0280         DECF    TMR1L           ;
00A3 1803               0281         BTFSC   STATUS, C       ;
```

```
00A4 038F                       DECF    TMR1H           ;
00A5 3008                       MOVLW   0x08            ; On match CCP1 = H level
00A6 0097                       MOVWF   CCP1CON         ;
00A7 3009                       MOVLW   0x09            ;
00A8 00B1                       MOVWF   DATA_CNT        ; 8-bits to transfer
00A9 01B2                       CLRF    ONES_CNT        ; Result after xmit holds the number of 1's in a byte
00AA 30FF                       MOVLW   0xFF            ;
00AB 00B3                       MOVWF   CCP1_INT_CNT    ; No CCP1 transmit interrups yet
00AC 1410                       BSF     T1CON, TMR1ON   ; Turn ON timer1
                        ;
                        ;
                        ; This code segment is an infinite loop that will always transmit the data
                        ; contained in the XMIT_DATA register. After each byte is transmitted a new
                        ; byte is read. If using PICMASTER (in stand alone mode), this is read from
                        ; a register that is updated after a break (at NOP). If in a system, PORTB
                        ; is read. All other variables are reinitalized after each byte.
                        ;
0299 NEXT_BYTE
                        ;
0301 WAIT               MOVF    DATA_CNT, w     ;
0302                    BTFSS   STATUS, Z       ; Is DATA_CNT = 0 ?
0303                    GOTO    WAIT            ; NO, must wait until YES
0304                    NOP                     ;
0305                if ( Debug )
0306                    bcf     PORTA, 0        ; Turn off strobe
0307                endif                       ;
0308                                            ;
0309                if ( PICMaster )
0310                    MOVF    DUMMY_PB, W     ;
0311                else                        ;
0312                    MOVF    PORTB, W        ;
0313                endif                       ;
0314                    MOVWF   XMIT_DATA       ; New data to transmit
0315                    MOVLW   0xFF            ;
0316                    MOVWF   CCP1_INT_CNT    ;
0317                    MOVLW   0x09            ;
0318                    MOVWF   DATA_CNT        ;
0319                    CLRF    ONES_CNT        ;
0320                    GOTO    NEXT_BYTE       ;
                        ;
                        ;
                        ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                        ;
                        ;
0325 OTHER_RESET    BTFSS   STATUS, TO      ; WDT Time-out?
0326 WDT_TIMEOUT    GOTO    ERROR1          ; YES, This is error condition
0327                if ( Debug_PU )
0328                    goto    START       ; MCLR reset, Goto START
0329                else
```

```
                        GOTO   MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                      endif
                      ;
                      if (Debug )
00BD 0000                NOP                  ; END lable for debug
                      endif
                      ;
                      ;
                 org     PMEM_END              ; End of Program Memory
07FF 2808             GOTO   ERROR1           ; If you get here your program was lost
                 end


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-
00C0 : ________________ ________________ ________________ ________________

0780 : ________________ ________________ ________________ ________________
07C0 : ________________ ________________ ________________ ________X

All other memory blocks unused.

Errors   :     0
Warnings :    10
```

APPENDIX B2: COMP.H

```
    nolist
;*****************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:        CLOCK.H
;       Revision:       7-13-94
;
;*****************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq        EQU     D'10000000'             ; Device Frequency is 4 MHz
PULSE_TIME      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ZERO_BIT      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ONE_BIT       EQU     ( Dev_Freq / D'4000' ) * D'376' / D'10000' )
;
ZERO_1S         EQU     ((( Dev_Freq / D'4000' ) * (D'6000' - (D'16' * D'188'))) / D'10000' )
ONE_1S          EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (3 * D'188' + D'14' * D'188')) / D'10000')
TWO_1S          EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (6 * D'188' + D'12' * D'188')) / D'10000')
THREE_1S        EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'9' * D'188' + D'10' * D'188')) / D'10000')
FOUR_1S         EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'12' * D'188' + 8 * D'188')) / D'10000')
FIVE_1S         EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'15' * D'188' + 6 * D'188')) / D'10000')
SIX_1S          EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'18' * D'188' + 4 * D'188')) / D'10000')
SEVEN_1S        EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'21' * D'188' + 2 * D'188')) / D'10000')
EIGHT_1S        EQU     (( ( Dev_Freq / D'4000' ) * (D'6000' - (D'24' * D'188') ) / D'10000' )
;
DB_HI_BYTE      EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY  EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR      EQU     40                      ; RAM Location
OUTER_CNTR      EQU     41                      ; RAM Location
;
T1OSO           EQU     0                       ; The RC0 / T1OSO / T1CKI
;
RESET_V         EQU     0x0000                  ; Address of RESET Vector
ISR_V           EQU     0x0004                  ; Address of Interrupt Vector
PMEM_END        EQU     0x07FF                  ; Last address in Program Memory
TABLE_ADDR      EQU     0x0400                  ; Address where to start Tables
;
COUNTER         EQU     0x021                   ;
;
XMIT_DATA       EQU     0x30
DATA_CNT        EQU     0x31
ONES_CNT        EQU     0x32
CCP1_INT_CNT    EQU     0x33
DUMMY_PB        EQU     0x40
CCPREG_HI       EQU     0x41
CCPREG_LO       EQU     0x42
;
        list
```

APPENDIX C: CAPT_2.LST

```
MPASM 01.01 Released    CAPT_2.ASM   7-19-1994  10:54:15                PAGE  1


LOC  OBJECT CODE     LINE  SOURCE TEXT

                     0001          LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program implements a real time clock using the TMR1 module of the
                     0006 ; PIC16Cxx family.
                     0007 ;
                     0008 ;       Program = CAPT_2.ASM
                     0009 ;       Revision Date:   7-19-94
                     0010 ;
                     0011 ;*****************************************************************************
                     0012 ;
                     0013 ;
                     0014 ; HARDWARE SETUP
                     0015 ;
                     0016 ;               CCP2 Compare Output
                     0017 ;               CCP1 Capture Input
                     0018 ;               CCP2  -> CCP1
                     0019 ;
                     0020 ;
                     0021                 INCLUDE <C74_reg.h>
                     0249
                     0021
                     0022                 INCLUDE <CAPT.h>
                     0022
                     0023 ;
                     0024 ;
0001                 0025 PICMaster       EQU     TRUE            ; A Debugging Flag
0001                 0026 Debug           EQU     TRUE            ; A Debugging Flag
0001                 0027 Debug_PU        EQU     TRUE            ; A Debugging Flag
                     0028 ;
                     0029 ;
                     0030 ; Reset address. Determine type of RESET
                     0031 ;
                     0032                 org     RESET_V         ; RESET vector location
0000 1683            0033 RESET           BSF     STATUS, RP0     ; Bank 1
0001 188E            0034                 BTFSC   PCON, POR       ; Power-up reset?
0002 282F            0035                 GOTO    START           ; YES
0003 2861            0036                 GOTO    OTHER_RESET     ; NO, a WDT or MCLR reset
                     0037 ;
                     0038 ; This is the Peripheral Interrupt routine. Need to determine the type
                     0039 ; of interrupt that occurred. The following interrupts are enabled:
```

```
0040 ;  1.  CCP1 Capture Occured
0041 ;  2.  CCP2 Compare Occured
0042 ;
0044         org     ISR_V                   ; Interrupt vector location
0045 PER_INT_V
0046                 BCF     STATUS, RP0     ; Bank 0
0047                 BTFSC   PIR1, CCP1IF    ; CCP1 Interrupt occured? (Capture)
0048                 GOTO    CAPTURE         ; YES, Service the CCP1 Interrupt
0049                 BTFSC   PIR2, CCP2IF    ; CCP2 Interrupt occured? (Compare)
0050                 GOTO    COMPARE         ; YES, Service the CCP2 Interrupt
0051                 BTFSC   PIR1, TMR1IF    ; NO, Timer 1 Overflow?
0052                 GOTO    T1OVFL          ; YES,
0053 ERROR1                                  ; NO, Error Condition - Unknown Interrupt
0054                 BSF     PORTD, 1        ; Toggle a PORT pin
0055                 BCF     PORTD, 1
0056                 GOTO    ERROR1
0057 ;
0058 ERROR2                                  ; NO, Error Condition - Unknown Interrupt
0059                 BSF     PORTD, 2        ; Toggle a PORT pin
0060                 BCF     PORTD, 2
0061                 GOTO    ERROR2
0062 ;
0063 ; The Compare generates a Square wave based on the value on PORTB (in DUMMY_PB)
0064 ; and on PORTD (in DUMMY_PD). PORTB is loaded into low compare latch and PORTD
0065 ; is loaded into the high compare latch. If the value of the ports is not changed,
0066 ; a capture overflow condition will occur when PORTD:PORTB > 7Fh. This overflow
0067 ; is only indicated by the time between captures being much less then expected.
0068 ;
0069 COMPARE
0070                 BCF     PIR2, CCP2IF    ; Clear CCP2 Interrupt Flag
0071             if ( PICMaster )
0072                 MOVF    DUMMY_PB, W     ;
0073             else
0074                 MOVF    PORTB, W        ;
0075             endif
0076                 ADDWF   CCPR2L, F       ; Update Compare register pair latch
0077                 BTFSC   STATUS, C       ;
0078                 INCF    CCPR2H, F       ;
0079             if ( PICMaster )
0080                 MOVF    DUMMY_PD, W     ;
0081             else
0082                 MOVF    PORTD, W        ;
0083             endif
0084                 ADDWF   CCPR2H, F       ;
0085                 INCF    CCP2_INT_CNT    ;
0086                 BSF     CCP2CON, 0      ; On Compare match, CCP2 pin = L
0087                 BTFSS   CCP2_INT_CNT, 0 ;
0088                 BCF     CCP2CON, 0      ; On Compare match, CCP2 pin = H

0004 1283
0005 190C
0006 281D
0007 180D
0008 2811
0009 180C
000A 282D

000B 1488
000C 1088
000D 280B

000E 1508
000F 1108
0010 280E

0011 100D

0012 0851

0013 079B
0014 1803
0015 0A9C

0016 0853

0017 079C
0018 0AB3
0019 141D
001A 1C33
001B 101D
```

```
0089 END_COMPARE
0090                RETFIE                  ; Return / Enable Global Interrupts
0091
0092 ;
0093 ; The result of the new capture minus the old capture is stored in the new capture
0094 ; registers (CAPT_NEW_H:CAPT_NEW_L)
0095 ;
0096 CAPTURE
0097                BCF     PIR1, CCP1IF    ; Clear CCP1 Interrupt Flag
0098                MOVF    CCPR1L, W       ; New capture value (low byte)
0099                MOVWF   CAPT_NEW_L      ;
0100                MOVF    CCPR1H, W       ; New capture value (high byte)
0101                MOVWF   CAPT_NEW_H      ;
0102 ;
0103                MOVF    CAPT_OLD_L, W   ;
0104                SUBWF   CAPT_NEW_L, F   ; Subtract the low bytes of the 2 captures
0105                BTFSS   STATUS, C       ; Did a borrow occur?
0106                DECF    CAPT_NEW_H, F   ; YES, Decrement old capture (high byte)
0107                MOVF    CAPT_OLD_H, W   ; New capture value (low byte)
0108                SUBWF   CAPT_NEW_H, F   ; Subtract the low bytes of the 2 captures
0109 LOAD_OLD      MOVF    CCPR1L, W       ; New capture value (low byte)
0110                MOVWF   CAPT_OLD_L      ;
0111                MOVF    CCPR1H, W       ; New capture value (high byte)
0112                MOVWF   CAPT_OLD_H      ;
0113 END_CAPTURE
0114                RETFIE
0115 ;
0116 ;
0117 T1OVFL
0118                BCF     PIR1, TMR1IF    ; Clear T1 Overflow Interrupt Flag
0119                RETFIE                  ; Return / Enable Global Interrupts
0120 ;
0121 ;*****************************************************************************
0122 ;*****     Start program here, Power-On Reset occurred.
0123 ;*****************************************************************************
0124 ;*****************************************************************************
0125 ;
0126 START           BCF     STATUS, RP0     ; POWER_ON Reset (Beginning of program)
0127                CLRF    TMR1H           ; Bank 0
0128                CLRF    TMR1L           ;
0129                                        ;
0130 ;
0131 MCLR_RESET     BCF     STATUS, RP0     ; A Master Clear Reset
0132                CLRF    STATUS          ; Bank 0
0133                CLRF    INTCON          ; Do initialization (Bank 0)
0134                CLRF    PIR1
0135                BSF     STATUS, RP0     ; Bank 1
0136
```

```
001C 0009
0020 0816
0021 00C0
001D 110C
001E 0815
001F 00C1
0022 0843
0023 02C1
0024 1C03
0025 03C0
0026 0842
0027 02C0
0028 0815
0029 00C3
002A 0816
002B 00C2
002C 0009
002D 100C
002E 0009
002F 1283
0030 018F
0031 018E
0032 1283
0033 0183
0034 018B
0035 018C
0036 1683
```

```
0037 3000                MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
0038 0081                MOVWF   OPTION_R        ;
0039 018C                CLRF    PIE1            ; Disable all peripheral interrupts
003A 018D                CLRF    PIE2            ; Disable all peripheral interrupts
003B 30FF                MOVLW   0xFF            ;
003C 009F                MOVWF   ADCON1          ; Port A is Digital.
                         ;
                         ;
003D 1283                BCF     STATUS, RP0     ; Bank 0
003E 0185                CLRF    PORTA           ; ALL PORT output should output Low.
003F 0186                CLRF    PORTB
0040 0187                CLRF    PORTC
0041 0188                CLRF    PORTD
0042 0189                CLRF    PORTE
0043 1010                BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                         ;
0044 1683                BSF     STATUS, RP0     ; Select Bank 1
0045 0185                CLRF    TRISA           ; RA5 -  0 outputs
0046 30FF                MOVLW   0xFF            ;
0047 0086                MOVWF   TRISB           ; RB7 - 0 inputs
0048 0187                CLRF    TRISC           ; RC Port are outputs
0049 1507                BSF     TRISC, 2        ; CCP1 is an INPUT
004A 0088                MOVWF   TRISD           ; RD Port are inputs
004B 0189                CLRF    TRISE           ; RE Port are outputs
004C 150C                BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
004D 140D                BSF     PIE2, CCP2IE    ; Enable CCP2 Interrupt
004E 1283                BCF     STATUS, RP0     ; Select Bank 0
                         ;
                         ; Initialize the Special Function Registers (SFR) interrupts
                         ;
004F 018C                CLRF    PIR1            ;
0050 018D                CLRF    PIR2            ;
0051 0190                CLRF    T1CON           ; Timer mode
0052 170B                BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0053 178B                BSF     INTCON, GIE     ; Enable all Interrupts
                         ;
                         ; Set-up timer and compare latches and then turn timer1 on.
                         ;
0054 1010                    BCF     T1CON, TMR1ON   ; Turn OFF timer1
                         if ( PICMaster )
0055 0851                    MOVF    DUMMY_PB, W     ;
                         else
                             MOVF    PORTB, W        ;
                         endif
0056 079B                    ADDWF   CCPR2L, F       ; Update Compare register pair latch
0057 1803                    BTFSC   STATUS, C       ;
0058 0A9C                    INCF    CCPR2H, F       ;
```

```
0185         if ( PICMaster )                ;
0059 08D3       MOVF    DUMMY_PD
0186         else                            ;
0187            MOVF    PORTD, W
0188         endif                           ;
0189                                         ;
005A 079C       ADDWF   CCPR2H, F            ;
005B 3008       MOVLW   0x08                 ; On match CCP2 = H level
005C 009D       MOVWF   CCP2CON              ;
005D 3005       MOVLW   0x05                 ; Capture on every rising edge
005E 0097       MOVWF   CCP1CON              ;
005F 1410       BSF     T1CON, TMR1ON        ; Turn ON timer1
0195 ;
0196 ;
0197 ;
0198 ;
0060 2860       goto    lzz                  ; Loop waiting for interrupts (for use with PICMASTER)
0199 lzz
0200 ;
0201 ;
0202 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
0203 ;
0061 1E03 OTHER_RESET   BTFSS  STATUS, TO     ; WDT Time-out?
0062 280B WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
0206            if ( Debug_PU )
0063 282F          goto   START              ; MCLR reset, Goto START
0208            else
0209               GOTO   MCLR_RESET         ; MCLR reset, Goto MCLR_RESET
0210            endif
0211 ;
0212            if (Debug )
0064 0000 END_START     NOP                  ; END lable for debug
0214            endif
0215 ;
0216 ;
0217         org     PMEM_END                ; End of Program Memory
07FF 280B       GOTO    ERROR1               ; If you get here your program was lost
0219
0220         end
0221
0222
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX--------------- ----------------

0780 : ---------------- ---------------- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ----------------X

All other memory blocks unused.

Errors   :   0
Warnings :  13
```

APPENDIX C2: CAPT.H

```
       nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:       CLOCK.H
;       Revision:      7-19-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq              EQU           D'4000000'            ; Device Frequency is 4 MHz
DB_HI_BYTE            EQU           (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY        EQU           (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR            EQU           40                    ; RAM Location
OUTER_CNTR            EQU           41                    ; RAM Location
;
T1OSO                 EQU           0                     ; The RC0 / T1OSO / T1CKI
;
RESET_V               EQU           0x0000                ; Address of RESET Vector
ISR_V                 EQU           0x0004                ; Address of Interrupt Vector
PMEM_END              EQU           0x07FF                ; Last address in Program Memory
TABLE_ADDR            EQU           0x0400                ; Address where to start Tables
;
COUNTER               EQU           0x021                 ;
CCP2_INT_CNT          EQU           0x33
;
;
; DUMMY_PD:DUMMY_PB contain the value to be loaded into the CCP2 compare registers
; (CCPR2H:CCPR2L)
;
DUMMY_PA              EQU           0x50
DUMMY_PB              EQU           0x51
DUMMY_PC              EQU           0x52
DUMMY_PD              EQU           0x53
DUMMY_PE              EQU           0x54
;
;
; CAPT_NEW_H:CAPT_NEW_L stores the NEW captured value and the result of the
; subtraction between this capture and the previous.
;    CAPT_NEW_H:CAPT_NEW_L  =  CAPT_NEW_H:CAPT_NEW_L - CAPT_OLD_H:CAPT_OLD_L
;
; After all computations the new capture value is moved to the CAPT_OLD_H:CAPT_OLD_L
; in preperation for the next capture value.
;
CAPT_NEW_H            EQU           0x040                 ;
CAPT_NEW_L            EQU           0x041                 ;
CAPT_OLD_H            EQU           0x042                 ;
CAPT_OLD_L            EQU           0x043                 ;
;
       list
```

AN587

Interfacing to an LCD Module

INTRODUCTION

This application note interfaces a PIC16CXX device to the Hitachi LM032L LCD character display module. This module is a two line by twenty character display. LCD modules are useful for displaying text information from a system. In large volume applications, the use of custom LCD displays become economical. These routines should be a good starting point for users whose application implement a custom LCD. This source code should be compatible with the PIC16C5X devices, after modifications for the special function register initialization, but has not been verified on those devices.

OPERATION

The Hitachi® LM032L LCD character display module can operate in one of two modes. The first (and default) mode is the 4-bit data interface mode. The second is the 8-bit data interface mode. When operating in 4-bit mode, two transfers per character / command are required. 8-bit mode, though easier to implement (less program memory) requires four additional I/O lines. The use of 8-bit mode is strictly a program memory size / I/O tradeoff. The three most common data interfaces from the microcontroller are:

1. An 8-bit interface.
2. A 4-bit interface, with data transfers on the high nibble of the port.
3. A 4-bit interface, with data transfers on the low nibble of the port.

The LCD module also has three control Signal, Enable (E), Read / Write (R_W), and Register Select (RS). These functions of these control signals are show in Table 1.

TABLE 1: CONTROL SIGNAL FUNCTIONS

Control Signal	Function
E	Causes data / control state to be latched Rising Edge = Latches control state (RS and R_W) Falling Edge = Latches data
RS	Register Select Control 0 = LCD in command mode 1 = LCD in data mode
R_W	Read / Write control 0 = LCD to read data 1 = LCD to write data

A single source file, with conditional assemble is used to generate these three options. This requires two flags. The flags and their results are shown in Table 2.

TABLE 2: CONDITIONAL ASSEMBLY FLAGS

Flags		
Four_bit	Data_HI	Result
1	0	4-bit mode. Data transferred on the low nibble of the port.
1	1	4-bit mode. Data transferred on the high nibble of the port.
0	X	8-bit mode.

Interfacing to an LCD Module

Figure 1A through Figure 1C show the block diagrams for the three different data interfaces. The LCD_CNTL and LCD_DATA lines are user definable to their port assignment. This is accomplished with EQUate statements in the source code. See Appendices B - D.

FIGURE 1A: 8-BIT DATA INTERFACE

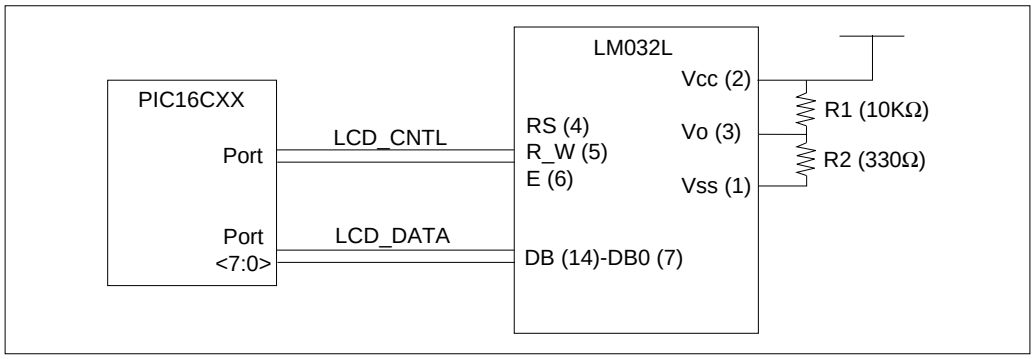


FIGURE 1B: 4-BIT MODE. DATA TRANSFERRED ON THE HIGH NIBBLE OF THE PORT

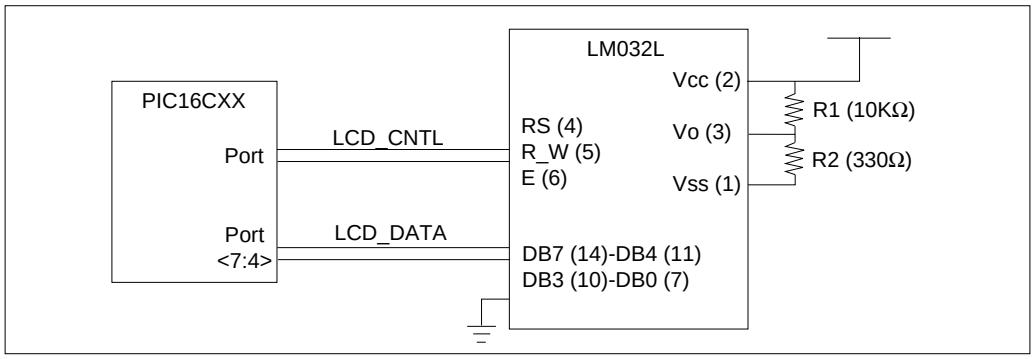
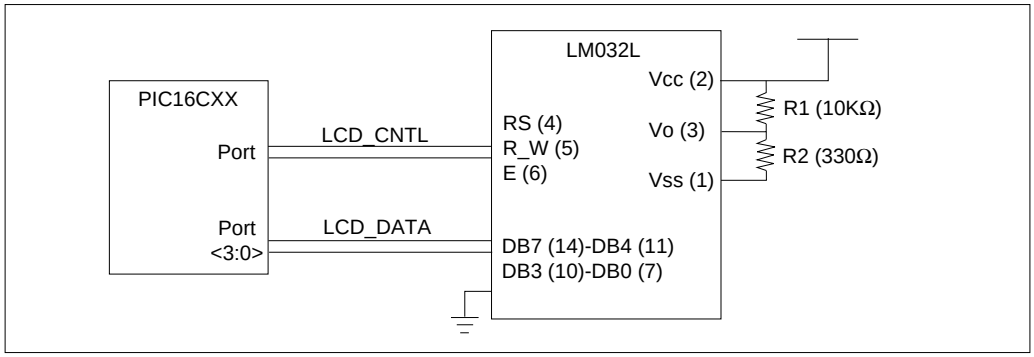


FIGURE 1C: 4-BIT MODE. DATA TRANSFERRED ON THE LOW NIBBLE OF THE PORT



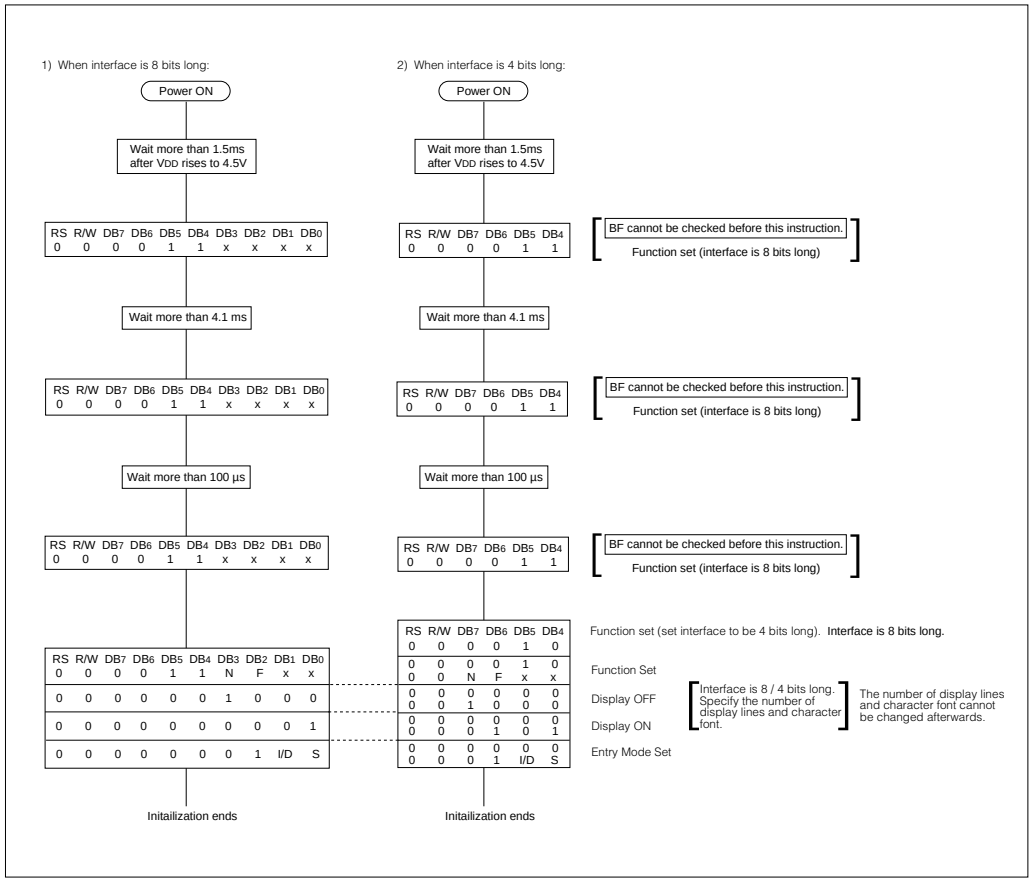
LCD 's (drivers) are slow devices when compared to microcontrollers. Care must be taken from having communication occur to quickly. The timing requirements of the LM032L are shown in Appendix A. It is recommended that the complete specifications of the LM032L be acquired from Hitachi or an Hitachi distributor. The literature number is CE-E613Q and M24T013 for the display driver.

When the module powers up, the default data transfer mode is 8-bit. The initialization sequence only requires commands that are 4-bit in length. The last initialization command then needs to be sent to the display to specify the data transfer width (4- or 8-bit). Then a delay of 4.6 ms must be executed before the LCD module can be initialized. Some of the LCD module commands are:

- 1 or 2 lines of characters
- Display on /off
- Clear display
- Increment / do not increment character address pointer after each character
- Load character address pointer

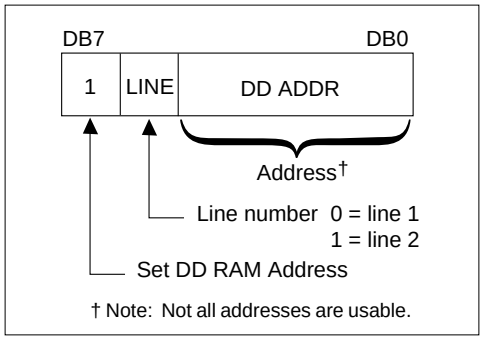
The initialization flow for the module is shown in Figure 2.

FIGURE 2: INITIALIZATION FLOW FOR LCD MODULE



After initialization, each character address is individually addressable. Figure 3 shows the structure of the command to specify the character address.

FIGURE 3: CHARACTER ADDRESS COMMAND FORMAT



The Hatachi Display Drive (HD44780A) has 80 bytes of RAM. The LM032L modules only use 40 bytes of the available RAM (2 x 20 characters). It is possible to use the remaining RAM locations for storage of other information.

Figure 4 shows the display data positions supported by the display driver as well as the characters actually displayed by the module (the shaded addresses).

The program example implemented here uses the auto character increment feature. This automatically increments the character address pointer after each character is written to the display.

CONCLUSION

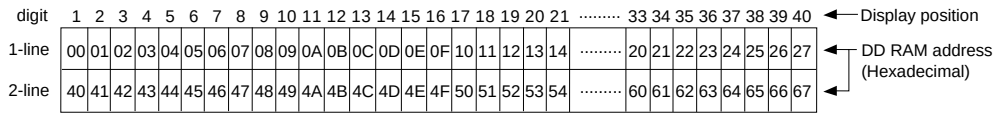
The Hitachi LM032L character display module is useful for the display of information. The selection of 4-bit or 8-bit data transfer mode is strictly a program memory size / I/O resource trade-off. The supplied code is easily used in one of three common data interfaces. The source is easily modifiable to the designers specific application needs. Other display modules / drivers maybe implemented with the appropriate modifications. Table 4 shows the resource requirements for the three subroutines SEND_CHAR, SEND_COMMAND, and BUSY_CHECK in the various data interface modes.

TABLE 4: RESOURCE REQUIREMENTS

Mode	Program Memory	Data Memory	Verified On
8-bit	32	3	PICDEM-2 *
4-bit, Data transferred on the high nibble of the port	53	3	PICDEM-2 *
4-bit, Data transferred on the high nibble of the port	53	3	Low Power Real Time Clock Board (AN582)

* Jumper J6 must be removed.

FIGURE 4: DISPLAY DRIVER (DD) RAM LOCATIONS



Note: Shaded locations are displayed on the LM032L display module.

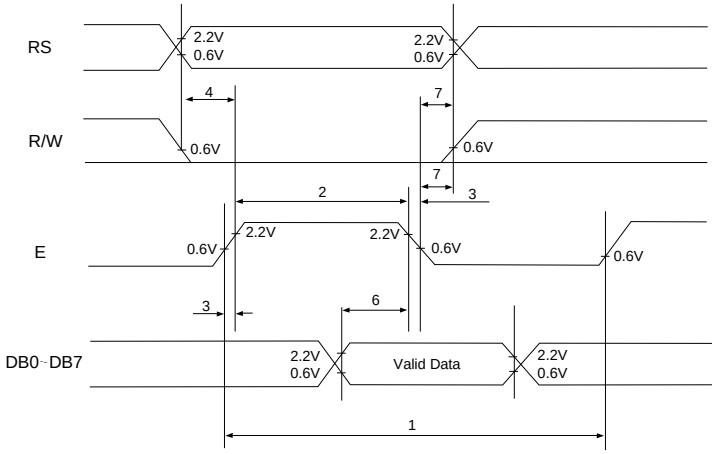
Written By: Mark Palmer - Sr. Application Engineer
Code By: Mark Palmer / Scott Fink - Sr. Application Engineers

APPENDIX A: LM032L TIMING REQUIREMENTS

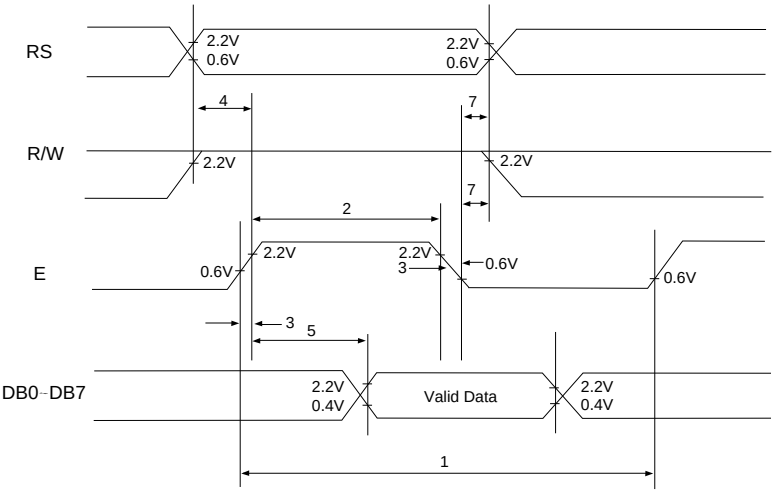
TIMING CHARACTERISTICS

Parm #	Symbol	Characterisitics	Min.	Typ.	Max.	Unit
1	t_{cyc}	Enable cycle time	1.0	—	—	µs
2	PW_{EH}	Enable pulse width	450	—	—	ns
3	t_{Er}, t_{Ef}	Enable rise / fall time	—	—	25	ns
4	t_{AS}	RS, R/W set up time	140	—	—	ns
5	t_{DDR}	Data delay time	—	—	320	ns
6	t_{DSW}	Data set up time	195	—	—	ns
7	t_H	Hold time	20	—	—	ns

DATA WRITE INTERFACE TIMING



DATA READ INTERFACE TIMING



Note: Refer to Hitachi documentation for most current timing specifications.

LM032L PIN CONNECTION

Pin No.	Symbol	Level	Function	
1	Vss	—	0V	
2	V_{DD}	—	+5V	Power Supply
3	V_O	—	—	
4	RS	H/L	L: Instruction Code Input H: Data Input	
5	R/W	H/L	H: Data Read (LCD module→MPU) L: Data Write (LCD module←MPU)	
6	E	H,H→L	Enable Signal	
7	DB0	H/L	Data Bus Line Note (1), (2)	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L		
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

Notes:

In the HD44780, the data can be sent in either a 4-bit 2-operation or a 8-bit 1-operation, so that it can interface to both 4- and 8-bit MPUs.

(1) When interface data is 4-bits long, data is transferred using only 4 buses of DB~DB_7 and DB_0~DB_3 are not used. Data transfer between the HD44780 and the MPU completes when 4-bit data is tranferred twice. Data of the higher order 4 bits (contents of DB_4~DB_7 when interface data is 8-bits long) is tranferred first and then lower order 4 bits (contents of DB_4~DB_7 when interface data is 8-bits long).

(2) When interface data is 8-bits long, data is transferred using 8 data buses of DB0~DB_7.

APPENDIX B: 8-BIT DATA INTERFACE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:53:47                         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;;
                     0012 ;;
                     0013         include <C74_reg.h>
                     0233
                     0013
                     0014         include <lm032l.h>
                     0014
                     0015 ;
0000                 0016 Four_bit        EQU     FALSE       ; Selects 4- or 8-bit data transfers
0001                 0017 Data_HI         EQU     TRUE        ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;;
                     0019 ;;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;;
                     0022 LCD_DATA        EQU     PORTB
                     0023 LCD_DATA_TRIS   EQU     TRISB
                     0024 ;;
                     0025     else
                     0026 ;;
0008                 0027 LCD_DATA        EQU     PORTD
0088                 0028 LCD_DATA_TRIS   EQU     TRISD
                     0029 ;;
                     0030     endif
                     0031 ;;
```

```
0005                    LCD_CNTL    EQU     PORTA
                        ;
                        ;
                        ;
                        ; LCD Display Commands and Control Signal names.
                        ;
                        ;   if ( Four_bit && !Data_HI )
                        ;
                        ;   else
                        ;
0003                    ;E          EQU     3               ; LCD Enable control line
0002                    ;R_W        EQU     2               ; LCD Read/Write control line
0001                    ;RS         EQU     1               ; LCD Register Select control line
                        ;
                        ;   endif
                        ;
                        ;
0030                    TEMP1       EQU     0x030
                        ;
                                    org     RESET_V         ; RESET vector location
0000 2808               RESET       GOTO    START           ;
                        ;
                        ; This is the Periperal Interrupt routine. Should NOT get here
                        ;
                                    org     ISR_V           ; Interrupt vector location
0004                    PER_INT_V
0004 1283               ERROR1      BCF     STATUS, RP0     ; Bank 0
0005 1407                           BSF     PORTC, 0
0006 1007                           BCF     PORTC, 0
0007 2804                           GOTO    ERROR1
                        ;
                        ;
                        ;
0008 0183               START       CLRF    STATUS          ; POWER_ON Reset (Beginning of program)
0009 018B                           CLRF    INTCON          ; Do initialization (Bank 0)
000A 018C                           CLRF    PIR1
000B 1683                           BSF     STATUS, RP0     ; Bank 1
000C 3000                           MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
000D 0081                           MOVWF   OPTION_R        ;
```

```
                        0077          CLRF    PIE1            ; Disable all peripheral interrupts
000E 018C               0078          MOVLW   0xFF            ;
000F 30FF               0079          MOVWF   ADCON1          ; Port A is Digital.
0010 009F               0080 ;
                        0081 ;
0011 1283               0082          BCF     STATUS, RP0     ; Bank 0
0012 0185               0083          CLRF    PORTA           ; ALL PORT output should output Low.
0013 0186               0084          CLRF    PORTB
0014 0187               0085          CLRF    PORTC
0015 0188               0086          CLRF    PORTD
0016 0189               0087          CLRF    PORTE
0017 1010               0088          BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                        0089 ;
0018 1683               0090          BSF     STATUS, RP0     ; Select Bank 1
0019 0185               0091          CLRF    TRISA           ; RA5 -  0 outputs
001A 30F0               0092          MOVLW   0xF0            ;
001B 0086               0093          MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187               0094          CLRF    TRISC           ; RC Port are outputs
001D 1407               0095          BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188               0096          CLRF    TRISD           ; RD Port are outputs
001F 0189               0097          CLRF    TRISE           ; RE Port are outputs
0020 140C               0098          BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781               0099          BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283               0100          BCF     STATUS, RP0     ; Select Bank 0
                        0101 ;
                        0103 ;
                        0104 ; Initilize the LCD Display Module
                        0105 ;
0023 0185               0106          CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                        0107
                        0108 DISPLAY_INIT
                        0109      if ( Four_bit && !Data_HI )
                        0110          MOVLW   0x02            ; Command for 4-bit interface low nibble
                        0111      endif
                        0112 ;
                        0113      if ( Four_bit && Data_HI )
                        0114          MOVLW   0x020           ; Command for 4-bit interface high nibble
                        0115      endif
                        0116 ;
                        0117      if ( !Four_bit )
0024 3038               0118          MOVLW   0x038           ; Command for 8-bit interface
                        0119      endif
                        0120 ;
0025 0088               0121          MOVWF   LCD_DATA        ;
0026 1585               0122          BSF     LCD_CNTL, E     ;
```

```
0027 1185                       BCF     LCD_CNTL, E     ;
                                ;
                                ; This routine takes the calculated times that the delay loop needs to
                                ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                                ; frequency of operation. These uses registers before they are needed to
                                ; store the time.
                                ;
0028 3006 LCD_DELAY     MOVLW   LCD_INIT_DELAY  ;
0029 00B3                       MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
002A 01B4                       CLRF    LSD             ;
002B 0BB4 LOOP2                 DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002C 282B                       GOTO    LOOP2           ;
002D 0BB3                       DECFSZ  MSD             ;
002E 282B END_LCD_DELAY         GOTO    LOOP2           ;
                                ;
                                ; Command sequence for 2 lines of 5x7 characters
                                ;
                                CMD_SEQ
                                ;
                                    if ( Four_bit )
                                        if ( !Data_HI )
0030 0088                               MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
0031 1585                               BSF     LCD_CNTL, E     ;
0032 1185                               BCF     LCD_CNTL, E     ;
                                        ;
                                    if ( Four_bit )             ; This code for only 4-bit mode (2nd xfer)
                                        if ( !Data_HI )
                                            MOVLW   0x08        ; 4-bit low nibble xfer
                                        else
                                            MOVLW   0x080       ; 4-bit high nibble xfer
                                        endif
                                            MOVWF   LCD_DATA    ;
                                            BSF     LCD_CNTL, E ;
                                            BCF     LCD_CNTL, E ;
```

```
0167     endif
0168 ;
0169 ; Busy Flag should be valid after this point
0170 ;
0171                 MOVLW   DISP_ON         ;
0172                 CALL    SEND_CMD        ;
0173                 MOVLW   CLR_DISP        ;
0174                 CALL    SEND_CMD        ;
0175                 MOVLW   ENTRY_INC       ;
0176                 CALL    SEND_CMD        ;
0177                 MOVLW   DD_RAM_ADDR     ;
0178                 CALL    SEND_CMD        ;
0179 ;
0180 ;
0181 ;
0182 ;Send a message the hard way
0183                 movlw   'M'
0184                 call    SEND_CHAR
0185                 movlw   'i'
0186                 call    SEND_CHAR
0187                 movlw   'c'
0188                 call    SEND_CHAR
0189                 movlw   'r'
0190                 call    SEND_CHAR
0191                 movlw   'o'
0192                 call    SEND_CHAR
0193                 movlw   'c'
0194                 call    SEND_CHAR
0195                 movlw   'h'
0196                 call    SEND_CHAR
0197                 movlw   'i'
0198                 call    SEND_CHAR
0199                 movlw   'p'
0200                 call    SEND_CHAR
0201
0202                 movlw   B'11000000'     ;Address DDRam first character, second line
0203                 call    SEND_CMD
0204
0205 ;Demonstration of the use of a table to output a message
0206                 movlw   0               ;Table address of start of message
0207 dispmsg
0208                 movwf   TEMP1           ;TEMP1 holds start of message address
0209                 call    Table
0210                 andlw   0FFh            ;Check if at end of message (zero
0211                 btfsc   STATUS,Z        ;returned at end)
0212                 goto    out
0213                 call    SEND_CHAR       ;Display character
0214                 movf    TEMP1,w         ;Point to next character

0033 300C
0034 206A
0035 3001
0036 206A
0037 3006
0038 206A
0039 3080
003A 206A

003B 304D
003C 2061
003D 3069
003E 2061
003F 3063
0040 2061
0041 3072
0042 2061
0043 306F
0044 2061
0045 3063
0046 2061
0047 3068
0048 2061
0049 3069
004A 2061
004B 3070
004C 2061

004D 30C0
004E 206A

004F 3000

0050 00B0
0051 2085
0052 39FF
0053 1903
0054 2859
0055 2061
0056 0830
```

```
                        addlw   1
                        goto    dispmsg
                out
                loop
                        goto    loop            ;Stay here forever
                ;
                ;
                INIT_DISPLAY
                        MOVLW   DISP_ON         ; Display On, Cursor On
                        CALL    SEND_CMD        ; Send This command to the Display Module
                        MOVLW   CLR_DISP        ; Clear the Display
                        CALL    SEND_CMD        ; Send This command to the Display Module
                        MOVLW   ENTRY_INC       ; Set Entry Mode Inc., No shift
                        CALL    SEND_CMD        ; Send This command to the Display Module
                        RETURN
                ;
                ;
                ;*****************************************************************
                ;* The LCD Module Subroutines                                    *
                ;*****************************************************************
                ;
                ;
                   if ( Four_bit )     ; 4-bit Data transfers?
                ;
                      if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
                ;
                ;*****************************************************************
                ;*SendChar - Sends character to LCD                              *
                ;*This routine splits the character into the upper and lower     *
                ;*nibbles and sends them to the LCD, upper nibble first.         *
                ;*****************************************************************
                ;
                SEND_CHAR
                        MOVWF   CHAR            ;Character to be sent is in W
                        CALL    BUSY_CHECK      ;Wait for LCD to be ready
                        MOVF    CHAR, w
                        ANDLW   0xF0            ;Get upper nibble
                        MOVWF   LCD_DATA        ;Send data to LCD
                        BCF     LCD_CNTL, R_W   ;Set LCD to read
                        BSF     LCD_CNTL, RS    ;Set LCD to data mode
                        BSF     LCD_CNTL, E     ;toggle E for LCD
                        BCF     LCD_CNTL, E
                        SWAPF   CHAR, w
                        ANDLW   0xF0            ;Get lower nibble
                        MOVWF   LCD_DATA        ;Send data to LCD
                        BSF     LCD_CNTL, E     ;toggle E for LCD
                        BCF     LCD_CNTL, E
```

```
0262                    RETURN
0263 ;
0264                ; 4-bit transfers on the low nibble of the PORT
0265 ;
0266 ;*************************************************************************
0267 ;* SEND_CHAR - Sends character to LCD                                     *
0268 ;* This routine splits the character into the upper and lower            *
0269 ;* nibbles and sends them to the LCD, upper nibble first.                *
0270 ;* The data is transmitted on the PORT<3:0> pins                         *
0271 ;*************************************************************************
0272 ;
0273 SEND_CHAR
0274                    MOVWF   CHAR            ; Character to be sent is in W
0275                    CALL    BUSY_CHECK      ; Wait for LCD to be ready
0276                    SWAPF   CHAR, W
0277                    ANDLW   0x0F            ; Get upper nibble
0278                    MOVWF   LCD_DATA        ; Send data to LCD
0279                    BCF     LCD_CNTL, R_W   ; Set LCD to read
0280                    BSF     LCD_CNTL, RS    ; Set LCD to data mode
0281                    BSF     LCD_CNTL, E     ; toggle E for LCD
0282                    BCF     LCD_CNTL, E
0283                    MOVF    CHAR, W
0284                    ANDLW   0x0F            ; Get lower nibble
0285                    MOVWF   LCD_DATA        ; Send data to LCD
0286                    BSF     LCD_CNTL, E     ; toggle E for LCD
0287                    BCF     LCD_CNTL, E
0288                    RETURN
0289 ;
0290                endif
0291            else
0292 ;
0293 ;*************************************************************************
0294 ;* SEND_CHAR - Sends character contained in register W to LCD            *
0295 ;* This routine sends the entire character to the PORT                   *
0296 ;* The data is transmitted on the PORT<7:0> pins                         *
0297 ;*************************************************************************
0298 ;
0299 SEND_CHAR
0300                    MOVWF   CHAR            ; Character to be sent is in W
0301                    CALL    BUSY_CHECK      ; Wait for LCD to be ready
0302                    MOVF    CHAR, w
0303                    MOVWF   LCD_DATA        ; Send data to LCD
0304                    BCF     LCD_CNTL, R_W   ; Set LCD in read mode
0305                    BSF     LCD_CNTL, RS    ; Set LCD in data mode
0306                    BSF     LCD_CNTL, E     ; toggle E for LCD
0307                    BCF     LCD_CNTL, E
0308                    RETURN

0061 00B6
0062 2073
0063 0836
0064 0088
0065 1105
0066 1485
0067 1585
0068 1185
0069 0008
```

```
;
    endif
;
;
;*****************************************************************************
;* SendCmd - Sends command to LCD                                            *
;* This routine splits the command into the upper and lower                  *
;* nibbles and sends them to the LCD, upper nibble first.                    *
;* The data is transmitted on the PORT<3:0> pins                             *
;*****************************************************************************
;
    if ( Four_bit )     ; 4-bit Data transfers?
;
        if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
;*****************************************************************************
;* SEND_CMD - Sends command to LCD                                           *
;* This routine splits the command into the upper and lower                  *
;* nibbles and sends them to the LCD, upper nibble first.                    *
;*****************************************************************************
;
SEND_CMD
            MOVWF   CHAR            ; Character to be sent is in W
            CALL    BUSY_CHECK      ; Wait for LCD to be ready
            MOVF    CHAR,w
            ANDLW   0xF0            ; Get upper nibble
            MOVWF   LCD_DATA        ; Send data to LCD
            BCF     LCD_CNTL,R_W    ; Set LCD to read
            BCF     LCD_CNTL,RS     ; Set LCD to command mode
            BSF     LCD_CNTL,E      ; toggle E for LCD
            BCF     LCD_CNTL,E
            SWAPF   CHAR,w
            ANDLW   0xF0            ; Get lower nibble
            MOVWF   LCD_DATA        ; Send data to LCD
            BSF     LCD_CNTL,E      ; toggle E for LCD
            BCF     LCD_CNTL,E
            RETURN
;
        else            ; 4-bit transfers on the low nibble of the PORT
;
SEND_CMD
            MOVWF   CHAR            ; Character to be sent is in W
            CALL    BUSY_CHECK      ; Wait for LCD to be ready
            SWAPF   CHAR, W
            ANDLW   0x0F            ; Get upper nibble
            MOVWF   LCD_DATA        ; Send data to LCD
            BCF     LCD_CNTL, R_W   ; Set LCD to read
```

```
                        BCF     LCD_CNTL, RS    ; Set LCD to command mode
                        BSF     LCD_CNTL, E     ; toggle E for LCD
                        BCF     LCD_CNTL, E
                        MOVF    CHAR, W
                        ANDLW   0x0F            ; Get lower nibble
                        MOVWF   LCD_DATA        ; Send data to LCD
                        BSF     LCD_CNTL, E     ; toggle E for LCD
                        BCF     LCD_CNTL, E
                        RETURN
                ;
                endif
            else
            ;
            ;*****************************************************************
            ;* SEND_CND - Sends command contained in register W to LCD     *
            ;* This routine sends the entire character to the PORT         *
            ;* The data is transmitted on the PORT<7:0> pins               *
            ;*****************************************************************
            ;
SEND_CMD
                        MOVWF   CHAR            ; Command to be sent is in W
                        CALL    BUSY_CHECK      ; Wait for LCD to be ready
                        MOVF    CHAR, w
                        MOVWF   LCD_DATA        ; Send data to LCD
                        BCF     LCD_CNTL, R_W   ; Set LCD in read mode
                        BCF     LCD_CNTL, RS    ; Set LCD in command mode
                        BSF     LCD_CNTL, E     ; toggle E for LCD
                        BCF     LCD_CNTL, E
                        RETURN
            ;
            endif
            ;
            ;
            if ( Four_bit )     ; 4-bit Data transfers?
            ;
                if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
            ;
            ;*****************************************************************
            ;* This routine checks the busy flag, returns when not busy    *
            ;*  Affects:                                                   *
            ;*    TEMP - Returned with busy/address                        *
            ;*****************************************************************
            ;
BUSY_CHECK
                        BSF     STATUS, RP0     ; Select Register page 1
                        MOVLW   0xFF            ; Set Port_D for input
                        MOVWF   LCD_DATA_TRIS
```

```
0405            BCF     STATUS, RP0     ; Select Register page 0
0406            BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0407            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0408            BSF     LCD_CNTL, E     ; Set E high
0409            BCF     LCD_CNTL, E     ; Set E low
0410            MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0411            ANDLW   0xF0            ; Mask out lower nibble
0412            MOVWF   TEMP
0413            BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0414            BCF     LCD_CNTL, E
0415            SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
0416            ANDLW   0x0F            ; Mask out upper nibble
0417            IORWF   TEMP            ; Combine nibbles
0418            BTFSC   TEMP, 7         ; Check busy flag, high = busy
0419            GOTO    BUSY_CHECK      ; If busy, check again
0420            BCF     LCD_CNTL, R_W
0421            BSF     STATUS, RP0     ; Select Register page 1
0422            MOVLW   0x0F
0423            MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
0424            BCF     STATUS, RP0     ; Select Register page 0
0425            RETURN
0426    ;
0427        else            ; 4-bit transfers on the low nibble of the PORT
0428    ;
0429    ;*************************************************************
0430    ;* This routine checks the busy flag, returns when not busy  *
0431    ;*  Affects:                                                 *
0432    ;*    TEMP - Returned with busy/address                      *
0433    ;*************************************************************
0434    ;
0435    BUSY_CHECK
0436            BSF     STATUS, RP0     ; Bank 1
0437            MOVLW   0xFF            ; Set PortB for input
0438            MOVWF   LCD_DATA_TRIS
0439            BCF     STATUS, RP0     ; Bank 0
0440            BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0441            BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0442            BSF     LCD_CNTL, E     ; Set E high
0443            BCF     LCD_CNTL, E     ; Set E low
0444            SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0445            ANDLW   0xF0            ; Mask out lower nibble
0446            MOVWF   TEMP
0447            BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0448            BCF     LCD_CNTL, E
0449            MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
0450            ANDLW   0x0F            ; Mask out upper nibble
0451            IORWF   TEMP, F         ; Combine nibbles
```

```
0452                    BTFSC   TEMP, 7         ; Check busy flag, high = busy
0453                    GOTO    BUSY_CHECK      ; If busy, check again
0454                    BCF     LCD_CNTL, R_W
0455                    BSF     STATUS, RP0     ; Bank 1
0456                    MOVLW   0xF0            ;
0457                    MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
0458                    BCF     STATUS, RP0     ; Bank 0
0459                    RETURN
0460 ;
0461                endif
0462                else
0463 ;
0464 ;*****************************************************************
0465 ;* This routine checks the busy flag, returns when not busy   *
0466 ;*  Affects:                                                     *
0467 ;*    TEMP - Returned with busy/address                        *
0468 ;*****************************************************************
0469 ;
0470 BUSY_CHECK
0471                    BSF     STATUS,RP0      ; Select Register page 1
0472                    MOVLW   0xFF            ; Set port_D for input
0473                    MOVWF   LCD_DATA_TRIS
0474                    BCF     STATUS, RP0     ; Select Register page 0
0475                    BCF     LCD_CNTL, RS    ; Set LCD for command mode
0476                    BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0477                    BSF     LCD_CNTL, E     ; Set E high
0478                    BCF     LCD_CNTL, E     ; Set E low
0479                    MOVF    LCD_DATA, w     ; Read busy flag, DDram address
0480                    MOVWF   TEMP
0481                    BTFSC   TEMP, 7         ; Check busy flag, high=busy
0482                    GOTO    BUSY_CHECK
0483                    BCF     LCD_CNTL, R_W
0484                    BSF     STATUS, RP0     ; Select Register page 1
0485                    MOVLW   0x00
0486                    MOVWF   LCD_DATA_TRIS   ; Set port_D for output
0487                    BCF     STATUS, RP0     ; Select Register page 0
0488                    RETURN
0489 ;
0490                endif
0491 ;
0492 ;Table
0493 ;
0494                    addwf   PCL             ;Jump to char pointed to in W reg
0495                    retlw   'M'
0496                    retlw   'i'
0497                    retlw   'c'
0498                    retlw   'r'
0499                    retlw   'o'
0500                    retlw   'c'

0073 1683
0074 30FF
0075 0088
0076 1283
0077 1085
0078 1505
0079 1585
007A 1185
007B 0808
007C 00B5
007D 1BB5
007E 2873
007F 1105
0080 1683
0081 3000
0082 0088
0083 1283
0084 0008

0085 0782
0086 344D
0087 3469
0088 3463
0089 3472
008A 346F
008B 3463
```

```
                        0501     retlw  'h'
                        0502     retlw  'i'
                        0503     retlw  'p'
                        0504     retlw  ' '
                        0505     retlw  'T'
                        0506     retlw  'e'
                        0507     retlw  'c'
                        0508     retlw  'h'
                        0509     retlw  'n'
                        0510     retlw  'o'
                        0511     retlw  'l'
                        0512     retlw  'o'
                        0513     retlw  'g'
                        0514     retlw  'y'
                        0515 Table_End
                        0516     retlw  0
                        0517 ;
                        0518     if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                        0519        MESSG  "Warning - User Definded: Table Table crosses page boundry in computed jump"
                        0520     endif
                        0521 ;
                        0522
                        0523
                        0524
                        0525     end
                        0526
                        0527
                        0528
                        0529
                        0530
008C 3468
008D 3469
008E 3470
008F 3420
0090 3454
0091 3465
0092 3463
0093 3468
0094 346E
0095 346F
0096 346C
0097 346F
0098 3467
0099 3479
009A 3400
```

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:53:47          PAGE 14

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXX-- ---------- ----------
00C0 : ---------- ---------- ---------- ----------

All other memory blocks unused.

Errors   :   0
Warnings :  13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

APPENDIX C: 4-BIT DATA INTERFACE, HIGH NIBBLE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:59:12                         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;
                     0012 ;
                     0013         include <C74_reg.h>
                     0233
                     0013
                     0014         include <lm032l.h>
                     0014
                     0015 ;
0001                 0016 Four_bit        EQU     TRUE        ; Selects 4- or 8-bit data transfers
0001                 0017 Data_HI         EQU     TRUE        ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;
                     0019 ;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;
                     0022 LCD_DATA        EQU     PORTB
                     0023 LCD_DATA_TRIS   EQU     TRISB
                     0024 ;
                     0025     else
                     0026 ;
0008                 0027 LCD_DATA        EQU     PORTD
0088                 0028 LCD_DATA_TRIS   EQU     TRISD
                     0029 ;
                     0030     endif
                     0031 ;
0005                 0032 LCD_CNTL        EQU     PORTA
                     0033 ;
                     0034 ;
```

```
0035 ;
0036 ; LCD Display Commands and Control Signal names.
0037 ;
0038     if ( Four_bit && !Data_HI )
0039 ;
0040 E                   EQU         0                   ; LCD Enable control line
0041 R_W                 EQU         1                   ; LCD Read/Write control line
0042 RS                  EQU         2                   ; LCD Register Select control
       line
0043 ;
0044     else
0045 ;
0046 E                   EQU         3                   ; LCD Enable control line
0047 R_W                 EQU         2                   ; LCD Read/Write control line
0048 RS                  EQU         1                   ; LCD Register Select control
       line
0049 ;
0050     endif
0051 ;
0052 ;
0053 TEMP1       EQU     0x030
0054 ;
0055         org     RESET_V             ; RESET vector location
0056 RESET       GOTO    START           ;
0057 ;
0058 ; This is the Periperal Interrupt routine. Should NOT get here
0059 ;
0060         org     ISR_V               ; Interrupt vector location
0061 PER_INT_V
0062 ERROR1      BCF     STATUS, RP0         ; Bank 0
0063             BSF     PORTC, 0
0064             BCF     PORTC, 0
0065             GOTO    ERROR1
0066 ;
0067 ;
0068 ;
0069 ;
0070 START       CLRF    STATUS              ; POWER_ON Reset (Beginning of program)
0071             CLRF    INTCON              ; Do initialization (Bank 0)
0072             CLRF    PIR1
0073             BSF     STATUS, RP0         ; Bank 1
0074             MOVLW   0x00                ; The LCD module does not like to work w/ weak pull-ups
0075             MOVWF   OPTION_R            ;
0076             CLRF    PIE1                ; Disable all peripheral interrupts
0077             MOVLW   0xFF                ;
0078             MOVWF   ADCON1              ; Port A is Digital.
0079 ;
0080 ;
0081 ;
```

```
0011 1283            BCF     STATUS, RP0     ; Bank 0
0012 0185            CLRF    PORTA           ; ALL PORT output should output Low.
0013 0186            CLRF    PORTB
0014 0187            CLRF    PORTC
0015 0188            CLRF    PORTD
0016 0189            CLRF    PORTE
0017 1010            BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                     ;
0018 1683            BSF     STATUS, RP0     ; Select Bank 1
0019 0185            CLRF    TRISA           ; RA5 -  0 outputs
001A 30F0            MOVLW   0xF0            ;
001B 0086            MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187            CLRF    TRISC           ; RC Port are outputs
001D 1407            BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188            CLRF    TRISD           ; RD Port are outputs
001F 0189            CLRF    TRISE           ; RE Port are outputs
0020 140C            BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781            BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283            BCF     STATUS, RP0     ; Select Bank 0
                     ;
                     ;
                     ; Initilize the LCD Display Module
                     ;
0023 0185            CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                     ;
                     DISPLAY_INIT
                         if ( Four_bit && !Data_HI )
0024 3020                    MOVLW   0x02            ; Command for 4-bit interface low nibble
                         endif
                     ;
                         if ( Four_bit && Data_HI )
0025 3020                    MOVLW   0x020           ; Command for 4-bit interface high nibble
                         endif
                     ;
                         if ( !Four_bit )
0026 3038                    MOVLW   0x038           ; Command for 8-bit interface
                         endif
                     ;
0027 0088                    MOVWF   LCD_DATA        ;
0028 1585                    BSF     LCD_CNTL, E     ;
0029 1185                    BCF     LCD_CNTL, E     ;
                     ;
                     ; This routine takes the calculated times that the delay loop needs to
                     ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                     ; frequency of operation. These uses registers before they are needed to
                     ; store the time.
                     ;
002A 3006            LCD_DELAY   MOVLW   LCD_INIT_DELAY  ;
```

```
                        MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
                        CLRF    LSD             ;
LOOP2                   DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
                        GOTO    LOOP2           ;
                        DECFSZ  MSD             ;
END_LCD_DELAY
                        GOTO    LOOP2           ;
;
; Command sequence for 2 lines of 5x7 characters
;
CMD_SEQ
;
    if ( Four_bit )
        if ( !Data_HI )
            MOVLW   0X02            ; 4-bit low nibble xfer
        else
            MOVLW   0X020           ; 4-bit high nibble xfer
        endif
;
    else
        MOVLW   0X038               ; 8-bit mode
    endif
;
    if ( Four_bit )
        MOVWF   LCD_DATA            ; This code for both 4-bit and 8-bit modes
        BSF     LCD_CNTL, E         ;
        BCF     LCD_CNTL, E         ;
;
    if ( Four_bit )                 ; This code for only 4-bit mode (2nd xfer)
        if ( !Data_HI )
            MOVLW   0x08            ; 4-bit low nibble xfer
        else
            MOVLW   0x080           ; 4-bit high nibble xfer
        endif
        MOVWF   LCD_DATA            ;
        BSF     LCD_CNTL, E         ;
        BCF     LCD_CNTL, E         ;
    endif
;
; Busy Flag should be valid after this point
;
        MOVLW   DISP_ON             ;
        CALL    SEND_CMD            ;
        MOVLW   CLR_DISP            ;
        CALL    SEND_CMD            ;
        MOVLW   ENTRY_INC           ;
        CALL    SEND_CMD            ;
        MOVLW   DD_RAM_ADDR         ;
        CALL    SEND_CMD            ;
```

```
0029 00B3
002A 01B4
002B 0BB4
002C 282B
002D 0BB3

002E 282B

002F 3020

0030 0088
0031 1585
0032 1185

0033 3080

0034 0088
0035 1585
0036 1185

0037 300C
0038 2074
0039 3001
003A 2074
003B 3006
003C 2074
003D 3080
003E 2074
```

```
0179 ;
0181 ;
0182 ;Send a message the hard way
0183                 movlw   'M'
0184                 call    SEND_CHAR
0185                 movlw   'i'
0186                 call    SEND_CHAR
0187                 movlw   'c'
0188                 call    SEND_CHAR
0189                 movlw   'r'
0190                 call    SEND_CHAR
0191                 movlw   'o'
0192                 call    SEND_CHAR
0193                 movlw   'c'
0194                 call    SEND_CHAR
0195                 movlw   'h'
0196                 call    SEND_CHAR
0197                 movlw   'i'
0198                 call    SEND_CHAR
0199                 movlw   'p'
0200                 call    SEND_CHAR
0201
0202                 movlw   B'11000000'     ;Address DDRam first character, second line
0203                 call    SEND_CMD
0204
0205 ;Demonstration of the use of a table to output a message
0206                 movlw   0               ;Table address of start of message
0207 dispmsg
0208                 movwf   TEMP1           ;TEMP1 holds start of message address
0209                 call    Table
0210                 andlw   0FFh            ;Check if at end of message (zero
0211                 btfsc   STATUS,Z        ;returned at end)
0212                 goto    out
0213                 call    SEND_CHAR       ;Display character
0214                 movf    TEMP1,w         ;Point to next character
0215                 addlw   1
0216                 goto    dispmsg
0217 out
0218 loop
0219                 goto    loop            ;Stay here forever
0220 ;
0221 ;
0222 INIT_DISPLAY
0223                 MOVLW   DISP_ON           ; Display On, Cursor On
0224                 CALL    SEND_CMD          ; Send This command to the Display Module
0225                 MOVLW   CLR_DISP          ; Clear the Display
0226                 CALL    SEND_CMD          ; Send This command to the Display Module
0227                 MOVLW   ENTRY_INC         ; Set Entry Mode Inc., No shift

003F 304D
0040 2065
0041 3069
0042 2065
0043 3063
0044 2065
0045 3072
0046 2065
0047 306F
0048 2065
0049 3063
004A 2065
004B 3068
004C 2065
004D 3069
004E 2065
004F 3070
0050 2065

0051 30C0
0052 2074

0053 3000

0054 00B0
0055 209B
0056 39FF
0057 1903
0058 285D
0059 2065
005A 0830
005B 3E01
005C 2854

005D 285D

005E 300C
005F 2074
0060 3001
0061 2074
0062 3006
```

```
0063 2074                       CALL    SEND_CMD            ; Send This command to the Display Module
0064 0008                       RETURN
                        ;
                        ;*****************************************************************************
                        ;* The LCD Module Subroutines                                                *
                        ;*****************************************************************************
                        ;
                          if ( Four_bit )     ; 4-bit Data transfers?
                        ;
                            if ( Data_HI )    ; 4-bit transfers on the high nibble of the PORT
                        ;
                        ;*****************************************************************************
                        ;*SendChar - Sends character to LCD                                         *
                        ;*This routine splits the character into the upper and lower               *
                        ;*nibbles and sends them to the LCD, upper nibble first.                   *
                        ;*****************************************************************************
                        ;
                        SEND_CHAR
0065 00B6                       MOVWF   CHAR                ;Character to be sent is in W
0066 2083                       CALL    BUSY_CHECK          ;Wait for LCD to be ready
0067 0836                       MOVF    CHAR, w
0068 39F0                       ANDLW   0xF0                ;Get upper nibble
0069 0088                       MOVWF   LCD_DATA            ;Send data to LCD
006A 1105                       BCF     LCD_CNTL, R_W       ;Set LCD to read
006B 1485                       BSF     LCD_CNTL, RS        ;Set LCD to data mode
006C 1585                       BSF     LCD_CNTL, E         ;toggle E for LCD
006D 1185                       BCF     LCD_CNTL, E
006E 0E36                       SWAPF   CHAR, w
006F 39F0                       ANDLW   0xF0                ;Get lower nibble
0070 0088                       MOVWF   LCD_DATA            ;Send data to LCD
0071 1585                       BSF     LCD_CNTL, E         ;toggle E for LCD
0072 1185                       BCF     LCD_CNTL, E
0073 0008                       RETURN
                        ;
                            else            ; 4-bit transfers on the low nibble of the PORT
                        ;
                        ;*****************************************************************************
                        ;* SEND_CHAR - Sends character to LCD                                        *
                        ;* This routine splits the character into the upper and lower                *
                        ;* nibbles and sends them to the LCD, upper nibble first.                    *
                        ;* The data is transmitted on the PORT<3:0> pins                             *
                        ;*****************************************************************************
                        ;
                        SEND_CHAR
0074                            MOVWF   CHAR                ; Character to be sent is in W
0075                            CALL    BUSY_CHECK          ; Wait for LCD to be ready
0076                            SWAPF   CHAR, W
```

```
0277            ANDLW   0x0F                ; Get upper nibble
0278            MOVWF   LCD_DATA            ; Send data to LCD
0279            BCF     LCD_CNTL, R_W       ; Set LCD to read
0280            BSF     LCD_CNTL, RS        ; Set LCD to data mode
0281            BSF     LCD_CNTL, E         ; toggle E for LCD
0282            BCF     LCD_CNTL, E
0283            MOVF    CHAR, W
0284            ANDLW   0x0F                ; Get lower nibble
0285            MOVWF   LCD_DATA            ; Send data to LCD
0286            BSF     LCD_CNTL, E         ; toggle E for LCD
0287            BCF     LCD_CNTL, E
0288            RETURN
0289 ;
0290        endif
0291    else
0292 ;
0293 ;*****************************************************************
0294 ;* SEND_CHAR - Sends character contained in register W to LCD    *
0295 ;* This routine sends the entire character to the PORT           *
0296 ;* The data is transmitted on the PORT<7:0> pins                 *
0297 ;*****************************************************************
0298 ;
0299 SEND_CHAR
0300            MOVWF   CHAR                ; Character to be sent is in W
0301            CALL    BUSY_CHECK          ; Wait for LCD to be ready
0302            MOVF    CHAR, W
0303            MOVWF   LCD_DATA            ; Send data to LCD
0304            BCF     LCD_CNTL, R_W       ; Set LCD in read mode
0305            BSF     LCD_CNTL, RS        ; Set LCD in data mode
0306            BSF     LCD_CNTL, E         ; toggle E for LCD
0307            BCF     LCD_CNTL, E
0308            RETURN
0309 ;
0310    endif
0311 ;
0313 ;
0314 ;*****************************************************************
0315 ;* SendCmd - Sends command to LCD                                *
0316 ;* This routine splits the command into the upper and lower      *
0317 ;* nibbles and sends them to the LCD, upper nibble first.        *
0318 ;* The data is transmitted on the PORT<3:0> pins                 *
0319 ;*****************************************************************
0320 ;
0321    if ( Four_bit )     ; 4-bit Data transfers?
0322 ;
0323        if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
0324 ;*****************************************************************
0325 ;*****************************************************************
```

```
0326 ;* SEND_CMD - Sends command to LCD                                  *
0327 ;* This routine splits the command into the upper and lower        *
0328 ;* nibbles and sends them to the LCD, upper nibble first.          *
0329 ;*************************************************************************
0330
0331 SEND_CMD
0332                MOVWF   CHAR            ; Character to be sent is in W
0333                CALL    BUSY_CHECK      ; Wait for LCD to be ready
0334                MOVF    CHAR,w
0335                ANDLW   0xF0            ; Get upper nibble
0336                MOVWF   LCD_DATA        ; Send data to LCD
0337                BCF     LCD_CNTL,R_W    ; Set LCD to read
0338                BCF     LCD_CNTL,RS     ; Set LCD to command mode
0339                BSF     LCD_CNTL,E      ; toggle E for LCD
0340                BCF     LCD_CNTL,E
0341                SWAPF   CHAR,w
0342                ANDLW   0xF0            ; Get lower nibble
0343                MOVWF   LCD_DATA        ; Send data to LCD
0344                BSF     LCD_CNTL,E      ; toggle E for LCD
0345                BCF     LCD_CNTL,E
0346                RETURN
0347 ;
0348            else                ; 4-bit transfers on the low nibble of the PORT
0349 ;
0350 SEND_CMD
0351                MOVWF   CHAR            ; Character to be sent is in W
0352                CALL    BUSY_CHECK      ; Wait for LCD to be ready
0353                SWAPF   CHAR, W
0354                ANDLW   0x0F            ; Get upper nibble
0355                MOVWF   LCD_DATA        ; Send data to LCD
0356                BCF     LCD_CNTL, R_W   ; Set LCD to read
0357                BCF     LCD_CNTL, RS    ; Set LCD to command mode
0358                BSF     LCD_CNTL, E     ; toggle E for LCD
0359                BCF     LCD_CNTL, E
0360                MOVF    CHAR, W
0361                ANDLW   0x0F            ; Get lower nibble
0362                MOVWF   LCD_DATA        ; Send data to LCD
0363                BSF     LCD_CNTL, E     ; toggle E for LCD
0364                BCF     LCD_CNTL, E
0365                RETURN
0366 ;
0367            endif
0368        else
0369 ;
0370 ;*************************************************************************
0371 ;* SEND_CND - Sends command contained in register W to LCD              *
0372 ;* This routine sends the entire character to the PORT                  *
0373 ;* The data is transmitted on the PORT<7:0> pins                        *

0074 00B6
0075 2083
0076 0836
0077 39F0
0078 0088
0079 1105
007A 1085
007B 1585
007C 1185
007D 0E36
007E 39F0
007F 0088
0080 1585
0081 1185
0082 0008
```

```
;*****************************************************************
;
SEND_CMD
                MOVWF   CHAR            ; Command to be sent is in W
                CALL    BUSY_CHECK      ; Wait for LCD to be ready
                MOVF    CHAR, w
                MOVWF   LCD_DATA        ; Send data to LCD
                BCF     LCD_CNTL, R_W   ; Set LCD in read mode
                BCF     LCD_CNTL, RS    ; Set LCD in command mode
                BSF     LCD_CNTL, E     ; toggle E for LCD
                BCF     LCD_CNTL, E
                RETURN
;
    endif
;
;
    if ( Four_bit )     ; 4-bit Data transfers?
;
        if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
;
;*****************************************************************
;* This routine checks the busy flag, returns when not busy      *
;*  Affects:                                                     *
;*    TEMP - Returned with busy/address                          *
;*****************************************************************
;
BUSY_CHECK
0083 1683       BSF     STATUS, RP0     ; Select Register page 1
0084 30FF       MOVLW   0xFF            ; Set Port_D for input
0085 0088       MOVWF   LCD_DATA_TRIS
0086 1283       BCF     STATUS, RP0     ; Select Register page 0
0087 1085       BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0088 1505       BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0089 1585       BSF     LCD_CNTL, E     ; Set E high
008A 1185       BCF     LCD_CNTL, E     ; Set E low
008B 0808       MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
008C 39F0       ANDLW   0xF0            ; Mask out lower nibble
008D 00B5       MOVWF   TEMP
008E 1585       BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
008F 1185       BCF     LCD_CNTL, E
0090 0E08       SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
0091 390F       ANDLW   0x0F            ; Mask out upper nibble
0092 04B5       IORWF   TEMP            ; Combine nibbles
0093 1BB5       BTFSC   TEMP, 7         ; Check busy flag, high = busy
0094 2883       GOTO    BUSY_CHECK      ; If busy, check again
0095 1105       BCF     LCD_CNTL, R_W
0096 1683       BSF     STATUS, RP0     ; Select Register page 1
0097 300F       MOVLW   0x0F
```

```
0098 0088                MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
0099 1283                BCF     STATUS, RP0     ; Select Register page 0
009A 0008                RETURN
                ;
                    else        ; 4-bit transfers on the low nibble of the PORT
                ;*****************************************************************
                ;* This routine checks the busy flag, returns when not busy      *
                ;*  Affects:                                                     *
                ;*    TEMP - Returned with busy/address                          *
                ;*****************************************************************
                ;
                BUSY_CHECK
                        BSF     STATUS, RP0     ; Bank 1
                        MOVLW   0xFF            ; Set PortB for input
                        MOVWF   LCD_DATA_TRIS
                        BCF     STATUS, RP0     ; Bank 0
                        BCF     LCD_CNTL, RS    ; Set LCD for Command mode
                        BSF     LCD_CNTL, R_W   ; Setup to read busy flag
                        BSF     LCD_CNTL, E     ; Set E high
                        BCF     LCD_CNTL, E     ; Set E low
                        SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
                        ANDLW   0xF0            ; Mask out lower nibble
                        MOVWF   TEMP            ;
                        BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
                        BCF     LCD_CNTL, E
                        MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
                        ANDLW   0x0F            ; Mask out upper nibble
                        IORWF   TEMP, F         ; Combine nibbles
                        BTFSC   TEMP, 7         ; Check busy flag, high = busy
                        GOTO    BUSY_CHECK      ; If busy, check again
                        BCF     LCD_CNTL, R_W
                        BSF     STATUS, RP0     ; Bank 1
                        MOVLW   0xF0            ;
                        MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
                        BCF     STATUS, RP0     ; Bank 0
                        RETURN
                ;
                    endif
                ;
                ;*****************************************************************
                ;* This routine checks the busy flag, returns when not busy      *
                ;*  Affects:                                                     *
                ;*    TEMP - Returned with busy/address                          *
                ;*****************************************************************
                ;
                BUSY_CHECK
```

```
0471                    BSF     STATUS,RP0      ; Select Register page 1
0472                    MOVLW   0xFF            ; Set port_D for input
0473                    MOVWF   LCD_DATA_TRIS
0474                    BCF     STATUS, RP0     ; Select Register page 0
0475                    BCF     LCD_CNTL, RS    ; Set LCD for command mode
0476                    BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0477                    BSF     LCD_CNTL, E     ; Set E high
0478                    BCF     LCD_CNTL, E     ; Set E low
0479                    MOVF    LCD_DATA, w     ; Read busy flag, DDram address
0480                    MOVWF   TEMP
0481                    BTFSC   TEMP, 7         ; Check busy flag, high=busy
0482                    GOTO    BUSY_CHECK
0483                    BCF     LCD_CNTL, R_W
0484                    BSF     STATUS, RP0     ; Select Register page 1
0485                    MOVLW   0x00
0486                    MOVWF   LCD_DATA_TRIS   ; Set port_D for output
0487                    BCF     STATUS, RP0     ; Select Register page 0
0488                    RETURN
0489 ;
0490     endif
0492 ;
0493 Table
0494                    addwf   PCL             ;Jump to char pointed to in W reg
009B 0782 0495          retlw   'M'
009C 344D 0496          retlw   'i'
009D 3469 0497          retlw   'c'
009E 3463 0498          retlw   'r'
009F 3472 0499          retlw   'o'
00A0 346F 0500          retlw   'c'
00A1 3463 0501          retlw   'h'
00A2 3468 0502          retlw   'i'
00A3 3469 0503          retlw   'p'
00A4 3470 0504          retlw   ','
00A5 3420 0505          retlw   'T'
00A6 3454 0506          retlw   'e'
00A7 3465 0507          retlw   'c'
00A8 3463 0508          retlw   'h'
00A9 3468 0509          retlw   'n'
00AA 346E 0510          retlw   'o'
00AB 346F 0511          retlw   'l'
00AC 346C 0512          retlw   'o'
00AD 346F 0513          retlw   'g'
00AE 3467 0514          retlw   'y'
00AF 3479 0515 Table_End
0516                    retlw   0
0517 ;
00B0 3400 0518          if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
0519                    MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
```

```
0520   endif
0521 ;
0522
0523
0524
0525   end
0526
0527
0528
0529
0530
```

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  0:59:12          PAGE 14
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X-------
00C0 : -------- -------- -------- --------

All other memory blocks unused.

Errors   :     0
Warnings :    13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

APPENDIX D: 4-BIT DATA INTERFACE, LOW NIBBLE LISTING

```
MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  5:29:26                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST P=16C64, F=INHX8M
                     0002 ;
                     0003 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     0004 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     0005 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     0006 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, R_W ).
                     0007 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     0008 ;
                     0009 ;       Program = LM032L.ASM
                     0010 ;       Revision Date:   5-10-94
                     0011 ;
                     0012 ;
                     0013        include <C74_reg.h>
                     0233
                     0013
                     0014        include <lm032l.h>
                     0014
                     0015 ;
0001                 0016 Four_bit         EQU     TRUE       ; Selects 4- or 8-bit data transfers
0000                 0017 Data_HI          EQU     FALSE       ; If 4-bit transfers, Hi or Low nibble of PORT
                     0018 ;
                     0019 ;
                     0020     if ( Four_bit && !Data_HI )
                     0021 ;
0006                 0022 LCD_DATA         EQU     PORTB
0086                 0023 LCD_DATA_TRIS    EQU     TRISB
                     0024 ;
                     0025     else
                     0026 ;
                     0027 LCD_DATA         EQU     PORTD
                     0028 LCD_DATA_TRIS    EQU     TRISD
                     0029 ;
                     0030     endif
                     0031 ;
0005                 0032 LCD_CNTL         EQU     PORTA
                     0033 ;
                     0034 ;
                     0035 ;
```

```
0036 ; LCD Display Commands and Control Signal names.
0037 ;
0038 ;    if ( Four_bit && !Data_HI )
0039 ;
0000                        EQU         0           ; LCD Enable control line
0001 R_W                    EQU         1           ; LCD Read/Write control line
0002 RS                     EQU         2           ; LCD Register Select control
     line
0043 ;    else
0044 ;
0045 ;
0046 E                      EQU         3           ; LCD Enable control line
0047 R_W                    EQU         2           ; LCD Read/Write control line
0048 RS                     EQU         1           ; LCD Register Select control
     line
0049 ;    endif
0050 ;
0051 ;
0052 ;
0053 TEMP1      EQU     0x030
0054 ;
0055            org     RESET_V                     ; RESET vector location
0056 RESET      GOTO    START                       ;
0057 ;
0058 ; This is the Periperal Interrupt routine. Should NOT get here
0059 ;
0061            org     ISR_V                       ; Interrupt vector location
0062 PER_INT_V
0063 ERROR1     BCF     STATUS, RP0                 ; Bank 0
0064            BSF     PORTC, 0
0065            BCF     PORTC, 0
0066            GOTO    ERROR1
0067 ;
0068 ;
0069 ;
0070 START                                           ; POWER_ON Reset (Beginning of program)
0071            CLRF    STATUS                      ; Do initialization (Bank 0)
0072            CLRF    INTCON                      ;
0073            CLRF    PIR1
0074            BSF     STATUS, RP0                 ; Bank 1
0075            MOVLW   0x00                        ; The LCD module does not like to work w/ weak pull-ups
0076            MOVWF   OPTION_R                    ;
0077            CLRF    PIE1                        ; Disable all peripheral interrupts
0078            MOVLW   0xFF                        ;
0079            MOVWF   ADCON1                      ; Port A is Digital.
0080 ;
0081 ;
0082            BCF     STATUS, RP0                 ; Bank 0
```

```
0012 0185                       CLRF    PORTA                   ; ALL PORT output should output Low.
0013 0186                       CLRF    PORTB
0014 0187                       CLRF    PORTC
0015 0188                       CLRF    PORTD
0016 0189                       CLRF    PORTE
0017 1010                       BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
0088 ;
0018 1683                       BSF     STATUS, RP0     ; Select Bank 1
0019 0185                       CLRF    TRISA           ; RA5 -  0 outputs
001A 30F0                       MOVLW   0xF0            ;
001B 0086                       MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
001C 0187                       CLRF    TRISC           ; RC Port are outputs
001D 1407                       BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
001E 0188                       CLRF    TRISD           ; RD Port are outputs
001F 0189                       CLRF    TRISE           ; RE Port are outputs
0020 140C                       BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0021 1781                       BSF     OPTION_R, RBPU  ; Disable PORTB pull-ups
0022 1283                       BCF     STATUS, RP0     ; Select Bank 0
0100 ;
0101 ;
0103 ;
0104 ; Initilize the LCD Display Module
0105 ;
0023 0185                       CLRF    LCD_CNTL        ; ALL PORT output should output Low.
0107
0108 DISPLAY_INIT
0109                    if ( Four_bit && !Data_HI )
0024 3002                       MOVLW   0x02            ; Command for 4-bit interface low nibble
0111                    endif
0112 ;
0113                    if ( Four_bit && Data_HI )
0114                            MOVLW   0x020           ; Command for 4-bit interface high nibble
0115                    endif
0116 ;
0117                    if ( !Four_bit )
0118                            MOVLW   0x038           ; Command for 8-bit interface
0119                    endif
0120 ;
0025 0086                       MOVWF   LCD_DATA        ;
0026 1405                       BSF     LCD_CNTL, E     ;
0027 1005                       BCF     LCD_CNTL, E     ;
0124 ;
0125 ; This routine takes the calculated times that the delay loop needs to
0126 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
0127 ; frequency of operation. These uses registers before they are needed to
0128 ; store the time.
0129 ;
0028 3006 LCD_DELAY     MOVLW   LCD_INIT_DELAY  ;
0029 00B3                       MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
0131
```

```
                        CLRF    LSD             ;
                        DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
                        GOTO    LOOP2           ;
                        DECFSZ  MSD             ;
                END_LCD_DELAY
                        GOTO    LOOP2           ;
                ;
                ; Command sequence for 2 lines of 5x7 characters
                ;
                CMD_SEQ
                ;
                    if ( Four_bit )
                        if ( !Data_HI )
                            MOVLW   0X02        ; 4-bit low nibble xfer
                        else
                            MOVLW   0X020       ; 4-bit high nibble xfer
                        endif
                ;
                    else
                        MOVLW   0X038           ; 8-bit mode
                    endif
                ;
                    if ( Four_bit )
                        MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
                        BSF     LCD_CNTL, E     ;
                        BCF     LCD_CNTL, E     ;
                ;
                    if ( Four_bit )             ; This code for only 4-bit mode (2nd xfer)
                        if ( !Data_HI )
                            MOVLW   0x08        ; 4-bit low nibble xfer
                        else
                            MOVLW   0x080       ; 4-bit high nibble xfer
                        endif
                        MOVWF   LCD_DATA        ;
                        BSF     LCD_CNTL, E     ;
                        BCF     LCD_CNTL, E     ;
                    endif
                ;
                ; Busy Flag should be valid after this point
                ;
                        MOVLW   DISP_ON         ;
                        CALL    SEND_CMD        ;
                        MOVLW   CLR_DISP        ;
                        CALL    SEND_CMD        ;
                        MOVLW   ENTRY_INC       ;
                        CALL    SEND_CMD        ;
                        MOVLW   DD_RAM_ADDR     ;
                        CALL    SEND_CMD        ;
                ;
```

```
                      0181 ;
                      0182 ;Send a message the hard way
003F 304D             0183         movlw   'M'
0040 2065             0184         call    SEND_CHAR
0041 3069             0185         movlw   'i'
0042 2065             0186         call    SEND_CHAR
0043 3063             0187         movlw   'c'
0044 2065             0188         call    SEND_CHAR
0045 3072             0189         movlw   'r'
0046 2065             0190         call    SEND_CHAR
0047 306F             0191         movlw   'o'
0048 2065             0192         call    SEND_CHAR
0049 3063             0193         movlw   'c'
004A 2065             0194         call    SEND_CHAR
004B 3068             0195         movlw   'h'
004C 2065             0196         call    SEND_CHAR
004D 3069             0197         movlw   'i'
004E 2065             0198         call    SEND_CHAR
004F 3070             0199         movlw   'p'
0050 2065             0200         call    SEND_CHAR
                      0201
0051 30C0             0202         movlw   B'11000000'     ;Address DDRam first character, second line
0052 2074             0203         call    SEND_CMD
                      0204
                      0205 ;Demonstration of the use of a table to output a message
0053 3000             0206         movlw   0               ;Table address of start of message
                      0207 dispmsg
0054 00B0             0208         movwf   TEMP1           ;TEMP1 holds start of message address
0055 209B             0209         call    Table
0056 39FF             0210         andlw   0FFh            ;Check if at end of message (zero
0057 1903             0211         btfsc   STATUS,Z        ;returned at end)
0058 285D             0212         goto    out
0059 2065             0213         call    SEND_CHAR       ;Display character
005A 0830             0214         movf    TEMP1,w         ;Point to next character
005B 3E01             0215         addlw   1
005C 2854             0216         goto    dispmsg
                      0217 out
                      0218 loop
005D 285D             0219         goto    loop            ;Stay here forever
                      0220 ;
                      0221 ;
                      0222 INIT_DISPLAY
005E 300C             0223         MOVLW   DISP_ON          ; Display On, Cursor On
005F 2074             0224         CALL    SEND_CMD         ; Send This command to the Display Module
0060 3001             0225         MOVLW   CLR_DISP         ; Clear the Display
0061 2074             0226         CALL    SEND_CMD         ; Send This command to the Display Module
0062 3006             0227         MOVLW   ENTRY_INC        ; Set Entry Mode Inc., No shift
0063 2074             0228         CALL    SEND_CMD         ; Send This command to the Display Module
```

```
0064 0008                       RETURN
0229                    ;
0230                    ;*****************************************************************
0231                    ;*****************************************************************
0232                    ;*                                                               *
0233                    ;* The LCD Module Subroutines                                    *
0234                    ;*                                                               *
0235                    ;*****************************************************************
0236                    ;*****************************************************************
0237                    ;
0238                    ;   if ( Four_bit )     ; 4-bit Data transfers?
0239                    ;
0240                    ;       if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
0241                    ;*****************************************************************
0242                    ;*SendChar - Sends character to LCD                             *
0243                    ;*This routine splits the character into the upper and lower    *
0244                    ;*nibbles and sends them to the LCD, upper nibble first.        *
0245                    ;*****************************************************************
0246                    ;
0247                    SEND_CHAR
0248 0B86 2083             MOVWF   CHAR            ;Character to be sent is in W
0249 0B87 2083             CALL    BUSY_CHECK      ;Wait for LCD to be ready
0250 0B88 0E36             MOVF    CHAR, w
0251 0B89 390F             ANDLW   0xF0            ;Get upper nibble
0252 0B8A 2083             MOVWF   LCD_DATA        ;Send data to LCD
0253 0B8B 1283             BCF     LCD_CNTL, R_W   ;Set LCD to read
0254 0B8C 1683             BSF     LCD_CNTL, RS    ;Set LCD to data mode
0255 0B8D 1683             BSF     LCD_CNTL, E     ;toggle E for LCD
0256 0B8E 1283             BCF     LCD_CNTL, E
0257 0B8F 0E36             SWAPF   CHAR, w
0258 0B90 390F             ANDLW   0xF0            ;Get lower nibble
0259 0B91 2083             MOVWF   LCD_DATA        ;Send data to LCD
0260 0B92 1683             BSF     LCD_CNTL, E     ;toggle E for LCD
0261 0B93 1283             BCF     LCD_CNTL, E
0262 0B94 0008             RETURN
0263                    ;
0264                    ;       else            ; 4-bit transfers on the low nibble of the PORT
0265                    ;*****************************************************************
0266                    ;*****************************************************************
0267                    ;* SEND_CHAR - Sends character to LCD                            *
0268                    ;* This routine splits the character into the upper and lower    *
0269                    ;* nibbles and sends them to the LCD, upper nibble first.        *
0270                    ;* The data is transmitted on the PORT<3:0> pins                 *
0271                    ;*****************************************************************
0272                    ;
0273                    SEND_CHAR
0274 0B95 2083             MOVWF   CHAR            ; Character to be sent is in W
0275 0B96 2083             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0276 0B97 0E36             SWAPF   CHAR, W
0277 0B98 390F             ANDLW   0x0F            ; Get upper nibble
```

```
0069 0086               MOVWF   LCD_DATA           ; Send data to LCD
006A 1085               BCF     LCD_CNTL, R_W      ; Set LCD to read
006B 1505               BSF     LCD_CNTL, RS       ; Set LCD to data mode
006C 1405               BSF     LCD_CNTL, E        ; toggle E for LCD
006D 1005               BCF     LCD_CNTL, E
006E 0836               MOVF    CHAR, W
006F 390F               ANDLW   0x0F               ; Get lower nibble
0070 0086               MOVWF   LCD_DATA           ; Send data to LCD
0071 1405               BSF     LCD_CNTL, E        ; toggle E for LCD
0072 1005               BCF     LCD_CNTL, E
0073 0008               RETURN
                        ;
                            endif
                            else
                        ;
                        ;*****************************************************************
                        ;* SEND_CHAR - Sends character contained in register W to LCD    *
                        ;* This routine sends the entire character to the PORT           *
                        ;* The data is transmitted on the PORT<7:0> pins                 *
                        ;*****************************************************************
                        ;
                        SEND_CHAR
0300                            MOVWF   CHAR               ; Character to be sent is in W
0301                            CALL    BUSY_CHECK         ; Wait for LCD to be ready
0302                            MOVF    CHAR, w
0303                            MOVWF   LCD_DATA           ; Send data to LCD
0304                            BCF     LCD_CNTL, R_W      ; Set LCD in read mode
0305                            BSF     LCD_CNTL, RS       ; Set LCD in data mode
0306                            BSF     LCD_CNTL, E        ; toggle E for LCD
0307                            BCF     LCD_CNTL, E
0308                            RETURN
0309                        ;
0310                            endif
0311                        ;
0312                        ;
0313                        ;*****************************************************************
0314                        ;* SendCmd - Sends command to LCD                                *
0315                        ;* This routine splits the command into the upper and lower      *
0316                        ;* nibbles and sends them to the LCD, upper nibble first.        *
0317                        ;* The data is transmitted on the PORT<3:0> pins                 *
0318                        ;*****************************************************************
0319                        ;
0320                        ;
0321                            if ( Four_bit )     ; 4-bit Data transfers?
0322                        ;
0323                                if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
0324                        ;*****************************************************************
0325                        ;*****************************************************************
```

```
0326 ;* SEND_CMD - Sends command to LCD                                  *
0327 ;* This routine splits the command into the upper and lower        *
0328 ;* nibbles and sends them to the LCD, upper nibble first.          *
0329 ;*****************************************************************
0330
0331 SEND_CMD
0332                 MOVWF   CHAR            ; Character to be sent is in W
0333                 CALL    BUSY_CHECK      ; Wait for LCD to be ready
0334                 MOVF    CHAR,w
0335                 ANDLW   0xF0            ; Get upper nibble
0336                 MOVWF   LCD_DATA        ; Send data to LCD
0337                 BCF     LCD_CNTL,R_W    ; Set LCD to read
0338                 BCF     LCD_CNTL,RS     ; Set LCD to command mode
0339                 BSF     LCD_CNTL,E      ; toggle E for LCD
0340                 BCF     LCD_CNTL,E
0341                 SWAPF   CHAR,w
0342                 ANDLW   0xF0            ; Get lower nibble
0343                 MOVWF   LCD_DATA        ; Send data to LCD
0344                 BSF     LCD_CNTL,E      ; toggle E for LCD
0345                 BCF     LCD_CNTL,E
0346                 RETURN
0347 ;
0348         else            ; 4-bit transfers on the low nibble of the PORT
0349 ;
0350 SEND_CMD
0351                 MOVWF   CHAR            ; Character to be sent is in W
0352                 CALL    BUSY_CHECK      ; Wait for LCD to be ready
0353                 SWAPF   CHAR, W
0354                 ANDLW   0x0F            ; Get upper nibble
0355                 MOVWF   LCD_DATA        ; Send data to LCD
0356                 BCF     LCD_CNTL, R_W   ; Set LCD to read
0357                 BCF     LCD_CNTL, RS    ; Set LCD to command mode
0358                 BSF     LCD_CNTL, E     ; toggle E for LCD
0359                 BCF     LCD_CNTL, E
0360                 MOVF    CHAR, W
0361                 ANDLW   0x0F            ; Get lower nibble
0362                 MOVWF   LCD_DATA        ; Send data to LCD
0363                 BSF     LCD_CNTL, E     ; toggle E for LCD
0364                 BCF     LCD_CNTL, E
0365                 RETURN
0366 ;
0367         endif
0368     else
0369 ;
0370 ;*****************************************************************
0371 ;* SEND_CND - Sends command contained in register W to LCD       *
0372 ;* This routine sends the entire character to the PORT           *
0373 ;* The data is transmitted on the PORT<7:0> pins                 *

0074 00B6
0075 2083
0076 0E36
0077 390F
0078 0086
0079 1085
007A 1105
007B 1405
007C 1005
007D 0836
007E 390F
007F 0086
0080 1405
0081 1005
0082 0008
```

```
;*****************************************************************
;
SEND_CMD
                MOVWF   CHAR                    ; Command to be sent is in W
                CALL    BUSY_CHECK              ; Wait for LCD to be ready
                MOVF    CHAR, w
                MOVWF   LCD_DATA                ; Send data to LCD
                BCF     LCD_CNTL, R_W           ; Set LCD in read mode
                BCF     LCD_CNTL, RS            ; Set LCD in command mode
                BSF     LCD_CNTL, E             ; toggle E for LCD
                BCF     LCD_CNTL, E
                RETURN
;
    endif
;
;
    if ( Four_bit )     ; 4-bit Data transfers?
;
        if ( Data_HI )  ; 4-bit transfers on the high nibble of the PORT
;
;*****************************************************************
;* This routine checks the busy flag, returns when not busy      *
;*  Affects:                                                     *
;*     TEMP - Returned with busy/address                         *
;*****************************************************************
;
BUSY_CHECK
                BSF     STATUS, RP0             ; Select Register page 1
                MOVLW   0xFF                    ; Set Port_D for input
                MOVWF   LCD_DATA_TRIS
                BCF     STATUS, RP0             ; Select Register page 0
                BCF     LCD_CNTL, RS            ; Set LCD for Command mode
                BSF     LCD_CNTL, R_W           ; Setup to read busy flag
                BSF     LCD_CNTL, E             ; Set E high
                BCF     LCD_CNTL, E             ; Set E low
                MOVF    LCD_DATA, W             ; Read upper nibble busy flag, DDRam address
                ANDLW   0xF0                    ; Mask out lower nibble
                MOVWF   TEMP
                BSF     LCD_CNTL, E             ; Toggle E to get lower nibble
                BCF     LCD_CNTL, E
                SWAPF   LCD_DATA, w             ; Read lower nibble busy flag, DDRam address
                ANDLW   0x0F                    ; Mask out upper nibble
                IORWF   TEMP                    ; Combine nibbles
                BTFSC   TEMP, 7                 ; Check busy flag, high = busy
                GOTO    BUSY_CHECK              ; If busy, check again
                BCF     LCD_CNTL, R_W
                BSF     STATUS, RP0             ; Select Register page 1
                MOVLW   0x0F
```

```
0423                    MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
0424                    BCF     STATUS, RP0     ; Select Register page 0
0425                    RETURN
0426 ;
0427                ; 4-bit transfers on the low nibble of the PORT
0428 ;
0429 ;*************************************************************************
0430 ;* This routine checks the busy flag, returns when not busy              *
0431 ;*  Affects:                                                             *
0432 ;*     TEMP - Returned with busy/address                                 *
0433 ;*************************************************************************
0434 ;
0435 BUSY_CHECK
0436                    BSF     STATUS, RP0     ; Bank 1
0437                    MOVLW   0xFF            ; Set PortB for input
0438                    MOVWF   LCD_DATA_TRIS
0439                    BCF     STATUS, RP0     ; Bank 0
0440                    BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0441                    BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0442                    BSF     LCD_CNTL, E     ; Set E high
0443                    BCF     LCD_CNTL, E     ; Set E low
0444                    SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
0445                    ANDLW   0xF0            ; Mask out lower nibble
0446                    MOVWF   TEMP            ;
0447                    BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0448                    BCF     LCD_CNTL, E
0449                    MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
0450                    ANDLW   0x0F            ; Mask out upper nibble
0451                    IORWF   TEMP, F         ; Combine nibbles
0452                    BTFSC   TEMP, 7         ; Check busy flag, high = busy
0453                    GOTO    BUSY_CHECK      ; If busy, check again
0454                    BCF     LCD_CNTL, R_W
0455                    BSF     STATUS, RP0     ; Bank 1
0456                    MOVLW   0xF0            ;
0457                    MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
0458                    BCF     STATUS, RP0     ; Bank 0
0459                    RETURN
0460 ;
0461                endif
0462                else
0463 ;
0464 ;*************************************************************************
0465 ;* This routine checks the busy flag, returns when not busy              *
0466 ;*  Affects:                                                             *
0467 ;*     TEMP - Returned with busy/address                                 *
0468 ;*************************************************************************
0469 ;
0470 BUSY_CHECK

0083 1683
0084 30FF
0085 0086
0086 1283
0087 1105
0088 1485
0089 1405
008A 1005
008B 0E06
008C 39F0
008D 00B5
008E 1405
008F 1005
0090 0806
0091 390F
0092 04B5
0093 1BB5
0094 2883
0095 1085
0096 1683
0097 30F0
0098 0086
0099 1283
009A 0008
```

```
                      BSF     STATUS,RP0      ; Select Register page 1
                      MOVLW   0xFF            ; Set port_D for input
                      MOVWF   LCD_DATA_TRIS
                      BCF     STATUS, RP0     ; Select Register page 0
                      BCF     LCD_CNTL, RS    ; Set LCD for command mode
                      BSF     LCD_CNTL, R_W   ; Setup to read busy flag
                      BSF     LCD_CNTL, E     ; Set E high
                      BCF     LCD_CNTL, E     ; Set E low
                      MOVF    LCD_DATA, w     ; Read busy flag, DDram address
                      MOVWF   TEMP
                      BTFSC   TEMP, 7         ; Check busy flag, high=busy
                      GOTO    BUSY_CHECK
                      BCF     LCD_CNTL, R_W
                      BSF     STATUS, RP0     ; Select Register page 1
                      MOVLW   0x00
                      MOVWF   LCD_DATA_TRIS   ; Set port_D for output
                      BCF     STATUS, RP0     ; Select Register page 0
                      RETURN
                  ;
                      endif
                  ;
                  Table
009B 0782             addwf   PCL     ;Jump to char pointed to in W reg
009C 344D             retlw   'M'
009D 3469             retlw   'i'
009E 3463             retlw   'c'
009F 3472             retlw   'r'
00A0 346F             retlw   'o'
00A1 3463             retlw   'c'
00A2 3468             retlw   'h'
00A3 3469             retlw   'i'
00A4 3470             retlw   'p'
00A5 3420             retlw   ' '
00A6 3454             retlw   'T'
00A7 3465             retlw   'e'
00A8 3463             retlw   'c'
00A9 3468             retlw   'h'
00AA 346E             retlw   'n'
00AB 346F             retlw   'o'
00AC 346C             retlw   'l'
00AD 346F             retlw   'o'
00AE 3467             retlw   'g'
00AF 3479             retlw   'y'
                  Table_End
00B0 3400             retlw   0
                  ;
                      if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                      MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
```

```
0520    endif
0521 ;
0522
0523
0524
0525    end
0526
0527
0528
0529
0530


MPASM 00.00.68 Intermediate    LM032L.ASM   6-8-1994  5:29:26          PAGE 14


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---------- ---------- ---------- ---------- ---------- ----------
0040 : ---------- ---------- ---------- ---------- ---------- ----------

0080 : ---------- ---------- ---------- ---------- ---------- X---------
00C0 : ---------- ---------- ---------- ----------

All other memory blocks unused.

Errors   :   0
Warnings :  13
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

APPENDIX E: LM032L.H

```
                    nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;                   PROGRAM:       CLOCK.H
;                   Revision:      5-10-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq            EQU             D'4000000'     ; Device Frequency is 4 MHz
DB_HI_BYTE          EQU             (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY      EQU             (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR          EQU             40             ; RAM Location
OUTER_CNTR          EQU             41             ; RAM Location

T1OSO               EQU             0              ; The RC0 / T1OSO / T1CKI
;
RESET_V             EQU             0x0000         ; Address of RESET Vector
ISR_V               EQU             0x0004         ; Address of Interrupt Vector
PMEM_END            EQU             0x07FF         ; Last address in Program Memory
TABLE_ADDR          EQU             0x0400         ; Address where to start Tables
;
HR_MIN_SW           EQU             0x7            ; The switch to select the units
INC_SW              EQU             0x6            ; The switch to increment the selected units
CLR_MIN_SW          EQU             0x5            ; The switch to clear the minutes and seconds
;
FLAG_REG            EQU             0x020          ; Register which contains flag bits
;
;  +------+-----+-----+-----------+-----+-----+----------+---------+
;  |  AM  |  -  |  -  | KEY_INPUT |  -  |  -  | MIN_UNIT | HR_UNIT |
;  +------+-----+-----+-----------+-----+-----+----------+---------+
;
AM                  EQU             0x07           ; Flag to specify if AM or PM
;
KEY_INPUT           EQU             0x04           ; Flag to specify if doing key inputs
;
MIN_UNIT            EQU             0x01           ; Flags to specify which units to operate on
HR_UNIT             EQU             0x00           ;  (HRS, MIN, or none)
;
HRS                 EQU             0x030          ; Holds counter value for HOURS
MIN                 EQU             0x031          ; Holds counter value for MINUTES
SECS                EQU             0x032          ; Holds counter value for SECONDS
```

```
MSD             EQU     0x033   ; Temporary register, Holds Most Significant Digit of BIN to BCD conversion
LSD             EQU     0x034   ; Temporary register, Holds Least Significant Digit of BIN to BCD conversion
TEMP            EQU     0x035   ; Temporary register
CHAR            EQU     0x036   ; Temporary register, Holds value to send to LCD module.
;
WAIT_CNTR       EQU     0x040   ; Counter that holds wait time for key inputs
;
; LCD Module commands
;
DISP_ON         EQU     0x00C   ; Display on
DISP_ON_C       EQU     0x00E   ; Display on, Cursor on
DISP_ON_B       EQU     0x00F   ; Display on, Cursor on, Blink cursor
DISP_OFF        EQU     0x008   ; Display off
CLR_DISP        EQU     0x001   ; Clear the Display
ENTRY_INC       EQU     0x006   ;
ENTRY_INC_S     EQU     0x007   ;
ENTRY_DEC       EQU     0x004   ;
ENTRY_DEC_S     EQU     0x005   ;
DD_RAM_ADDR     EQU     0x080   ; Least Significant 7-bit are for address
DD_RAM_UL       EQU     0x080   ; Upper Left coner of the Display
;
                list
```

NOTES:

MICROCHIP

AN580

Using Timer1 in Asynchronous Clock Mode

INTRODUCTION

This application note discusses the use of the Timer1 module, of the PIC16CXX family, for an asynchronous clock. The Timer1 module has it own oscilator circuitry, which allows the timer to keep real time, even when the device is in sleep mode. When the device is in sleep, the oscillator will continue to increment Timer1. An overflow of Timer1 causes a timer1 interrupt (if enabled) and will wake the processor from sleep. The interrupt service routine, then can do the desired task.

OVERVIEW

Timer1 is a 16-bit counter with a 2-bit prescaler. Timer1 can be incremented from either the internal clock, an external clock, or an external oscillator. Timer1 can be configured to synchronize or not synchronize the external clock sources. The asynchronous operation allows timer1 to increment when the device is in sleep. Figure 1 is a block diagram of Timer1.

To set up Timer1 for asynchronous operation the timer1 control register (T1CON) must have the following bits configured:

- TMR1CS set (external clock source)
- T1CKS<1:0> configured for the desired prescaler
- T1INSYNC set (asynchronous operation)
- TMR1ON set (enables Timer1)
- Set T1OSCEN, if using an external oscilator

In asynchronous operation, if the clock source is an external clock, the clock must be on the T1CKI pin. If the clock source is a crystal oscillator, the crystal is connected accross the T1OSO and T1OSI pins. Please refer to the device Data Sheet for recommended capacitor selection for the Timer1 oscillator.

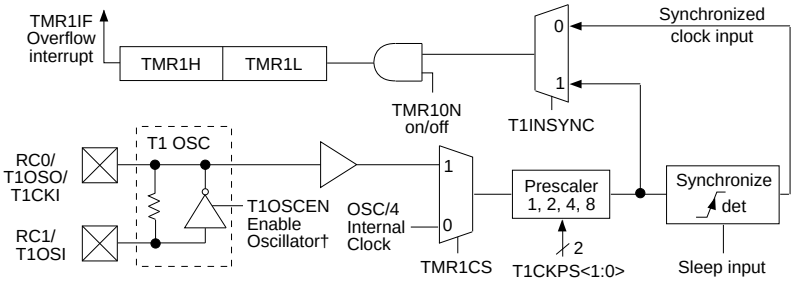
When using Timer1 in asynchronous mode, the use of an external clock minimizes the operating and sleep currents. This is because the oscillator circuitry is disabled. Though the external clock may give the lower device currents, the use of a crystal oscillator may lead to lower system currant consumption and system cost.

System current consumption can also be reduced by having the timer1 overflow interrupt wake the processor from sleep at the desired interval, With a 32.768 KHz crystal, Timer1's overflow rate ranges from 2 to 16 seconds, depending on the prescaler chosen. Table 1 shows timer1 overflow times for various crystal frequencies and prescaler values.

TABLE 1: TIMER1 OVERFLOW TIMES

	Frequency (KHz)		
Prescaler	**32.768**	**100**	**200**
1	2 Seconds	0.655	0.327 Seconds
2	4 Seconds	1.31 Seconds	0.655 Seconds
4	8 Seconds	2.62 Seconds	1.31 Seconds
8	16 Seconds	5.24 Seconds	2.62 Seconds

FIGURE 1: TIMER1 BLOCK DIAGRAM



† When the T1 OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Using Timer1 in Asynchronous Clock Mode

As can be seen the 32 Khz crystal, gives very nice overflow rates. These crystals, refered to as watch crystals, also can be relativly inexpensive. In many applications the 2 second overflow time, of a 32 KHz crystal, is too long. An easy way to reduce the the overflow time is during the interrupt service routine, to load the TMR1H register with a value. Table 2 shows the overflow times, depending on the value loaded into the TMR1H register and a prescaler of 1.

TABLE 2: TMR1H LOAD VALUES / TIMER1 OVERFLOW TIMES

TMR1H Load Value	Overflow Time (@ 32.768 KHz)
80 h	1 Second
C0 h	0.5 Seconds
E0 h	0.25 Seconds
F0 h	0.125 Seconds

Note: The loading of either TMR1H or TMR1L causes the prescaler to be cleared. When Timer1 is in operation, extreme care should be taken in modifying either the TMR1H or TMR1L registers, since this automatically modifies the prescaler to 1.

The code segment shown in Example 1 initializes the Timer1 module for asynchronous mode, enables the Timer1 interrupt, and the interrupt service routine loads the TMR1H register with a value.

CONCLUSION

Timer1 gives designers a powerful timebase function. The asynchronous operation and oscillator circuitry gives designers the ability to easily keep real time, while minimizing power consumption and external logic.

Author: Mark Palmer - Sr. Application Engineer
Logic Products Division

EXAMPLE 1: TIMER1 CODE SEGMENT FOR ASYNCHRONOUS OPERATION

```
      org   0x000
Reset_V     GOTO    START
;
      org   0x004
PER_INT_V
            BCF     STATUS, RP0    ; Bank 0
            BTFSC   PIR1, TMR1IF   ; Timer 1 overflowed?
            GOTO    T1_OVRFL       ; YES, Service the Timer1 Overflow Interrupt
;
; Should NEVER get here
;
ERROR1                              ; NO, Unknown Interrupt Source
            BSF     PORTD, 1        ; Toggle a port pin to indicate error
            BCF     PORTD, 1
            GOTO    ERROR1
;
T1_OVRFL
            BCF     PIR1, TMR1IF   ; Clear Timer 1 Interrupt Flag
            MOVLW   0x80           ; Since doing key inputs, clear TMR1
            MOVWF   TMR1H          ;   for 1 sec overflow.
            :
            :                      ; Do Interrupt stuff here
            :
            RETFIE                 ; Return / Enable Global Interrupts
;
;
START                               ; POWER_ON Reset (Beginning of program)
            CLRF    STATUS         ; Do initialization (Bank 0)
            BCF     T1CON, TMR1ON  ; Timer 1 is NOT incrementing
            :
            :                      ; Do Initialization stuff here
            :
            MOVLW   0x80           ; TIM1H:TMR1L = 0x8000 gives 1 second
            MOVWF   TMR1H          ;   overflow, at 32 KHz.
            CLRF    TMR1L          ;
;
            CLRF    INTCON
            CLRF    PIR1
            BSF     STATUS, RP0    ; Bank 1
            CLRF    PIE1           ; Disable all peripheral interrupts
;
    if ( C74_REV_A )               ; See PIC16C74 Errata
            BSF     TRISC, T1OSO   ; RC0 needs to be input for the oscillator to function
    endif
            BSF     PIE1, TMR1IE   ; Enable TMR1 Interrupt
;
; Initialize the Special Function Registers (SFR) interrupts
;
            BCF     STATUS, RP0    ; Bank 0
            CLRF    PIR1           ;
            BSF     INTCON, PEIE   ; Enable Peripheral Interrupts
            BSF     INTCON, GIE    ; Enable all Interrupts
;
            MOVLW   0x0E
            MOVWF   T1CON          ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
            BSF     T1CON, TMR1ON  ; Turn Timer 1 ON
;
zzz         SLEEP
            GOTO    zzz            ; Sleep, wait for TMR1 interrupt
```


NOTES:

AN582

Low-Power Real Time Clock

INTRODUCTION

This application note implements a low-power real time clock using the Timer1 module of the PIC16CXX family of processors. Timer1 can operate from its own crystal source, which allows the timer to increment while the device is in sleep mode. The device is placed into sleep to minimize the current consumption. Only the events that require processing will wake the device from sleep. These are a key input and a Timer1 overflow.

OPERATION

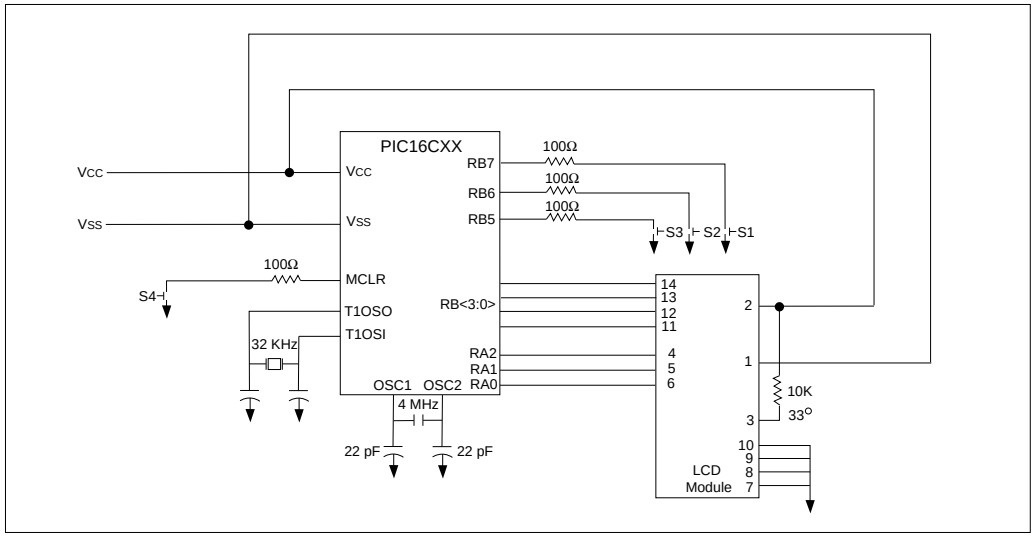
Upon power-up, the device goes into an initial state. This state sets the display to 12:00 PM and waits for Timer1 to generate an interrupt (every second). The Timer1 overflow interrupt wakes the device from sleep This causes the time registers (HRS, MIN, SECS) to be updated. If the SECS register contains an even value (SECS<0> = 0), the colon (":") is not displayed. This gives a visual indication for each second.

There are three keys for the setting of the clock. The SELECT_UNITS Key (S1) selects which units are to be modified (hours, minutes, off). The selected units are blanked for a second then flashed for one second. The INC Key (S2) increments the selected units. While incrementing, the selected units are displayed. After a key has not been depressed for more then one second, the selected units will begin to flash. The CLR_MIN Key (S3) clears the minutes and seconds. CLR_MIN is useful for exactly setting the time to the “top of the hour” as announced in radio broadcasts. After the INC or SELECT_UNITS keys are depressed, the user has ten seconds to depress the next key. After no key has been depressed for ten seconds, the unit returns to the clock mode.

To simplify the design time, a standard Hitachi LCD display module is used. In most applications requiring a LCD display, a custom LCD display is used. The LCD interface software would need to be modified to suit the specific LCD display driver being used.

Figure 1 is a block diagram of the design. RA<2:0> are the control signals to the LCD display, RB<3:0> is the 4-bit data bus, and RB<7:5> is the input switches. The OSC1 pin is connected to an RC network, which generates approximately a 4 MHz device frequency. The device frequency does not need to be stable, since the Timer1 module operates asynchronously. This allows the device's oscillaor to be configures for RC mode. This oscillator mode is the least expensive and has the quickest start-up time. Timer1 is where the accurate frequency is required. This crystal is connected to the T1OSI and T1OSO pins. A good choice for a crystal is a 32.786 KHz (watch) crystal. Table 1 is a list of the components and their part number.

FIGURE 1: CLOCK BLOCK DIAGRAM



Low-Power Real Time Clock

Relative to most microelectronics, the LCD's are slow devices. A good portion of the time spent in the Interrupt Service Routine, is talking to and updating the LCD module. To minimize power consumption, the device should be in the sleep mode as much as possible.

By using the conditional assemble, if a flag (called Debug) is true, the total time spent in the subroutine can be seen on the PORTD<0> pin (the high time). Measuring this time on an oscilloscope displayed a typical time of 800 uS that the device is awake. This 800 uS operation is out of the 1 second time that the device needs to service the interrupt (a TMR1 overflow).

The accuracy of a real time clock using Timer1 depends on the accuracy of the crystal being used. The more accurate the crystal, the higher the cost. So as always there is a cost / performance trade-off to be made. A crystal rated with an accuracy of 20 PPM (parts per million), could cause an error of about 1.7 seconds per day. For many applications, this should be adequate (said from someone who doesn't wear a watch).

The program presented here shows one method for a real time clock. Trade-offs between code size, current consumption and desired operation have been made. Some possible alternative implementations are:

1. When displaying the time, update only the characters that changed.
2. Turn off the display during sleep
3. LCD module data interface of 8-bits, not the 4-bit interface.

Alternative 1 can reduce the time awake, by keeping track of which characters need to be updated. The majority of the time it will be only the position which contains either the ":" or the " ". Next would be the ones place of the minutes, then the tens place of the minutes, etc. The display would only need to be completely updated 2 times every 24 hrs. This would reduce the amount of time talking with the LCD display at the cost of some program / data memory.

Depending on the requirements of the application and the characteristics of the display, alternative 2 could be implemented by turning the power off and on (at a given rate) to the display. This technique may lead to a lower system current consumption. Evaluation upon the desired display / display driver is recommended.

Alternative 3 uses the LCD module in an 8-bit mode, will reduce the size of the display routines (save about 20 words of program memory) at the cost of four additional I/O lines. For some applications this may be a good trade off to get the additional program memory space. The percentage of operating time saved is slight and should not give substantial power savings.

TABLE 1: LIST OF COMPONENTS†

Description	Part Number	Manufacturer	Quantity
LCD Module (2 x 20 Characteristics)	LM032L	Hitachi	1
Switches	EVQPADO4M	Panasonic	4
Microcontroller	PIC16C64 / 74	Microchip	1
32.768 KHz Crystal	NC26 / NC38	FOX	1
4 MHz Crystal	ECS-40-20-1	ECS	1

† Most components available from DigiKey.

CONCLUSION

The Timer1 module allows many applications to include a real time clock at minimal system cost. This time function can be useful in consumer applications (display time) as well as in industrial applications (data time stamp). The accuracy of the time is strictly dependent on the accuracy of the crystal. Table 2 shows the program resource requirements.

TABLE 2: PROGRAM RESOURCE REQUIREMENTS

Resource			Words / Bytes	Cycles
Program Memory	Initialization		61	61
	Clock Operation	Increment Time W.C.	106	35 + Display
		Key Input W.C.		35 + Display Time
Data Memory	Display‡		208	526†
	Variables		5	N.A.
	Scratch RAM		4	N.A.

† Dependent on LCD Module (re: BUSY_CHECK subroutine)

‡ Assumes worst case numbers and best case response from LCD module.

Author: Mark Palmer - Sr. Application Engineer
Logic Products Division

APPENDIX A: SOURCE CODE LISTING (CLOCK_01.LST)

```
MPASM 01.01 Released     CLOCK.ASM   5-13-1994  13:11:9               PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*******************************************************************************
                     0004 ;
                     0005 ; This program implements a real time clock using the TMR1 module of the
                     0006 ; PIC16Cxx family. A LCD display module is used to display (update) the
                     0007 ; time every second. Three keys are used to set the time.
                     0008 ;
                     0009 ;       Program = CLOCK.ASM
                     0010 ;       Revision Date:   5-15-94
                     0011 ;
                     0012 ;*******************************************************************************
                     0013 ;
                     0014 ;
                     0015 ; HARDWARE SETUP
                     0016 ;   LCD Control Lines
                     0017 ;       RA0 = E     (Enable)
                     0018 ;       RA1 = R_W   (Read/Write)
                     0019 ;       RA2 = RS    (Register Select)
                     0020 ;   LCD Data Lines
                     0021 ;       RB<3:0>
                     0022 ;   Switch Inputs
                     0023 ;       RB7 = Select Hour / Minute / Off
                     0024 ;       RB6 = Increment Hour / Minute
                     0025 ;       RB5 = Reset Minutes to 00
                     0026 ;
                     0027             INCLUDE <C74_reg.h>
                     0233
                     0027
                     0028             INCLUDE <CLOCK.h>
                     0028
                     0029 ;
0006                 0030 LCD_DATA        EQU     PORTB       ; The LCD data is on the lower 4-bits
0086                 0031 LCD_DATA_TRIS   EQU     TRISB       ; The TRIS register for the LCD data
0005                 0032 LCD_CNTL        EQU     PORTA       ; Three control lines
                     0033 ;
0000                 0034 PICMaster       EQU     FALSE        ; A Debugging Flag
0000                 0035 Debug           EQU     FALSE        ; A Debugging Flag
```

```
0001                    0036 Debug_PU       EQU     TRUE        ; A Debugging Flag
                        0037 ;
                        0038 ;
                        0039 ; Reset address. Determine type of RESET
                        0040 ;
                        0041         org    RESET_V             ; RESET vector location
0000 1683               0042 RESET          BSF    STATUS, RP0  ; Bank 1
0001 188E               0043                BTFSC  PCON, POR    ; Power-up reset?
0002 290C               0044                GOTO   START        ; YES
0003 295F               0045                GOTO   OTHER_RESET  ; NO, a WDT or MCLR reset
                        0046 ;
                        0047 ; This is the Periperal Interrupt routine. Need to determine the type
                        0048 ; of interrupt that occurred. The following interrupts are enabled:
                        0049 ;   1.  PORTB Change (RBIF)
                        0050 ;   2.  TMR1 Overflow Interrupt (T1IF)
                        0051 ;
                        0053         org    ISR_V               ; Interrupt vector location
                        0054 PER_INT_V
                        0055         if ( Debug )
                        0056             bsf    PORTD, 0        ; Set high, use to measure total
                        0057         endif                      ;    time in Int Service Routine
                        0058 ;
0004 1283               0059             BCF    STATUS, RP0     ; Bank 0
0005 180C               0060             BTFSC  PIR1, TMR1IF    ; Timer 1 overflowed?
0006 2843               0061             GOTO   T1_OVRFL        ; YES, Service the Timer1 Overflow Interrupt
0007 1C0B               0062             BTFSS  INTCON, RBIF    ; NO, Did PORTB change?
0008 28D0               0063             GOTO   ERROR1          ; NO, Error Condition - Unknown Interrupt
                        0064 ;
                        0065 PORTB_FLAG                        ; Are any of PORTB's inputs active?
0009 0806               0066             MOVF   PORTB, W       ;
000A 39E0               0067             ANDLW  0xE0           ; Keep only the 3 switch values
000B 00B5               0068 DEBOUNCE    MOVWF  TEMP           ;
000C 3002               0069             MOVLW  DB_HI_BYTE     ; This is the debounce delay
000D 08B3               0070             MOVF   MSD            ;
000E 01B4               0071             CLRF   LSD            ;
000F 0BB4               0072 KB_D_LP1    DECFSZ LSD            ;
0010 280F               0073             GOTO   KB_D_LP1       ;
0011 0BB3               0074             DECFSZ MSD            ;
0012 280F               0075             GOTO   KB_D_LP1       ;
0013 0806               0076 END_DELAY   MOVF   PORTB, W       ;
0014 39E0               0077             ANDLW  0xE0           ; Keep only the 3 switch values
0015 02B5               0078             SUBWF  TEMP, F        ;
0016 1D03               0079             BTFSS  STATUS, Z      ; Is the Zero bit set?
                        0080                                   ;   (switches were the same on 2 reads)
0017 280B               0081             GOTO   DEBOUNCE       ; NO, Try another read
0018 00B5               0082 KEY_MATCH   MOVWF  TEMP           ; YES, need to see which is depressed.
                        0083 ;
0019 3080               0084             MOVLW  0x80           ; Since doing key inputs, clear TMR1
```

```
001A 008F                       MOVWF   TMR1H                   ;   for 1 sec overflow.
001B 018E                       CLRF    TMR1L                   ;
001C 100C                       BCF     PIR1, TMR1IF            ; Clear Timer 1 Interrupt Flag
                                                                ;
001D 1FB5                       BTFSS   TEMP, HR_MIN_SW         ; Is the hour-min-off switch depressed?
001E 2826                       GOTO    SELECT_UNITS            ; YES, specify the units selected
001F 1F35                       BTFSS   TEMP, INC_SW            ; Is the inc switch depressed?
0020 282B                       GOTO    INC_UNIT                ; YES, Increment the selected Units
0021 1EB5                       BTFSS   TEMP, CLR_MIN_SW        ; Is the clear minute switch depressed?
0022 2835                       GOTO    CLR_MIN                 ; YES, clear the minutes.
                        ;
                        ; No key match occured, or finished with PortB interrupt and need to clear interrupt condition.
                        ;
0023 0886               CLR_RB          MOVF    PORTB, F        ; No RB<7:5> keys are depressed (rising edge Int.)
0024 100B                       BCF     INTCON, RBIF            ; Clear the PORTB mismatch condition
                                if ( Debug )                    ; Clear the PORTB Int Flag
0025 0009                       bcf     PORTD, 0                ; Set low, use to measure total
                                endif                           ;  time in Int Service Routine
                                RETFIE                          ; Return / Enable Global Interrupts
                        ;
0026 30FF               SELECT_UNITS
                                MOVLW   0xFF                    ;
0027 00C0                       MOVWF   WAIT_CNTR               ; WAIT_CNTR has LSb set after each SELECT UNIT key press.
0028 0AA0                       INCF    FLAG_REG, F             ; Increment the pointer to the MIN_UNIT:HR_UNIT
0029 1620                       BSF     FLAG_REG, KEY_INPUT     ;
002A 2875                       GOTO    DISPLAY                 ; Flash the Display of the selected unit
                        ;
002B 01C0               INC_UNIT
002C 1820                       CLRF    WAIT_CNTR               ; WAIT_CNTR is cleared to zero after each key press.
002D 285C                       BTFSC   FLAG_REG, HR_UNIT       ; Are the hour units selected?
002E 1CA0                       GOTO    INC_HRS                 ; YES, Increment the hour units
002F 2823                       BTFSS   FLAG_REG, MIN_UNIT      ; Are the minute units selected?
                                GOTO    CLR_RB                  ; NO, Not a valid key. Clear flags
                        ;
0030 0AB1                       INCF    MIN, F                  ; YES, Increment the minute units
0031 303C                       MOVLW   0x3C                    ; This is Decimal 60
0032 0231                       SUBWF   MIN, W                  ; MIN - 60 = ?
0033 1D03                       BTFSS   STATUS, Z               ; MIN = 60?
0034 2875                       GOTO    DISPLAY                 ; NO, display time
                                                                ; YES, MIN = 0 (use code from CLR_MIN)
0035 01B1               CLR_MIN         CLRF    MIN             ; MIN = 0
0036 3004                       MOVLW   0x04                    ; Clear the seconds
0037 00B2                       MOVWF   SECS                    ; Initial Second count = 4
0038 3080                       MOVLW   0x80                    ; Clear Timer 1, for 1 sec overflow
0039 008F                       MOVWF   TMR1H                   ;
003A 018E                       CLRF    TMR1L                   ;
003B 100C                       BCF     PIR1, TMR1IF            ; Clear the TMR1 overflow interrupt.
```

```
003C 01C0  0134          CLRF    WAIT_CNTR               ; WAIT_CNTR is cleared to zero after each key press.
003D 1AB5  0135          BTFSC   TEMP, CLR_MIN_SW        ; Is the clear minute switch depressed?
003E 2875  0136          GOTO    DISPLAY                 ; NO. Rollover from increment key
003F 10A0  0137          BCF     FLAG_REG, MIN_UNIT      ; YES, Clear ALL relevant flags
0040 1020  0138          BCF     FLAG_REG, HR_UNIT       ;
0041 1220  0139          BCF     FLAG_REG, KEY_INPUT     ;
0042 2875  0140          GOTO    DISPLAY                 ;
           0141 ;
           0143 ;
           0144 T1_OVRFL
0043 100C  0145          BCF     PIR1, TMR1IF            ; Clear Timer 1 Interrupt Flag
0044 1E20  0146          BTFSS   FLAG_REG, KEY_INPUT     ; Are we using the key inputs?
0045 284F  0147          GOTO    INC_TIME                ; NO, Need to Increment the time
0046 0AC0  0148          INCF    WAIT_CNTR, F            ; YES,
0047 300A  0149          MOVLW   0x0A                    ; 10 counts x 1 seconds
0048 0240  0150          SUBWF   WAIT_CNTR, W            ; Has the 10 Sec wait for key expired?
0049 1D03  0151          BTFSS   STATUS, Z               ; Is the result 0?
004A 2875  0152          GOTO    DISPLAY                 ; NO, Display value
004B 01C0  0153          CLRF    WAIT_CNTR               ; YES, Clear WAIT_CNTR
004C 1220  0154          BCF     FLAG_REG, KEY_INPUT     ;
004D 1020  0155          BCF     FLAG_REG, HR_UNIT       ;
004E 10A0  0156          BCF     FLAG_REG, MIN_UNIT      ;
           0157 ;
           0158 ;
004F 3080  0159 INC_TIME MOVLW   0x80                    ;
0050 008F  0160          MOVWF   TMR1H                   ; 1 Second Overflow
0051 0AB2  0161          INCF    SECS, F                 ;
0052 1F32  0162          BTFSS   SECS, 6                 ;
0053 2875  0163          GOTO    DISPLAY                 ;
0054 3004  0164          MOVLW   0x04                    ;
0055 00B2  0165          MOVWF   SECS                    ;
0056 0AB1  0166          INCF    MIN, F                  ;
0057 303C  0167          MOVLW   0x3C                    ; W = 60d
0058 0231  0168          SUBWF   MIN, W                  ;
0059 1D03  0169          BTFSS   STATUS, Z               ;
005A 2875  0170          GOTO    DISPLAY                 ;
005B 01B1  0171          CLRF    MIN                     ;
005C 0AB0  0172 INC_HRS  INCF    HRS, F                  ;
005D 300C  0173          MOVLW   0x0C                    ; It is now 12:00, Toggle AM / PM
005E 0230  0174          SUBWF   HRS, W                  ;
005F 1D03  0175          BTFSS   STATUS, Z               ;
0060 2867  0176          GOTO    CK_13                   ; Need to check if HRS = 13
0061 1FA0  0177          BTFSS   FLAG_REG, AM            ; Was it AM or PM
0062 2865  0178          GOTO    SET_AM                  ; Was PM, Needs to be AM
0063 13A0  0179          BCF     FLAG_REG, AM            ; It is PM
0064 2875  0180          GOTO    DISPLAY                 ;
0065 17A0  0181 SET_AM   BSF     FLAG_REG, AM            ; It is AM
0066 2875  0182          GOTO    DISPLAY                 ;
```

```
0183 ;
0184 CK_13       MOVLW   0x0D            ; Check if HRS = 13
0185             SUBWF   HRS, W          ;
0186             BTFSS   STATUS, Z       ;
0187             GOTO    DISPLAY         ;
0188             CLRF    HRS             ;
0189             INCF    HRS, F          ;
0190             GOTO    DISPLAY         ;
0191 ;
0193 INIT_DISPLAY
0194             MOVLW   DISP_ON         ; Display On, Cursor On
0195             CALL    SEND_CMD        ; Send This command to the Display Module
0196             MOVLW   CLR_DISP        ; Clear the Display
0197             CALL    SEND_CMD        ; Send This command to the Display Module
0198             MOVLW   ENTRY_INC       ; Set Entry Mode Inc., No shift
0199             CALL    SEND_CMD        ; Send This command to the Display Module
0200             RETURN
0201 ;
0202 DISPLAY
0203             MOVLW   DD_RAM_ADDR     ;
0204             CALL    SEND_CMD        ;
0205 ;
0206             BTFSC   FLAG_REG, KEY_INPUT ; Do we need to flash the selectected units?
0207             GOTO    FLASH_UNITS     ; YES, we need to flash selected units
0208             CALL    LOAD_HRS        ; NO, do a normal display
0209             CALL    LOAD_COLON      ;
0210             CALL    LOAD_MIN        ;
0211             GOTO    LOAD_AM         ;
0212 ;
0213 FLASH_UNITS
0214             CLRF    PCLATH          ; This clears PCLATH, This table in 1st
0215             MOVF    FLAG_REG, W     ;   256 bytes of program memory
0216             ANDLW   0x03            ; only HR_UNIT and MIN_UNIT bit can be non-zero
0217 UNIT_TBL
0218             ADDWF   PCL             ; HR_UNIT:MIN_UNIT
0219             GOTO    NO_UNITS        ;    0 0  - Display everything.
0220             GOTO    HR_UNITS        ;    0 1  - Flash the hour units
0221             GOTO    MIN_UNITS       ;    1 0  - Flash the minute units
0222 UNIT_TBL_END
0223             MOVLW   0xFC            ;    1 1  - Need to clear FLAG_REG
0224             ANDWF   FLAG_REG, F     ;                <HR_UNIT:MIN_UNIT>
0225             GOTO    NO_UNITS        ;    0 0  - Display everything.
0226 ;
0227     if ( (UNIT_TBL & 0x0FF) >= (UNIT_TBL_END & 0x0FF) )
0228       MESSG  "Warning: Table UNIT_TBL crosses page boundry in computed jump"
0229     endif
0230 ;
0231 ;
```

```
                        0232 HR_UNITS
0087 1C40               0233            BTFSS   WAIT_CNTR, 0    ; If WAIT_CNTR is odd,
                        0234                                    ;   hour digits are displayed as blank
0088 288D               0235            GOTO    SKIP_BLK_HRS    ;
0089 3020               0236            MOVLW   ' '             ;
008A 20D4               0237            CALL    SEND_CHAR       ;
008B 3020               0238            MOVLW   ' '             ;
008C 20D4               0239            CALL    SEND_CHAR       ;
                        0240 SKIP_BLK_HRS
008D 1C40               0241            BTFSS   WAIT_CNTR, 0    ; WAIT_CNTR was even, display hour digits
008E 20A4               0242            CALL    LOAD_HRS        ;
                        0243 ;
008F 303A               0244            MOVLW   ':'             ; : always on, display all other character
0090 20D4               0245            CALL    SEND_CHAR       ;
0091 20B2               0246            CALL    LOAD_MIN        ;
0092 28BB               0247            GOTO    LOAD_AM         ;
                        0248 ;
                        0250 MIN_UNITS
0093 20A4               0251            CALL    LOAD_HRS        ; Display hours
0094 303A               0252            MOVLW   ':'             ; : always on
0095 20D4               0253            CALL    SEND_CHAR       ;
0096 1C40               0254            BTFSS   WAIT_CNTR, 0    ; If WAIT_CNTR is odd,
                        0255                                    ;   minute digits are displayed as blank
0097 289C               0256            GOTO    SKIP_BLK_MIN    ;
0098 3020               0257            MOVLW   ' '             ;
0099 20D4               0258            CALL    SEND_CHAR       ;
009A 3020               0259            MOVLW   ' '             ;
009B 20D4               0260            CALL    SEND_CHAR       ;
                        0261 SKIP_BLK_MIN
009C 1C40               0262            BTFSS   WAIT_CNTR, 0    ; WAIT_CNTR was even, display minute digits
009D 20B2               0263            CALL    LOAD_MIN        ;
009E 28BB               0264            GOTO    LOAD_AM         ;
                        0265 ;
                        0266 NO_UNITS
009F 20A4               0267            CALL    LOAD_HRS        ; Display all character
00A0 303A               0268            MOVLW   ':'             ;
00A1 20D4               0269            CALL    SEND_CHAR       ;
00A2 20B2               0270            CALL    LOAD_MIN        ;
00A3 28BB               0271            GOTO    LOAD_AM         ;
                        0272 ;
                        0273 LOAD_HRS
00A4 0830               0274            MOVF    HRS, W          ; Load the Wreg with the value
00A5 20C7               0275            CALL    BIN_2_BCD       ;   to convert to BCD
00A6 0833               0276            MOVF    MSD, W          ; Load the MSD value into the Wreg
00A7 2400               0277            CALL    NUM_TABLE       ; Get the ASCII code
00A8 20D4               0278            CALL    SEND_CHAR       ; Send this Character to the Display
                        0279 ;
00A9 0834               0280            MOVF    LSD, W          ; Load the LSD value into the Wreg
```

```
00AA 2400                       CALL    NUM_TABLE       ; Get the ASCII code
00AB 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
00AC 0008                       RETURN
                        ;
00AD 3020               LOAD_COLON  MOVLW   ' '          ; ASCII value for a Blank space
00AE 1832                       BTFSC   SECS, 0         ; Is it an EVEN or ODD second
00AF 3E1A                       ADDLW   ':' - ' '       ; Is ODD, Second colon is ON.
                                                        ;   Add delta offset of ASCII Characters
00B0 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
00B1 0008                       RETURN
                        ;
00B2 0831               LOAD_MIN    MOVF    MIN, W      ; Load the Wreg with the value
00B3 20C7                       CALL    BIN_2_BCD       ;   to convert to BCD
00B4 0833                       MOVF    MSD, W          ; Load the MSD value into the Wreg
00B5 2400                       CALL    NUM_TABLE       ; Get the ASCII code
00B6 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
                        ;
00B7 0834                       MOVF    LSD, W          ; Load the LSD value into the Wreg
00B8 2400                       CALL    NUM_TABLE       ; Get the ASCII code
00B9 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
00BA 0008                       RETURN
                        ;
00BB 3020               LOAD_AM     MOVLW   ' '         ; ASCII value for a Blank space
00BC 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
00BD 3041                       MOVLW   'A'             ; ASCII value for a Blank space
00BE 1FA0                       BTFSS   FLAG_REG, AM    ; Is it AM or PM
00BF 3E0F                       ADDLW   'P' - 'A'       ; Is PM, Add delta offset of ASCII Characters
00C0 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
00C1 304D                       MOVLW   'M'
00C2 20D4                       CALL    SEND_CHAR       ; Send this Character to the Display
                        ;
00C3 1683                       BSF     STATUS, RP0     ; Bank 1
00C4 1381                       BCF     OPTION_R, RBPU  ; Turn on PORTB Pull-up
00C5 1283                       BCF     STATUS, RP0     ; Bank 0
00C6 2823                       GOTO    CLR_RB          ; You've displayed the time, Clear RBIF
                        ;
                        ;
                        ;*****************************************************************
                        ; The BIN_2_BCD routine converts the binary number, in the W register, to a
                        ; binary coded decimal (BCD) munber. This BCD number is stored MSD:LSD. This
                        ; routine is used by the DISPLAY subroutine, to convert the time values.
                        ;*****************************************************************
                        ;
00C7 01B3               BIN_2_BCD   CLRF    MSD             ; This value contain the 10's digit value
00C8 00B4                       MOVWF   LSD             ; This value contain the 1's digit value
00C9 300A               TENS_SUB    MOVLW   .10         ; A decimal 10
00CA 0234                       SUBWF   LSD, W          ;
00CB 1C03                       BTFSS   STATUS, C       ; Did this subtract cause a Negative Result?
```

```
00CC 3400           RETLW   0               ; YES, Return from this Routine
00CD 00B4           MOVWF   LSD             ; No, move the result into LSD
00CE 0AB3           INCF    MSD, F          ; Increment the most significat digit
00CF 28C9           GOTO    TENS_SUB        ;
                    ;
                    ;
                    ; Should NEVER get here
                    ;
00D0 1283           ERROR1  BCF     STATUS, RP0     ; Bank 0
                    ;
                        if ( Debug )
                            bsf     PORTD, 1
                            bcf     PORTD, 1
                        else
00D1 1407               BSF     PORTC, 0
00D2 1007               BCF     PORTC, 0
                        endif
00D3 28D0               GOTO    ERROR1
                    ;
                    ;
                    ;*****************************************************************
                    ;* SendChar - Sends character to LCD                             *
                    ;* This routine splits the character into the upper and lower    *
                    ;* nibbles and sends them to the LCD, upper nibble first.        *
                    ;* The data is transmitted on the PORT<3:0> pins                 *
                    ;*****************************************************************
                    ;
00D4 00B6           SEND_CHAR
00D4 00B6               MOVWF   CHAR            ; Character to be sent is in W
00D5 20F2               CALL    BUSY_CHECK      ; Wait for LCD to be ready
00D6 0E36               SWAPF   CHAR, W
00D7 390F               ANDLW   0x0F            ; Get upper nibble
00D8 0086               MOVWF   LCD_DATA        ; Send data to LCD
00D9 1085               BCF     LCD_CNTL, R_W   ; Set LCD to read
00DA 1505               BSF     LCD_CNTL, RS    ; Set LCD to data mode
00DB 1405               BSF     LCD_CNTL, E     ; toggle E for LCD
00DC 1005               BCF     LCD_CNTL, E
00DD 0836               MOVF    CHAR, W
00DE 390F               ANDLW   0x0F            ; Get lower nibble
00DF 0086               MOVWF   LCD_DATA        ; Send data to LCD
00E0 1405               BSF     LCD_CNTL, E     ; toggle E for LCD
00E1 1005               BCF     LCD_CNTL, E
00E2 0008               RETURN
                    ;
                    ;*****************************************************************
```

```
0377 ;* SendCmd - Sends command to LCD                                      *
0378 ;* This routine splits the command into the upper and lower           *
0379 ;* nibbles and sends them to the LCD, upper nibble first.             *
0380 ;* The data is transmitted on the PORT<3:0> pins                      *
0381 ;*****************************************************************************
0382
0383 SEND_CMD
0384                 MOVWF   CHAR            ; Character to be sent is in W
0385                 CALL    BUSY_CHECK      ; Wait for LCD to be ready
0386                 SWAPF   CHAR, W
0387                 ANDLW   0x0F            ; Get upper nibble
0388                 MOVWF   LCD_DATA        ; Send data to LCD
0389                 BCF     LCD_CNTL, R_W   ; Set LCD to read
0390                 BCF     LCD_CNTL, RS    ; Set LCD to command mode
0391                 BSF     LCD_CNTL, E     ; toggle E for LCD
0392                 BCF     LCD_CNTL, E
0393                 MOVF    CHAR, W
0394                 ANDLW   0x0F            ; Get lower nibble
0395                 MOVWF   LCD_DATA        ; Send data to LCD
0396                 BSF     LCD_CNTL, E     ; toggle E for LCD
0397                 BCF     LCD_CNTL, E
0398                 RETURN
0400 ;*****************************************************************************
0401 ;* This routine checks the busy flag, returns when not busy          *
0402 ;*  Affects:                                                         *
0403 ;*     TEMP - Returned with busy/address                             *
0404 ;*****************************************************************************
0405
0406 BUSY_CHECK
0407 ;
0408         if ( Debug )
0409                 bsf     PORTD, 3
0410                 bcf     PORTD, 3
0411         endif
0412                 CLRF    LCD_DATA        ;** Have PORTB<3:0> output low
0413                 BSF     STATUS, RP0     ; Bank 1
0414                 BSF     OPTION_R, RBPU  ; Turn off PORTB Pull-up
0415                 MOVLW   0xFF            ; Set PortB for input
0416                 MOVWF   LCD_DATA_TRIS
0417                 BCF     STATUS, RP0     ; Bank 0
0418                 BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0419                 BSF     LCD_CNTL, R_W   ; Setup to read busy flag
0420                 BSF     LCD_CNTL, E     ; Set E high
0421                 BCF     LCD_CNTL, E     ; Set E low
0422                 SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address

00E3 00B6
00E4 20F2
00E5 0E36
00E6 390F
00E7 0086
00E8 1085
00E9 1105
00EA 1405
00EB 1005
00EC 0836
00ED 390F
00EE 0086
00EF 1405
00F0 1005
00F1 0008

00F2 0186
00F3 1683
00F4 1781
00F5 30FF
00F6 0086
00F7 1283
00F8 1105
00F9 1485
00FA 1405
00FB 1005
00FC 0E06
```

```
00FD 39F0          0423          ANDLW   0xF0            ; Mask out lower nibble
00FE 00B5          0424          MOVWF   TEMP            ;
00FF 1405          0425          BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
0100 1005          0426          BCF     LCD_CNTL, E     ;
0101 0806          0427          MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
0102 390F          0428          ANDLW   0x0F            ; Mask out upper nibble
0103 04B5          0429          IORWF   TEMP, F         ; Combine nibbles
0104 1BB5          0430          BTFSC   TEMP, 7         ; Check busy flag, high = busy
0105 28F2          0431          GOTO    BUSY_CHECK      ; If busy, check again
0106 1085          0432          BCF     LCD_CNTL, R_W   ;
0107 1683          0433          BSF     STATUS, RP0     ; Bank 1
0108 30F0          0434          MOVLW   0xF0            ;
0109 0086          0435          MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
010A 1283          0436          BCF     STATUS, RP0     ; Bank 0
010B 0008          0437          RETURN
                   0438 ;
                   0440 ;
                   0441 ;*****************************************************************
                   0442 ;*****     Start program here, Power-On Reset occurred.
                   0443 ;*****************************************************************
                   0444 ;
                   0445 START                         ; POWER_ON Reset (Beginning of program)
010C 1283          0446          BCF     STATUS, RP0     ; Bank 0
010D 300C          0447          MOVLW   0x0C            ; Decimal 12
010E 00B0          0448          MOVWF   HRS             ; HOURS = 12
010F 01B1          0449          CLRF    MIN             ; MIN   = 00
0110 3000          0450          MOVLW   0x00            ;
0111 00A0          0451          MOVWF   FLAG_REG        ; PM light is on
0112 3004          0452          MOVLW   0x04            ; Initial value of seconds (64d - 60d)
0113 00B2          0453          MOVWF   SECS            ; This allows a simple bit test to see if 60
                   0454                                  ;  secs has elapsed.
0114 3080          0455          MOVLW   0x80            ; TIM1H:TMR1L = 0x8000 gives 1 second
0115 008F          0456          MOVWF   TMR1H           ;  overflow, at 32 KHz.
0116 018E          0457          CLRF    TMR1L           ;
                   0458 ;
                   0459 MCLR_RESET                    ; A Master Clear Reset
0117 1283          0460          BCF     STATUS, RP0     ; Bank 0
0118 0183          0461          CLRF    STATUS          ; Do initialization (Bank 0)
0119 018B          0462          CLRF    INTCON
011A 018C          0463          CLRF    PIR1
011B 1683          0464          BSF     STATUS, RP0     ; Bank 1
011C 3000          0465          MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
011D 0081          0466          MOVWF   OPTION_R        ;
011E 018C          0467          CLRF    PIE1            ; Disable all peripheral interrupts
011F 30FF          0468          MOVLW   0xFF            ;
0120 009F          0469          MOVWF   ADCON1          ; Port A is Digital.
                   0470 ;
                   0471 ;
```

```
0121 1283           BCF     STATUS, RP0     ; Bank 0
0122 0185           CLRF    PORTA           ; ALL PORT output should output Low.
0123 0186           CLRF    PORTB
0124 0187           CLRF    PORTC
0125 0188           CLRF    PORTD
0126 0189           CLRF    PORTE
0127 1010           BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                    ;
0128 1683           BSF     STATUS, RP0     ; Select Bank 1
0129 0185           CLRF    TRISA           ; RA5 -  0 outputs
012A 30F0           MOVLW   0xF0            ;
012B 0086           MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
012C 0187           CLRF    TRISC           ; RC Port are outputs
012D 1407           BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
012E 0188           CLRF    TRISD           ; RD Port are outputs
012F 0189           CLRF    TRISE           ; RE Port are outputs
0130 140C           BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0131 1381           BCF     OPTION_R, RBPU  ; Enable PORTB pull-ups
0132 1283           BCF     STATUS, RP0     ; Select Bank 0
0133 0886           MOVF    PORTB, F        ; Need to clear 1st RBIF, due to
0134 100B           BCF     INTCON, RBIF    ;    set up of PORTB
                    ;
                    ;
                    ; Initilize the LCD Disply Module
                    ;
0135 0185           CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                    ;
                    DISPLAY_INIT
0136 3002           MOVLW   0x02            ; Command for 4-bit interface
0137 0086           MOVWF   LCD_DATA        ;
0138 1405           BSF     LCD_CNTL, E     ;
0139 1005           BCF     LCD_CNTL, E     ;
                    ;
                    ; This routine takes the calculated times that the delay loop needs to
                    ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                    ; frequency of operation. These uses registers before they are needed to
                    ; store the time.
                    ;
013A 3006           LCD_DELAY   MOVLW   LCD_INIT_DELAY  ;
013B 00B3                       MOVWF   MSD             ; Use MSD and LSD Registers to Initilize LCD
013C 01B4                       CLRF    LSD             ;
013D 0BB4           LOOP2       DECFSZ  LSD             ; Delay time = MSD * ((3 * 256) + 3) * Tcy
013E 293D                       GOTO    LOOP2           ;
013F 0BB3                       DECFSZ  MSD             ;
                    END_LCD_DELAY
0140 293D                       GOTO    LOOP2           ;
                    ;
                    ; Command sequence for 2 lines of 5x7 characters
```

```
0141 3002       0521 ;
                0522 CMD_SEQ     MOVLW   0X02
0142 0086       0523             MOVWF   LCD_DATA
0143 1405       0524             BSF     LCD_CNTL, E     ;
0144 1005       0525             BCF     LCD_CNTL, E     ;
0145 3008       0526             MOVLW   0x08            ;
0146 0086       0527             MOVWF   LCD_DATA        ;
0147 1405       0528             BSF     LCD_CNTL, E     ;
0148 1005       0529             BCF     LCD_CNTL, E     ;
                0530 ;
                0531 ; Busy Flag should be valid after this point
                0532 ;
0149 300C       0533             MOVLW   DISP_ON         ;
014A 20E3       0534             CALL    SEND_CMD        ;
014B 3001       0535             MOVLW   CLR_DISP        ;
014C 20E3       0536             CALL    SEND_CMD        ;
014D 3006       0537             MOVLW   ENTRY_INC       ;
014E 20E3       0538             CALL    SEND_CMD        ;
014F 3080       0539             MOVLW   DD_RAM_ADDR     ;
0150 20E3       0540             CALL    SEND_CMD        ;
                0541 ;
                0543 ;
                0544 ; Initialize the Special Function Registers (SFR) interrupts
                0545 ;
0151 018C       0546             CLRF    PIR1            ;
0152 300E       0547             MOVLW   0x0E
0153 0090       0548             MOVWF   T1CON           ; RC1 is overridden by TCKO
0154 170B       0549             BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0155 158B       0550             BSF     INTCON, RBIE    ; Disable PORTB<7:4> Change Interrupts
0156 178B       0551             BSF     INTCON, GIE     ; Enable all Interrupts
                0552 ;
0157 206E       0553             CALL    INIT_DISPLAY    ;
0158 2075       0554             CALL    DISPLAY         ;
                0555 ;
0159 300E       0556             MOVLW   0x0E
015A 0090       0557             MOVWF   T1CON           ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
015B 1410       0558             BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
                0559 ;
                0560         if ( PICMaster )
                0561 lzz         goto    lzz             ; Loop waiting for interrupts (for use with PICMASTER)
                0562         else
                0563 ;
015C 0063       0564 SLEEP_LP    SLEEP                   ; Wait for Change on PORTB interrupt. or TMR1 timeout
015D 0000       0565             NOP                     ;
015E 295C       0566             GOTO    SLEEP_LP        ;
                0567 ;
                0568         endif
                0569 ;
```

```
0570 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
0571 ;
0572 OTHER_RESET   BTFSS  STATUS, TO     ; WDT Time-out?
0573 WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
0574               if ( Debug_PU )
0575                 goto   START        ; MCLR reset, Goto START
0576               else
0577                 GOTO   MCLR_RESET   ; MCLR reset, Goto MCLR_RESET
0578               endif
0579 ;
0580               if (Debug )
0581 END_START       NOP                ; END lable for debug
0582               endif
0583 ;
0585 ;
0586               org    TABLE_ADDR
0587 ;
0588 NUM_TABLE         MOVWF   TEMP                ; Store value to TEMP register
0589                   MOVLW   HIGH (TABLE_ADDR)   ; Ensure that the PCLATH high has the
0590                   MOVWF   PCLATH              ;   correct value
0591                   MOVF    TEMP, W             ; Value into table
0592                   ANDLW   0x0F                ; Mask to 4-bits (00 - 0Fh)
0593 NUM_TBL           ADDWF   PCL, F              ; Determine Offset into table
0594                   RETLW   '0'                 ; ASCII value of "0" in W register
0595                   RETLW   '1'                 ; ASCII value of "1" in W register
0596                   RETLW   '2'                 ; ASCII value of "2" in W register
0597                   RETLW   '3'                 ; ASCII value of "3" in W register
0598                   RETLW   '4'                 ; ASCII value of "4" in W register
0599                   RETLW   '5'                 ; ASCII value of "5" in W register
0600                   RETLW   '6'                 ; ASCII value of "6" in W register
0601                   RETLW   '7'                 ; ASCII value of "7" in W register
0602                   RETLW   '8'                 ; ASCII value of "8" in W register
0603                   RETLW   '9'                 ; ASCII value of "9" in W register
0604                                               ; Any enter after is in error (Display an E)
0605                   RETLW   'E'                 ; ASCII value of "E" in W register
0606                   RETLW   'E'                 ; ASCII value of "E" in W register
0607                   RETLW   'E'                 ; ASCII value of "E" in W register
0608                   RETLW   'E'                 ; ASCII value of "E" in W register
0609                   RETLW   'E'                 ; ASCII value of "E" in W register
0610 NUM_TBL_END       RETLW   'E'                 ; ASCII value of "E" in W register
0611 ;
0612               if ( (NUM_TBL & 0x0FF) >= (NUM_TBL_END & 0x0FF) )
0613                 MESSG  "Warning: Table NUM_TBL crosses page boundry in computed jump"
0614               endif
0615 ;
0616 ;
0617               org    PMEM_END             ; End of Program Memory
0618                      GOTO    ERROR1       ;  If you get here your program was lost
```

```
0619
0620     end
0621
0622

MPASM 01.01 Released     CLOCK.ASM   5-13-1994  13:11:9                PAGE 15

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XX______ ________

0400 : XXXXXXXXXXXXXXXX XXXXX_____ ________ ________
0440 : ________ ________ ________ ________

0780 : ________ ________ ________ ________
07C0 : ________ ________ ________ _______X

All other memory blocks unused.

Errors   :     0
Warnings :    16
```

NOTE: Special Function Register data memory locations in Bank 1, are specified by their true address in the file C74_REG.H. The use of the MPASM assembler will generate a warning message, when these lables are used with direct addressing.

APPENDIX B: CLOCK_01.H INCLUDE FILE

```
       nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;       PROGRAM:       CLOCK_01.H
;       Revision:      5-04-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq               EQU     D'4000000'     ; Device Frequency is 4 MHz
DB_HI_BYTE             EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY         EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR             EQU     40             ; RAM Location
OUTER_CNTR             EQU     41             ; RAM Location
;
T1OSO                  EQU     0              ; The RC0 / T1OSO / T1CKI
;
RESET_V                EQU     0x0000         ; Address of RESET Vector
ISR_V                  EQU     0x0004         ; Address of Interrupt Vector
PMEM_END               EQU     0x07FF         ; Last address in Program Memory
TABLE_ADDR             EQU     0x0400         ; Address where to start Tables
;
HR_MIN_SW              EQU     0x7            ; The switch to select the units
INC_SW                 EQU     0x6            ; The switch to increment the selected units
CLR_MIN_SW             EQU     0x5            ; The switch to clear the minutes and seconds
;
FLAG_REG               EQU     0x020          ; Register which contains flag bits
;
;  +------+---+---+-----------+---+---+---------+--------+
;  |  AM  | - | - | KEY_INPUT | - | - | MIN_UNIT | HR_UNIT |
;  +------+---+---+-----------+---+---+---------+--------+
;
AM                     EQU     0x07           ; Flag to specify if AM or PM
;
KEY_INPUT              EQU     0x04           ; Flag to specify if doing key inputs
;
MIN_UNIT               EQU     0x01           ; Flags to specify which units to operate on
HR_UNIT                EQU     0x00           ;  (HRS, MIN, or none)
;
HRS                    EQU     0x030          ; Holds counter value for HOURS
MIN                    EQU     0x031          ; Holds counter value for MINUTES
SECS                   EQU     0x032          ; Holds counter value for SECONDS
MSD                    EQU     0x033          ; Temp. register, Holds Most Significant
                                              ;   Digit of BIN to BCD conversion
LSD                    EQU     0x034          ; Temporary register, Holds Least Significant
                                              ;   Digit of BIN to BCD conversion
TEMP                   EQU     0x035          ; Temporary register
CHAR                   EQU     0x036          ; Temporary register,
                                              ;  Holds value to send to LCD module.
;
WAIT_CNTR              EQU     0x040          ; Counter that holds wait time for key inputs
;
; LCD Display Commands and Control Signal names.
;
E                      EQU     0              ; LCD Enable control line
R_W                    EQU     1              ; LCD Read/Write control line
RS                     EQU     2              ; LCD Register Select control line
;
; LCD Module commands
;
DISP_ON                EQU     0x00C          ; Display on
DISP_ON_C              EQU     0x00E          ; Display on, Cursor on
DISP_ON_B              EQU     0x00F          ; Display on, Cursor on, Blink cursor
```

```
DISP_OFF              EQU     0x008          ; Display off
CLR_DISP              EQU     0x001          ; Clear the Display
ENTRY_INC             EQU     0x006          ;
ENTRY_INC_S           EQU     0x007          ;
ENTRY_DEC             EQU     0x004          ;
ENTRY_DEC_S           EQU     0x005          ;
DD_RAM_ADDR           EQU     0x080          ; Least Significant 7-bit are for address
DD_RAM_UL             EQU     0x080          ; Upper Left coner of the Display
;
       list
```

APPENDIX C: C74_REG.H INCLUDE FILE

```
    nolist
;
;   File =   C64_reg.h
;   Rev. History:     08-04-93 by MP
;                     10-18-93 by MP to make Page ok
;                     11-15-93 by MP to have correct pages for SFR
;
; EQUates for Special Function Registers
;
;
INDF                  EQU           00
RTCC                  EQU           01
OPTION_R              EQU           81
PCL                   EQU           02
STATUS                EQU           03
FSR                   EQU           04
PORTA                 EQU           05
TRISA                 EQU           85
PORTB                 EQU           06
TRISB                 EQU           86
PORTC                 EQU           07
TRISC                 EQU           87
PORTD                 EQU           08
TRISD                 EQU           88
PORTE                 EQU           09
TRISE                 EQU           89
PCLATH                EQU           0A
INTCON                EQU           0B
PIR1                  EQU           0C
PIE1                  EQU           8C
TMR1L                 EQU           0E
PCON                  EQU           8E
TMR1H                 EQU           0F
T1CON                 EQU           10
TMR2                  EQU           11
T2CON                 EQU           12
PR2                   EQU           92
SSPBUF                EQU           13
SSPADD                EQU           93
SSPCON                EQU           14
SSPSTAT               EQU           94
CCPR1L                EQU           15
CCPR1H                EQU           16
CCP1CON               EQU           17
RCSTA                 EQU           18
TXSTA                 EQU           98
TXREG                 EQU           19
SPBRG                 EQU           99
RCREG                 EQU           1A
CCPR2L                EQU           1B
CCPR2H                EQU           1C
CCP2CON               EQU           1D
ADRES                 EQU           1E
ADCON0                EQU           1F
ADCON1                EQU           9F

;
;*******************************************
;************   Bit Deffinitions   ************
;*******************************************
;
; STATUS register (Address 03/83)
;
IRP                   EQU           7
RP1                   EQU           6
RP0                   EQU           5
```

```
TO                      EQU             4
PD                      EQU             3
Z                       EQU             2
DC                      EQU             1
C                       EQU             0
;
; INTCON register (Address 0B/8B)
;
GIE                     EQU             7
PEIE                    EQU             6
RTIE                    EQU             5
INTE                    EQU             4
RBIE                    EQU             3
RTIF                    EQU             2
INTF                    EQU             1
RBIF                    EQU             0
;
; PIR1 register (Address 0C)
;
PSPIF                   EQU             7
SSPIF                   EQU             3
CCP1IF                  EQU             2
TMR2IF                  EQU             1
TMR1IF                  EQU             0
;
; PIE1 register (Address 8C)
;
PSPIE                   EQU             7
SSPIE                   EQU             3
CCP1IE                  EQU             2
TMR2IE                  EQU             1
TMR1IE                  EQU             0
;
; OPTION register (Address 81)
;
RBPU                    EQU             7
INTEDG                  EQU             6
RTS                     EQU             5
RTE                     EQU             4
PSA                     EQU             3
PS2                     EQU             2
PS1                     EQU             1
PS0                     EQU             0
;
; PCON register (Address 8E)
;
POR                     EQU             1
;
; TRISE register (Address 89)
;
IBF                     EQU             7
OBF                     EQU             6
IBOV                    EQU             5
PSPMODE                 EQU             4
TRISE2                  EQU             2
TRISE1                  EQU             1
TRISE0                  EQU             0

;
; T1CON register (Address 10)
;
T1CKPS1                 EQU             5
T1CKPS0                 EQU             4
T1OSCEN                 EQU             3
T1INSYNC                EQU             2
TMR1CS                  EQU             1
TMR1ON                  EQU             0
;
```

```
; T2CON register (Address 12)
;
TOUTPS3                 EQU             6
TOUTPS2                 EQU             5
TOUTPS1                 EQU             4
TOUTPS0                 EQU             3
TMR2ON                  EQU             2
T2CKPS1                 EQU             1
T2CKPS0                 EQU             0
;
; SSPCON register (Address 14)
;
WCOL                    EQU             7
SSPOV                   EQU             6
SSPEN                   EQU             5
CKP                     EQU             4
SSPM3                   EQU             3
SSPM2                   EQU             2
SSPM1                   EQU             1
SSPM0                   EQU             0
;
; SSPSTAT register (Address 94)
;
DA                      EQU             5
P                       EQU             4
S                       EQU             3
RW                      EQU             2
UA                      EQU             1
BF                      EQU             0
;
; CCP1CON register (Address 17)
;
CCP1X                   EQU             5
CCP1Y                   EQU             4
CCP1M3                  EQU             3
CCP1M2                  EQU             2
CCP1M1                  EQU             1
CCP1M0                  EQU             0
;
; RCSTA register (Address 18)
;
SPEN                    EQU             7
RC89                    EQU             6
SREN                    EQU             5
CREN                    EQU             4
FERR                    EQU             2
OERR                    EQU             1
RCD8                    EQU             0
;
; TXSTA register (Address 98)
;
CSRC                    EQU             7
TX89                    EQU             6
TXEN                    EQU             5
SYNC                    EQU             4
BRGH                    EQU             2
TRMT                    EQU             1
TXD8                    EQU             0


;
; CCP2CON register (Address 1D)
;
CCP2X                   EQU             5
CCP2Y                   EQU             4
CCP2M3                  EQU             3
CCP2M2                  EQU             2
CCP2M1                  EQU             1
CCP2M0                  EQU             0
```

```
;
; ADCON0 register (Address 1F)
;
ADCS1                   EQU             7
ADCS0                   EQU             6
CHS2                    EQU             5
CHS1                    EQU             4
CHS0                    EQU             3
GO                              EQU             2
DONE                    EQU             2
ADON                    EQU             0
;
; ADCON1 register (Address 9F)
;
PCFG2                   EQU             2
PCFG1                   EQU             1
PCFG0                   EQU             0
;
;*********************************************
;**** Bits for destination control
;****    W = W register is destination
;****    F = File register is destination
;*********************************************
;
W                       EQU             0
F                       EQU             1
;
FALSE                   EQU             0
TRUE                    EQU             1

    list
```

NOTES:

AN546

Using the Analog to Digital Converter

INTRODUCTION

This application note is intended for PIC16C7X users with various degrees of familiarity with analog system design. The various sections discuss the following topics:

- Commonly used A/D terminology
- How to configure and use the PIC16C71 A/D
- Various ways to generate external reference voltage (V_{REF})
- Configuring RA0-RA3 pins

COMMONLY USED A/D TERMINOLOGY

The Ideal Transfer Function

In an A/D converter, an analog voltage is mapped into an N-bit digital value. This mapping function is defined as the transfer function. An ideal transfer is one in which there are no errors or non-linearity. It describes the "ideal" or intended behavior of the A/D. Figure 1 shows the ideal transfer function for the PIC16C7X A/D. Note that the digital output value is 00h for analog input voltage range of 0 to 1LSB. In some converters, the first transition point is at 0.5LSB and not at 1LSB as shown in Figure 2. Either way, knowing the transfer function the user can appropriately interpret the data.

Transition Point

It is the analog input voltage at which the digital output switches from one code to the next. The transition point is typically not a single threshold, rather a small region of uncertainty (see Figure 3) The transition point is therefore defined as the statistical average of many conversions. Stated differently, it is the voltage input at which the uncertainty of the conversion is 50%.

Code Width

It is the distance (voltage differential) between two transition points. Ideally the Code Width should be 1LSB. See Figure 1.

FIGURE 1 - PIC16C7X IDEAL TRANSFER FUNCTION

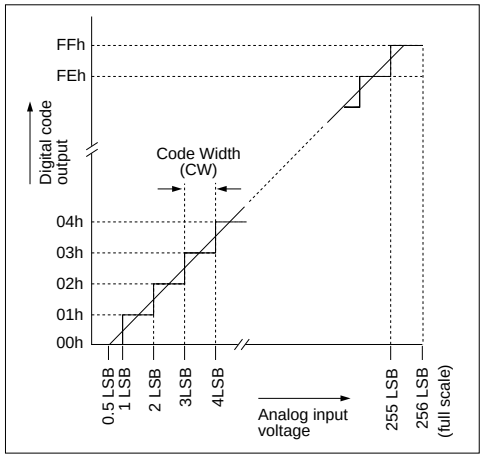
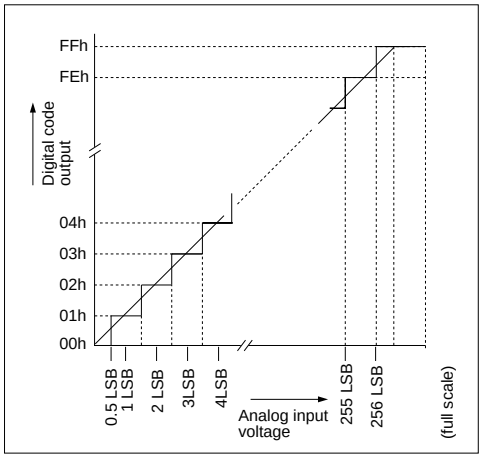


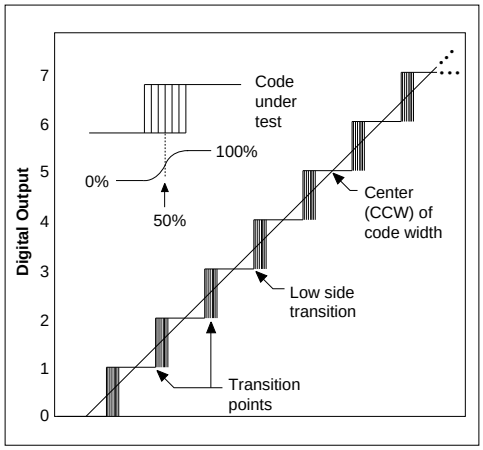
FIGURE 2 - ALTERNATE TRANSFER FUNCTION



Center of Code Width

It is the midpoint between two transition points. See Figure 3.

FIGURE 3 - TRANSITION POINTS



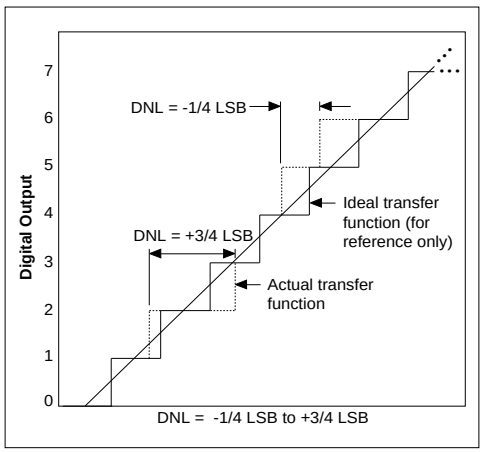
Differential Non-Linearity (DNL)

It is the deviation in code-width from 1LSB (Figure 7). The difference is calculated for each and every transition. The largest difference is reported as DNL.

It is important to note that the DNL is measured after the transfer function is normalized to match offset error and gain error.

Note that the DNL cannot be any less than -1LSB. In the other direction, DNL can be >1LSB.

FIGURE 7 - DIFFERENTIAL NON-LINEARITY



Absolute Error

The maximum deviation between any transition point from the corresponding ideal transfer function is defined as the absolute error. This is how it is measured and reported in the PIC16C7X (Figure 8). The notable difference between absolute error and INL is that the measured data is not normalized for full scale and offset errors.

It is probably the first parameter the user will look at to evaluate an A/D. Sometimes absolute error is reported as the sum of offset, full-scale and integral non linearity errors.

Total Unadjusted Error

It is the same as absolute error. Again, sometimes it is reported as the sum of offset, full-scale and integral non-linearity errors.

No Missing Code

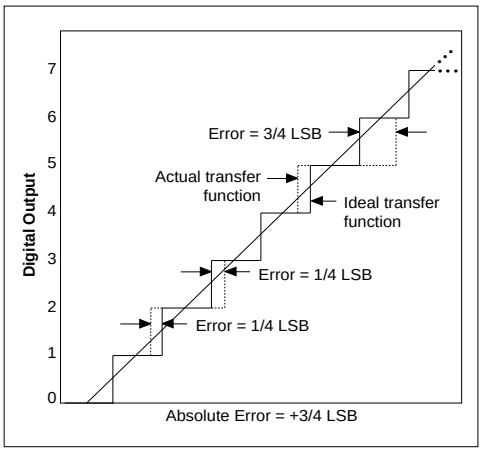
No missing code implies that as the analog input voltage is gradually increased from zero to full scale (or vice versa), all digital codes are produced. Stated otherwise, changing analog input voltage from one quantum of the analog range to the next adjacent range will not produce a change in the digital output by more than one code count.

Monotonic

Monotonicity guarantees that an increase (or decrease) in the analog input value will result in an equal or greater digital code (or less). Monotonicity does not guarantee that there are no missing codes. However, it is an important criterion for feedback control systems. Non-monotonicity may cause oscillations in such a system.

The first derivative of a monotonic function always has the same sign.

FIGURE 8 - ABSOLUTE ERROR



Ratiometric Conversion

It is the A/D conversion process where the binary result is a ratio of the supply voltage or reference voltage, the latter being equal to full-scale value by default. The PIC16C7X is a ratiometric A/D converter where the result depends on VDD or VREF.

In some A/D's, an absolute reference is provided resulting in “absolute conversion”.

Sample and Hold

In sample and hold type A/D converters, the analog input has a switch (typically a FET switch in CMOS) which is opened for a short duration to capture the analog input voltage onto an on-chip capacitor. Conversion is typically started after the sampling switch is closed.

Track and Hold

It is basically the same as sample and hold, except the sampling switch is typically left on. Therefore the voltage on the on-chip holding capacitor “tracks” the analog input voltage. To begin a conversion, the sampling switch is shut off.

The PIC16C7X A/D falls in this category.

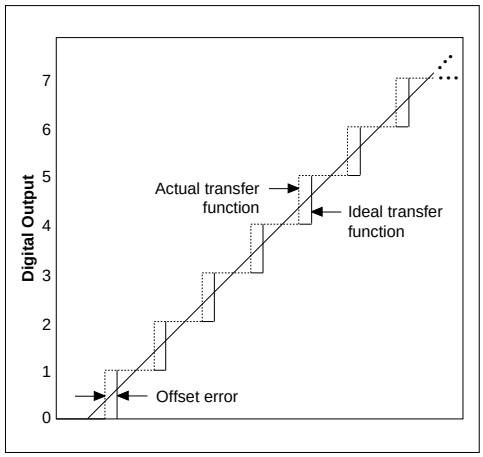
Sampling Time

It is the time required to charge the on-chip holding capacitor to the same value as on the analog input pin. The sampling time depends on the magnitude of the holding capacitor and the source impedance of the analog voltage input.

Offset Error (or Zero Error)

It is the difference between the first actual (measured) transition point and the first ideal transition point as shown in Figure 4. It can be corrected by the user by subtracting the offset error from each conversion result.

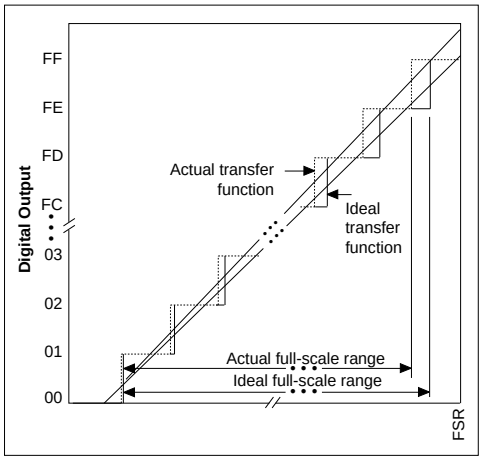
FIGURE 4 - OFFSET ERROR



Full Scale Error (or Gain Error)

It is the difference between the ideal Full Scale and the actual (measured) full scale range (see Figure 5). It is also called gain error, because the error changes the slope of the ideal transfer function creating a gain factor. It can be corrected by the user by multiplying each conversion result by the inverse of the gain.

FIGURE 5 - FULL SCALE ERROR

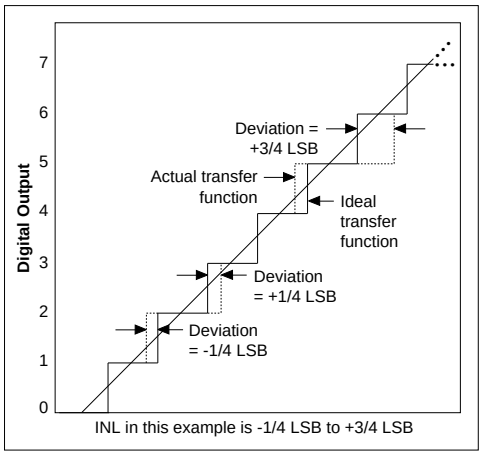


Integral Non-Linearity (INL), or Relative Error

It is the deviation of a transition point from its corresponding point on the ideal transfer curve (Figure 6). The maximum difference is reported as the INL of the converter.

It is important to note that Full Scale Error and the Offset Error are normalized to match end transition points before measuring the INL.

FIGURE 6 - INTEGRAL NON-LINEARITY



Using the Analog to Digital Converter

HOW TO USE THE PIC16C71 A/D

The A/D in the PIC16C71 is easy to set up and use. There are a few considerations:

1. Select either V_{DD} or V_{REF} as reference voltage. More on using V_{REF} input later.
2. Select A/D conversion clock (tad): 2 tosc, 8 tosc, 32 tosc or trc (internal RC clock). For the first three options, make sure that tad ≥ 2.0 µs. If deterministic conversion time is required, select tosc time base. If conversion during SLEEP is required, select trc.
3. Channel Selection : If only one A/D channel is required, program the ADCON1 register to 03h. This configures the A/D pins as digital I/O. If multiple channels are required, prior to each conversion the new channel must be selected.
4. Sampling and Conversion: After a new channel is selected, a minimum amount of sampling time must be allowed before GO bit in ADCON0 is set to begin conversion. Once conversion begins, it is OK to select the next channel, ***but sampling does not begin until current conversion is complete.*** Therefore, it is always necessary to provide minimum required sampling time

   i) after a conversion

   ii) after a new channel is selected

   iii) after A/D is turned on (ADON = 1).
5. Reading Result: Completion of conversion can be determined by either polling GO/$\overline{\text{DONE}}$ bit to cleared, polling the ADIF bit to be set, or waiting for an ADIF interrupt.

ADDITIONAL TIPS:

1. The GO bit and the ADON bit may not be set at once. After the A/D is turned on by setting ADON, at least 5µs time must be allowed before conversion begins, longer if sampling time requirement is not met within 5µs.
2. Aborting a conversion: A conversion can be aborted by clearing GO bit. The A/D converter will stop conversion and revert back to sampling state.
3. Using ADRES register as a normal register: The A/D only writes to ADRES at the end of a conversion. Therefore, it is possible to use ADRES as a normal file register between conversions and when A/D is off.

The following are a few examples of using the A/D.

EXAMPLE 1: HOW TO DO A SIMPLE ADC CONVERSION

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Always ch2, internal RC OSC.
InitializeAD
           bsf        STATUS, 5      ; select pg1
           movlw      B'00000000'    ; select RA0-RA3…
           movwf      ADCON1         ; as analog inputs
           bcf        STATUS, 5      ; select pg0
           movlw      B'11010001'    ; select: RC osc, ch2...
           movwf      ADCON0         ; turn on A/D
Convert    call       sample-delay   ; provide necessary sampling time
;
           bsf        ADCON0, 2      ; start new A/D conversion
loop
           btfsc      ADCON0, 2      ; A/D over?
           goto       loop           ; no then loop
;
           movf       ADRES, w       ; yes then get A/D value
;
```

A detailed code listing is in Appendix A.

EXAMPLE 2: HOW TO DO SEQUENTIAL CHANNEL CONVERSIONS

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3 in a round robin fashion, internal RC OSC.
; Load results in 4 consecutive addresses starting at ADTABLE (10h)
;
InitializeAD
          bsf       STATUS, 5     ; select pg1
          movlw     B'00000000'   ; select RA0-RA3…
          movwf     ADCON1        ; as analog inputs
          bcf       STATUS, 5     ; select pg0
          movlw     B'11000001'   ; select: RC osc, ch0...
          movwf     ADCON0        ; turn on A/D
          movlw     ADTABLE       ; point fsr to top of...
          movwf     FSR           ; table
;
new_ad    call      sample_delay  ; provide necessary sampling time
          bsf       ADCON0, 2     ; start new A/D conversion
loop
          btfsc     ADCON0, 2     ; A/D over?
          goto      loop          ; no then loop
;
          movf      adres, w      ; yes then get A/D value
          movwf     0             ; load indirectly
          movlw     4             ; select next channel
          addwd     ADCON0        ;      /
          bcf       ADCON0, 5     ; reset carry over bit.
; increment pointer to correct table offset.
          clrf      temp          ; clear temp register
          btfsc     ADCON0, 3     ; test lsb of channel select
          bsf       temp, 0       ; set if ch1 selected
          btfsc     ADCON0, 4     ; test msb of channel select
          bsf       temp, 1       ;      /
          movlw     ADTABLE       ; get table address
          addwf     temp, w       ; add with temp
          movwf     FSR           ; move into indirect
          goto      new_ad
;
```

A detailed code listing is in Appendix B.

EXAMPLE 3: HOW TO WRITE THE INTERRUPT HANDLER FOR THE ADC

```
            org         0x00
            goto        start
            org         0x04
            goto        service_ad      ; interrupt vector
;
;           org         0x10
start
            movlw       B'00000000'     ;init I/O ports
            movwf       PORT_B
            tris        PORT_B
;
            call        InitializeAD
update
            bcf         flag,adover     ; reset software A/D flag
            call        SetupDelay      ; setup delay >= 10uS.
            bcf         ADCON0,adif     ; reset A/D int flag (ADIF
            bsf         ADCON0,adgo     ; start new A/D conversion
            bsf         INTCON,gie      ; enable global interrupt
loop
            btfsc       flag,adover     ; A/D over?
            goto        update          ; yes start new conv.
            goto        loop            ; no then keep checking
; InitializeAD, initializes and sets up the A/D hardware.
; select ch0 to ch3, RC OSC., a/d interrupt.
InitializeAD
            bsf         STATUS, 5       ; select pg1
            movlw       B'00000000'     ; select RA0-RA3...
            movwf       ADCON1          ; as analog inputs
            bcf         STATUS, 5       ; select pg0
            clrf        INTCON          ; clr all interrupts
            bsf         INTCON, 6       ; enable A/D int.
            movlw       B'11010001'     ; select: RC osc, ch2...
            movwf       ADCON0          ; turn on A/D
            return
;
service_ad
            btfss       ADCON0, 1       ; A/D interrupt?
            retfie                      ; no then ignore
            movf        ADRES, W        ; get A/D value
            return                      ; do not enable int
;
```

A detailed code listing is in Appendix C.

EXAMPLE 4: HOW TO DO CONVERSIONS DURING SLEEP MODE

```
;
; InitializeAD, initializes and sets up the A/D hardware.
; Select ch0 to ch3, internal RC OSC.
; While doing the conversion put unit to sleep.  This will
; minimize digital noise interference.
; Note that ad's RC osc. has to be selected in this instance.
;
InitializeAD
            bsf         STATUS, 5       ; select pg1
            movlw       B'00000000'     ; select RA0-RA3...
            movwf       ADCON1          ; as analog inputs
            bcf         STATUS, 5       ; select pg0
            movlw       B'11000001'     ; select: RC osc, ch0...
            movwf       ADCON0          ; turn on A/D & ADIE
            movlw       ADTABLE         ; point fsr to top of...
            movwf       FSR             ; table
;
new_ad
            bsf         ADCON0, 2       ; start new A/D conversion
            sleep                       ; goto sleep
; when A/D is over program will continue from here
;
            movf        ADRES, w        ; get A/D value
;
```

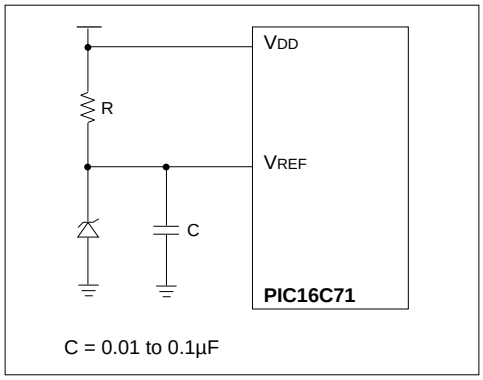
A detailed code listing is in Appendix D.

USING EXTERNAL REFERENCE VOLTAGE

When using external reference voltage, keep in mind that any analog input voltage must not exceed VREF.

An inexpensive way to generate VREF is by employing zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as 10 µA. However, larger currents (1mA - 20mA) are recommended for stability, as well as lower impedance of the VREF source.

FIGURE 9 - LOW COST VOLTAGE REFERENCE



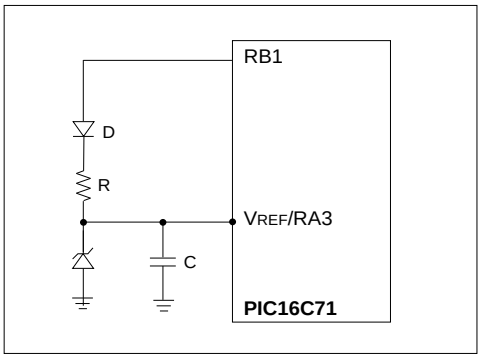
POWER MANAGEMENT IN USING VREF

In power sensitive applications, user may turn on VREF generator using another I/O pin as shown in Figure 10. Drive a “1” on RB1 pin in this example when using the A/D. Drive a “0” on RB1 pin when not using the A/D converter.

Note that this way RB1 is not floating. Even if VREF decays to some intermediate voltage, it will not cause the input buffer on RB1 to draw current.

Alternately, use RA0, RA1 or RA2 pin to supply the current instead of RB1. Configure the RA pin as analog (this will turn off its input buffer). Then use it as a digital output (Figure 11).

FIGURE 10 - POWER-SENSITIVE APPLICATIONS #1



ZENERS AND REFERENCE GENERATORS

Finally, various reference voltage generator chips (typically using on-chip band-gap reference) are available. These are more accurate.

TABLE 1 - ZENERS AND REFERENCE GENERATORS

Zeners	VZ	Tolerance
1N746	3.3V	±5%
1N747	3.6V	±5%
1N748	3.9V	±5%
1N749	4.3V	±5%
1N750	4.7V	±5%
1N751	5.1V	±5%
1N752	5.6V	±5%
Voltage References	**VREF**	**Tolerance**
AD580 (Maxim)	2.5V	±3% to ±0.4%
LM385	2.5V	±1.5%
LM1004	2.5V	±1.2%
LT1009 (LIN. Tech.)	2.5V	±0.2%
LT1019 (LIN. Tech.)	5.0V	±0.2%
LT1021 (LIN. Tech.)	5.0V	±0.05% to ±1%
LT1029 (LIN. Tech.)	5.0V	±0.2% to ±1%

VREF IMPEDANCE AND CURRENT SUPPLY REQUIREMENTS

Ideally, VREF should have as low a source impedance as possible. Referring to Figure 9, VREF source impedence ≈ R. However, smaller R increases current consumption. Since VREF is used to charge capacitor arrays inside the A/D converter and the holding capacitor CHOLD ≈ 51pF, the following guideline should be met:

tad = 6 (1K + R) 51.2 pF + 1.677µs

tad = conversion clock. For tad = 2µs and for CHOLD = 50 pF, RVREF ≈ 50Ω.

For VREF impedance higher than this, the conversion clock (tad) should be increased appropriately.

FIGURE 11 - POWER-SENSITIVE APPLICATIONS #2

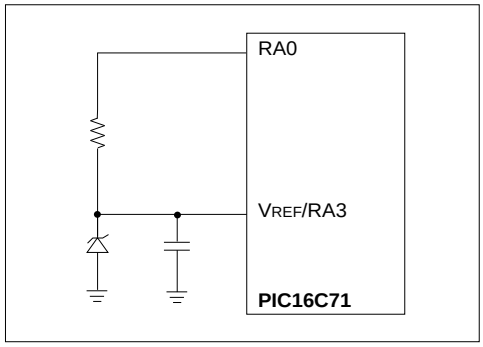


Table 2 gives examples of the maximum rate of conversion per bit, relating to the voltage reference impedance.

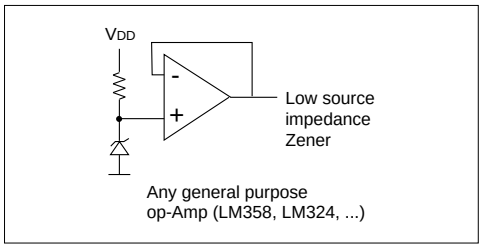
TABLE 2 - MAXIMUM RATE OF CONVERSION / BIT

RVREF	Tad (Max)
1K	2.29 µs
5K	3.52 µs
10K	5.056 µs
50K	16.66 µs
100K	32.70 µs

Assumes no external capacitors.

To achieve a low source impedance when using a Zener diode, a voltage follower circuit is recommended. This is shown in Figure 11A.

FIGURE 11A - VOLTAGE FOLLOWER CIRCUIT



CONFIGURING PORT A INPUTS AS ANALOG OR DIGITAL

Two bits in ADCON1 register PCFG1 and PCFG0 control how pins RA0–RA3 are configured. When any of these pins are selected as analog:

- The digital input buffer is turned off to save current (see Figure 12). Reading the port will read this pin as '0'.
- TRIS bit still controls the output buffer on this pin. So, normally the TRIS bit will be set (input).
- However, if the TRIS bit is cleared, then the pin will output whatever is in the data latch.

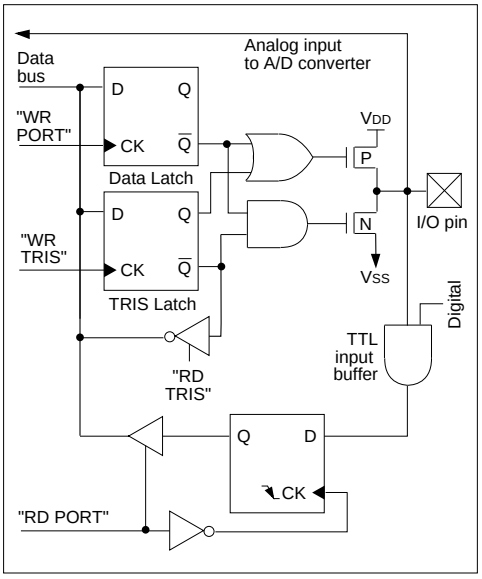
When any of these pins are selected as digital:

- The analog input still directly connects to the A/D and therefore the pin can be used as analog input.
- The digital input buffer is not disabled.

The user has, therefore, great flexibility in configuring these pins.

3

FIGURE 12 - BLOCK DIAGRAM OF RA0-RA3 PINS



CURRENT CONSUMPTION THROUGH INPUT BUFFER

A CMOS input buffer will draw current when the input voltage is around its threshold. (See Figure 13.)

In power-sensitive applications, the RA pins when used as analog inputs should be configured as "analog" to avoid unintended power drain.

Other considerations and tips:

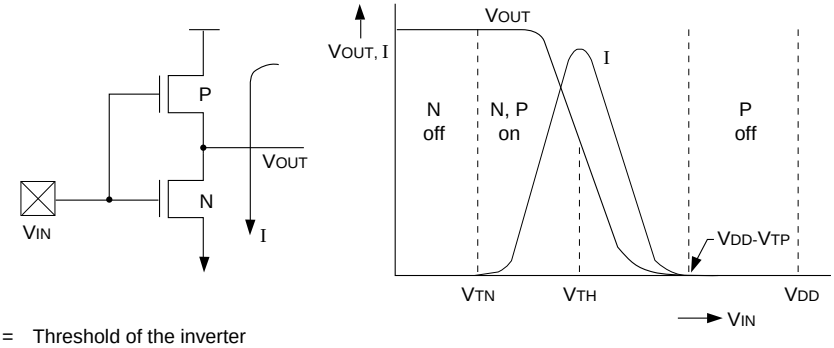
1. If possible, avoid any digital output next to analog inputs.
2. Avoid digital inputs that switch frequently (e.g., clocks) next to analog inputs.
3. If V_{REF} is used, then no analog pin being sampled should exceed V_{REF}.

SUMMARY

The PIC16C71 A/D converter is simple to use. It is versatile and low power.

Authors: *Sumit Mitra, Stan D'Souza and Russ Cooper*
Logic Products Division

FIGURE 13 - A SIMPLE CMOS INPUT BUFFER:



V_{TH} = Threshold of the inverter

V_{TN} = Device threshold of NMOS pull-down

$-V_{TP}$ = Device threshold of PMOS pull-up

I = On-current (or through current) of the inverter

I_{MAX} = Maximum on-current occurs when $V_{IN} = V_{TH}$. Value of I_{MAX} depends on the sizes of the devices. The larger the devices, the faster the input buffer, and the larger the value of I_{MAX}. Typically, $I_{MAX} \approx 0.2mA - 1mA$.

APPENDIX A - SINGLE CHANNEL A/D (SAD)

```
MPASM 1.00 Released       SAD.ASM   7-15-1994  13:27:21                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;TITLE   “Single channel A/D (SAD)”
                     0002 ;This program is a simple implementation of the PIC16C71’s
                     0003 ;A/D. 1 Channel is selected (CH0).
                     0004 ;The A/D is configured as follows:
                     0005 ;       Vref = +5V internal.
                     0006 ;       A/D Osc. =  internal RC
                     0007 ;       A/D Channel = CH0
                     0008 ;Hardware for this program is the PICDEMO board.
                     0009 ;
                     0010 ;
                     0011         LIST P=16C71,F=INHX8M
                     0012 ;
                     0013     include “picreg.equ”
                     0083
                     0013
                     0014 ;
0010                 0015 TEMP    EQU     10h
0001                 0016 adif    equ     1
0002                 0017 adgo    equ     2
                     0018 ;
                     0019         ORG     0x00
                     0020 ;
                     0021 ;
0000 2810            0022         goto    start
                     0023 ;
                     0024         org     0x04
0004 281C            0025         goto    service_int      ;interrupt vector
                     0026 ;
                     0027 ;
                     0028         org     0x10
                     0029 start
0010 3000            0030         movlw   B’00000000'     ;set port b as
0011 0086            0031         movwf   PORT_B          ;all outputs
0012 0066            0032         tris    PORT_B          ;       /
                     0033 ;
0013 201D            0034         call    InitializeAD
                     0035 update
0014 0809            0036         movf    ADRES,W         ;get a/d value
0015 0086            0037         movwf   PORT_B          ;output to port b
0016 2025            0038         call    SetupDelay      ;setup time >= 10uS.
0017 1088            0039         bcf     ADCON0,adif     ;clear int flag
0018 1508            0040         bsf     ADCON0,adgo     ;start new conversion
                     0041 loop
0019 1888            0042         btfsc   ADCON0,adif     ;a/d done?
001A 2814            0043         goto    update          ;yes then update new value.
001B 2819            0044         goto    loop            ;no then keep checking
                     0045 ;
                     0046 ;no interrupts are enabled, so if the program ever reaches here,
                     0047 ;it should be returned with the global interrupts disabled.
                     0048 service_int
001C 0008            0049         return                  ;do not enable global.
                     0050 ;
                     0051 ;
                     0052 ;
                     0053 ;InitializeAD, initializes and sets up the A/D hardware.
                     0054 ;Select ch0 to ch3 as analog inputs, fosc/2 and read ch3.
                     0055 ;
                     0056 InitializeAD
001D 1683            0057         bsf     STATUS,5        ;select pg1
001E 3000            0058         movlw   B’00000000'     ;select ch0-ch3...
001F 0088            0059         movwf   ADCON1          ;as analog inputs
```

```
0020 1283           0060         bcf     STATUS,5        ;select pg0
0021 30C1           0061         movlw   B'11000001'     ;select:RC,ch0..
0022 0088           0062         movwf   ADCON0          ;turn on A/D.
0023 0189           0063         clrf    ADRES           ;clr result reg.
0024 0008           0064         return
                    0065 ;
                    0066 ;This routine is a software delay of 10uS for the a/d setup.
                    0067 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0068 ;a value of 3 to give 9uS, plus the move etc should result in
                    0069 ;a total time of > 10uS.
                    0070 SetupDelay
0025 3003           0071         movlw   .3
0026 0090           0072         movwf   TEMP
                    0073 SD
0027 0B90           0074         decfsz  TEMP
0028 2827           0075         goto    SD
0029 0008           0076         return
                    0077
                    0078
                    0079         END
                    0080
                    0081


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X----- XXXXXXXXXXXXXXXX XXXXXXXXXX---- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

APPENDIX B

```
MPASM 1.00 Released     SLPAD.ASM   7-15-1994  13:27:15                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001
                     0002 ;TITLE   “A/D in Sleep Mode”
                     0003 ;This program is a simple implementation of the PIC16C71’s
                     0004 ;A/D feature. This program demonstrates
                     0005 ;how to do a a/d in sleep mode on the PIC16C71.
                     0006 ;The A/D is configured as follows:
                     0007 ;       Vref = +5V internal.
                     0008 ;       A/D Osc. = internal RC
                     0009 ;       A/D Interrupt = OFF
                     0010 ;       A/D Channels = ch 0
                     0011 ;
                     0012 ;The ch0 A/D result is displayed as a 8 bit binary value
                     0013 ;on 8 leds connected to port b. Hardware used is that of
                     0014 ;the PICDEMO board.
                     0015 ;
                     0016 ;
                     0017         LIST P=16C71,F=INHX8M
                     0018 ;
                     0019     include “picreg.equ”
                     0083
                     0019
                     0020 ;
0010                 0021 TEMP    EQU     10h
0001                 0022 adif    equ     1
0002                 0023 adgo    equ     2
                     0024 ;
                     0025 ;
                     0026         ORG     0x00
                     0027 ;
                     0028 ;
0000 2810            0029         goto    start
                     0030 ;
                     0031         org     0x04
0004 281B            0032         goto    service_int      ;interrupt vector
                     0033 ;
                     0034 ;
                     0035         org     0x10
                     0036 start
0010 3000            0037         movlw   B’00000000'     ;make port b all
0011 0086            0038         movwf   PORT_B          ;outputs.
0012 0066            0039         tris    PORT_B          ;       /
                     0040 ;
0013 201C            0041         call    InitializeAD
                     0042 update
0014 0809            0043         movf    ADRES,W
0015 0086            0044         movwf   PORT_B          ;save in table
0016 2025            0045         call    SetupDelay      ;
0017 1088            0046         bcf     ADCON0,adif     ;clr a/d flag
0018 1508            0047         bsf     ADCON0,adgo     ;start new a/d conversion
                     0048 ;
0019 0063            0049         sleep
001A 2814            0050         goto    update          ;wake up and update
                     0051 ;


                     0052 service_int
001B 0008            0053         return                  ;do not enable int
                     0054 ;
                     0055 ;InitializeAD, initializes and sets up the A/D hardware.
                     0056 InitializeAD
001C 1683            0057         bsf     STATUS,5        ;select pg1
001D 3000            0058         movlw   B’00000000'     ;select ch0-ch3...
```

```
001E 0088           0059         movwf   ADCON1          ;as analog inputs
001F 1283           0060         bcf     STATUS,5        ;select pg0
0020 30C1           0061         movlw   B'11000001'     ;select:internal RC, ch0.
0021 0088           0062         movwf   ADCON0          ;turn on a/d
0022 018B           0063         clrf    INTCON          ;clear all interrupts
0023 170B           0064         bsf     INTCON,ADIE     ;enable a/d
0024 0008           0065         return
                    0066 ;
                    0067 ;This routine is a software delay of 10uS for the a/d setup.
                    0068 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0069 ;a value of 3 to give 9uS, plus the move etc should result in
                    0070 ;a total time of > 10uS.
                    0071 SetupDelay
0025 3003           0072         movlw   .3
0026 0090           0073         movwf   TEMP
                    0074 SD
0027 0B90           0075         decfsz  TEMP
0028 2827           0076         goto    SD
0029 0008           0077         return
                    0078
                    0079 ;
                    0080
                    0081         END
                    0082
                    0083


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X----- XXXXXXXXXXXXXXXX XXXXXXXXXX---- --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

APPENDIX C

```
MPASM 1.00 Released     INTAD.ASM   7-15-1994  13:27:32                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001
                     0002 ;TITLE      “Single channel A/D with interrupts”
                     0003 ;This program is a simple implementation of the PIC16C71’s
                     0004 ;A/D. 1 Channel is selected (CH0). A/D interrupt is turned on,
                     0005 ;hence on completion of a/d conversion, an interrupt is generated.
                     0006 ;The A/D is configured as follows:
                     0007 ;       Vref = +5V internal.
                     0008 ;       A/D Osc. = internal RC Osc.
                     0009 ;       A/D Interrupt = On
                     0010 ;       A/D Channel = CH0
                     0011 ;
                     0012 ;The A/D result is displayed as a 8 bit value on 8 leds connected
                     0013 ;to port b. Hardware setup is the PICDEMO board.
                     0014 ;
                     0015 ;
                     0016         LIST P=16C71,F=INHX8M
                     0017 ;
                     0018     include “picreg.equ”
                     0083
                     0018
                     0019 ;
0010                 0020 flag    equ     10
0011                 0021 TEMP    equ     11
0000                 0022 adover  equ     0
0001                 0023 adif    equ     1
0002                 0024 adgo    equ     2
0006                 0025 adie    equ     6
0007                 0026 gie     equ     7
0005                 0027 rp0     equ     5
                     0028 ;
                     0029         ORG     0x00
                     0030 ;
                     0031 ;
0000 2810            0032         goto    start
                     0033 ;
                     0034         org     0x04
0004 281C            0035         goto    service_ad      ;interrupt vector
                     0036 ;
                     0037 ;
                     0038         org     0x10
                     0039 start
0010 3000            0040         movlw   B’00000000'     ;init i/o ports
0011 0086            0041         movwf   PORT_B
0012 0066            0042         tris    PORT_B
                     0043 ;
0013 2022            0044         call    InitializeAD
                     0045 update
0014 1010            0046         bcf     flag,adover     ;reset software a/d flag
0015 202B            0047         call    SetupDelay      ;setup delay >= 10uS.
0016 1088            0048         bcf     ADCON0,adif     ;reset a/d int flag (ADIF)
0017 1508            0049         bsf     ADCON0,adgo     ;start new a/d conversion
0018 178B            0050         bsf     INTCON,gie      ;enable global interrupt
                     0051 loop

0019 1810            0052         btfsc   flag,adover     ;a/d over?
001A 2814            0053         goto    update          ;yes start new conv.
001B 2819            0054         goto    loop            ;no then keep checking
                     0055 ;
                     0056 service_ad
001C 1C88            0057         btfss   ADCON0,adif     ;ad interrupt?
001D 0009            0058         retfie                  ;no then ignore
001E 0809            0059         movf    ADRES,W         ;get a/d value
```

```
001F 0086           0060         movwf   PORT_B          ;output to port b
0020 1410           0061         bsf     flag,adover     ;a/d done set
0021 0008           0062         return                  ;do not enable int
                    0063 ;
                    0064 ;
                    0065 ;InitializeAD, initializes and sets up the A/D hardware.
                    0066 ;select ch0 to ch3, RC OSC., a/d interrupt.
                    0067 InitializeAD
0022 1683           0068         bsf     STATUS,rp0      ;select pg1
0023 3000           0069         movlw   B'00000000'     ;select ch0-ch3...
0024 0088           0070         movwf   ADCON1          ;as analog inputs
0025 1283           0071         bcf     STATUS,rp0      ;select pg0
0026 018B           0072         clrf    INTCON          ;clr all interrupts
0027 170B           0073         bsf     INTCON,adie     ;enable a/d int.
0028 30C1           0074         movlw   B'11000001'     ;select:RC osc,ch0...
0029 0088           0075         movwf   ADCON0          ;turn on a/d
002A 0008           0076         return
                    0077 ;
                    0078 ;This routine is a software delay of 10uS for the a/d setup.
                    0079 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0080 ;a value of 3 to give 9uS, plus the move etc should result in
                    0081 ;a total time of > 10uS.
                    0082 SetupDelay
002B 3003           0083         movlw   .3
002C 0091           0084         movwf   TEMP
                    0085 SD
002D 0B91           0086         decfsz  TEMP
002E 282D           0087         goto    SD
002F 0008           0088         return
                    0089 ;
                    0090 ;
                    0091         END
                    0092
                    0093



MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-X------ XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX --------
0040 : -------- -------- -------- --------

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

APPENDIX D

```
MPASM 1.00 Released    MULTAD.ASM   7-15-1994  13:27:26                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;TITLE   “A/D using Multiple Channels”
                     0002 ;This program is a simple implementation of the PIC16C71’s
                     0003 ;A/D feature. This program demonstrates
                     0004 ;how to select multiple channels on the PIC16C71.
                     0005 ;The A/D is configured as follows:
                     0006 ;       Vref = +5V internal.
                     0007 ;       A/D Osc. = internal RC osc.
                     0008 ;       A/D Interrupt = Off
                     0009 ;       A/D Channels = all in a “Round Robin” format.
                     0010 ;       A/D reuslts are stored in ram locations as follows:
                     0011 ;       ch0 —> ADTABLE + 0
                     0012 ;       ch1 —> ADTABLE + 1
                     0013 ;       ch2 —> ADTABLE + 2
                     0014 ;       ch3 —> ADTABLE + 3
                     0015 ;
                     0016 ;The ch0 A/D result is displayed as a 8 bit value on 8 leds
                     0017 ;connected to port b.
                     0018 ;Hardware: PICDEMO board.
                     0019 ;               Stan D’Souza 7/6/93.
                     0020 ;
                     0021         LIST P=16C71,F=INHX8M
                     0022 ;
                     0023     include “picreg.equ”
                     0083
                     0023
                     0024 ;
0010                 0025 TEMP    EQU     10h
0001                 0026 adif    equ     1
0002                 0027 adgo    equ     2
                     0028 ;
0006                 0029 ch2     equ     6
0007                 0030 ch3     equ     7
000C                 0031 flag    equ     0C
0020                 0032 ADTABLE equ     20
                     0033 ;
                     0034         ORG     0x00
                     0035 ;
                     0036 ;
0000 2810            0037         goto    start
                     0038 ;
                     0039         org     0x04
0004 2823            0040         goto    service_int      ;interrupt vector
                     0041 ;
                     0042 ;
                     0043         org     0x10
                     0044 start
0010 3000            0045         movlw   B’00000000'     ;make port b
0011 0086            0046         movwf   PORT_B          ;as all outputs
0012 0066            0047         tris    PORT_B          ;       /
                     0048 ;
0013 2024            0049         call    InitializeAD
                     0050 update
0014 0809            0051         movf    ADRES,W

0015 0080            0052         movwf   0               ;save in table
0016 3020            0053         movlw   ADTABLE         ;chk if ch0
0017 0204            0054         subwf   FSR,W           ;     /
0018 1D03            0055         btfss   STATUS,Z        ;yes then skip
0019 281C            0056         goto    NextAd          ;else do next channel
001A 0809            0057         movf    ADRES,W         ;get a/d value
001B 0086            0058         movwf   PORT_B          ;output to port b
                     0059 NextAd
```

```
001C 202E           0060         call    NextChannel     ;select next channel
001D 203A           0061         call    SetupDelay      ;set up > = 10uS
001E 1088           0062         bcf     ADCON0,adif     ;clear flag
001F 1508           0063         bsf     ADCON0,adgo     ;start new a/d conversion
                    0064 loop
0020 1888           0065         btfsc   ADCON0,adif     ;a/d done?
0021 2814           0066         goto    update          ;yes then update
0022 2820           0067         goto    loop            ;wait till done
                    0068 ;
                    0069 service_int
0023 0008           0070         return                  ;do not enable int
                    0071 ;
                    0072 ;
                    0073 ;InitializeAD, initializes and sets up the A/D hardware.
                    0074 InitializeAD
0024 1683           0075         bsf     STATUS,5        ;select pg1
0025 3000           0076         movlw   B'00000000'     ;select ch0-ch3...
0026 0088           0077         movwf   ADCON1          ;as analog inputs
0027 1283           0078         bcf     STATUS,5        ;select pg0
0028 30C1           0079         movlw   B'11000001'     ;select:fosc/2, ch0.
0029 0088           0080         movwf   ADCON0          ;turn on a/d
002A 3020           0081         movlw   ADTABLE         ;get top of table address
002B 0084           0082         movwf   FSR             ;load into indirect reg
002C 0189           0083         clrf    ADRES           ;clr result reg.
002D 0008           0084         return
                    0085 ;
                    0086 ;NextChannel, selects the next channel to be sampled in a
                    0087 ;"round-robin" format.
                    0088 NextChannel
002E 3008           0089         movlw   0x08            ;get channel offset
002F 0788           0090         addwf   ADCON0          ;add to conf. reg.
0030 1288           0091         bcf     ADCON0,5        ;clear any carry over
                    0092 ;increment pointer to correct a/d result register
0031 0190           0093         clrf    TEMP
0032 1988           0094         btfsc   ADCON0,3        ;test lsb of chnl select
0033 1410           0095         bsf     TEMP,0          ;set if ch1 or ch3
0034 1A08           0096         btfsc   ADCON0,4        ;test msb of chnl select
0035 1490           0097         bsf     TEMP,1          ;set if ch0 or ch2
0036 3020           0098         movlw   ADTABLE         ;get top of table
0037 0710           0099         addwf   TEMP,W          ;add with temp
0038 0084           0100         movwf   FSR             ;allocate new address
0039 0008           0101         return
                    0102 ;
                    0103 ;This routine is a software delay of 10uS for the a/d setup.
                    0104 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    0105 ;a value of 3 to give 9uS, plus the move etc should result in
                    0106 ;a total time of > 10uS.
                    0107 SetupDelay
003A 3003           0108         movlw   .3
003B 0090           0109         movwf   TEMP
                    0110 SD
003C 0B90           0111         decfsz  TEMP
003D 283C           0112         goto    SD
003E 0008           0113         return
                    0114
                    0115 ;
                    0116
                    0117         END
                    0118
                    0119

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X—X——— XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-
0040 : ———————— ———————— ———————— ————————

All other memory blocks unused.

Errors   :    0
Warnings :    0
```

AN557

Four Channel Digital Voltmeter with Display and Keyboard

INTRODUCTION

The PIC16C71 is a member of a new family of 8-bit microcontrollers, namely the PIC16CXX. The salient features of the PIC16C71 are:

- Improved and enhanced instruction set
- 14-bit instruction word
- Interrupt capability
- On-chip four channel, 8-bit A/D converter

This application note demonstrates the capability of the PIC16C71. To make this application note easier for the end user, it has been broken down into four sub-sections:

Section 1: Implements the multiplexing of four 7 segment LEDs with the PIC16C71.
Section 2: Implements the multiplexing of four 7 segment LEDs as well as the sampling of a 4x4 Keypad.
Section 3: Implements the multiplexing of four 7 segment LEDs as well as the sampling of one A/D channel.
Section 4: Implements the multiplexing of four 7 segment LEDs, sampling a 4x4 keypad and four A/D channels.

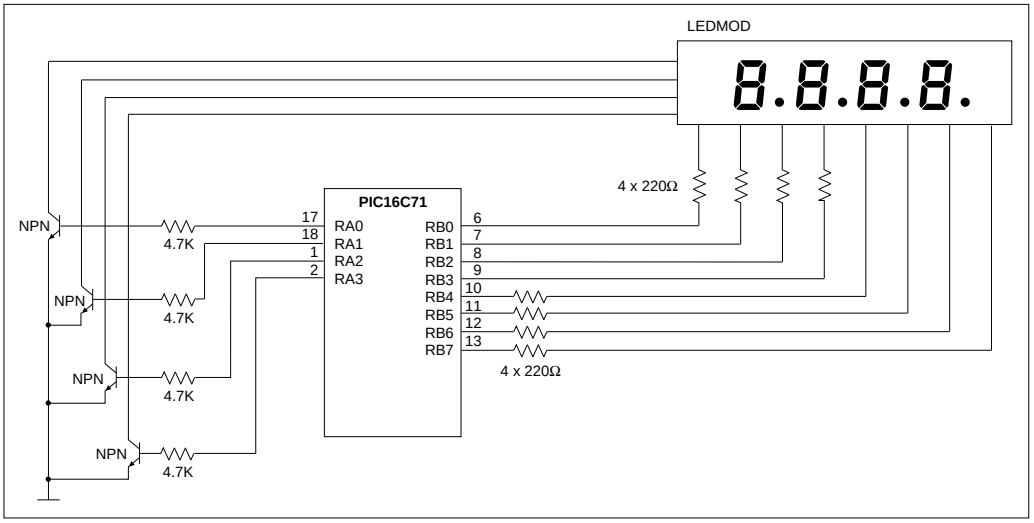
IMPLEMENTATION

SECTION 1: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS

Hardware

The PIC16C71's I/O ports have an improved sink/source specification. Each I/O pin can sink up to 25 mA and source 20 mA, in addition total Port B source current is 100 mA and sink current is 150 mA. Port A is rated for 50 mA source current and 80 mA sink current. This makes the PIC16C71 ideal for driving 7 segment LEDs. Since the total number of I/O is limited to 13, the 8-bit Port B is used to drive the 8 LEDs, while external sink transistors or MOSFETs are used to sink the digit current (See Figure 1). Another alternative is to use ULN2003 open collector sink current drivers, which are available in 16 pin DIP or very small S0-16 packages. Each transistor on the ULN2003 can sink a maximum of 500 mA and the base drive can be directly driven from the Port A pins.

FIGURE 1 - MULTIPLEXING FOUR 7 SEGMENT LEDS



Software

The multiplexing is achieved by turning on each LED for a 5 msec duration every 20 µs. This gives an update rate of 50 Hz, which is quite exceptable to the human eye as a steady display. The 5 msec time base is generated by dividing the 4.096 MHz oscillator clock. The internal prescaler is configured to be a divide by 32 and assigned to the RTCC. The RTCC is pre-loaded with a value = 96. The RTCC will increment to FF and then roll over to 00 after a period = (256-96)*(32*4/4096000) = 5 µs. When the RTCC rolls over, the RTIF flag is set and since the RTIE and GIE bits are enabled, an interrupt is generated.

The software implements a simple timer which increments at a 1 second rate. Every second, the 4 nibble (two 8-bit registers MsdTime and LsdTime) are incremented in a BCD format. The lower 4 bits of LsdTime correspond to the least significant digit (LSD) on the display. The high 4 bits of LsdTime correspond to the second significant digit of the display and so on. Depending on which display is turned on, the corresponding 4 bit BCD value is extracted from either MsdTime or LsdTime, and decoded to a 7 segment display. The RTCC interrupt is generated at a steady rate of 5 µs and given an instruction time of 1 us. The entire display update program can reside in the interrupt service routine with no chance of getting an interrupt within an interrupt. The Code listing for Section 1 is in Appendix A.

SECTION 2: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND SCANNING A 4X4 KEYPAD

Hardware

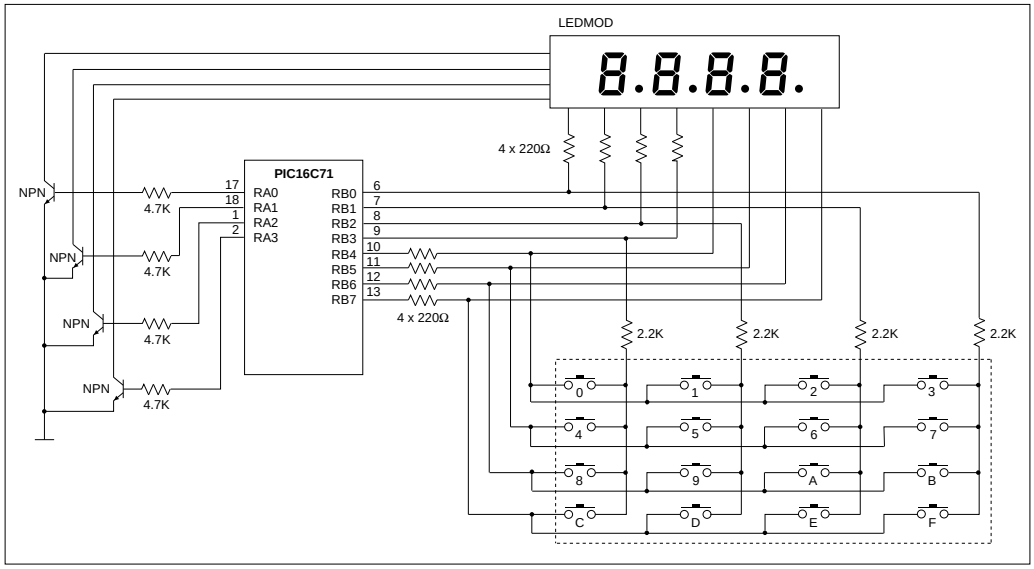
A 4x4 keypad can be very easily interfaced to the PIC16C71's Port B (see Figure 2). Internal pull-ups on pins RB4 to RB7 can be enabled and disabled by setting the RBPU bit in the OPTION register. The internal pull-ups have a value of 20K at 5V (typical). In order to sense a low level at the input, the switch is "connected" to ground through a 2.2K resistor. A key hit normally lasts from 50 msec to as long as a person holds the key down. In order not to miss any key hits, the keypad is sampled every 20 µs (just after the update of the MSD).

Software

To sample the keypad, the digit sinks are first disabled. Port B is then configured with RB4-RB7 as inputs and RB0 -RB3 as outputs driven high. The pull-ups on RB4-RB7 are enabled. Sequentially RB0 to RB3 are made low while RB4 to RB7 are checked for a key hit (a low level). One key hit per scan is demonstrated in this program. Multiple key hits per scan can very easily be implemented. Once the key hit is sensed, a 40 msec debounce period elapses before key sampling is resumed. No more key hits are sensed until the present key is released. This prevents erroneous key inputs.

The program basically inputs the key hit and displays its value as a hexadecimal character on the multiplexed 7-segment LEDs. The Code Listing for Section 2 is in Appendix B.

FIGURE 2 - MULTIPLEXING FOUR 7 SEGMENT LEDS WITH A 4X4 KEYPAD



SECTION 3: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS AND THE A/D CHANNEL 0

Hardware

The four analog channels are connected to RA0-RA3. If any of these pins are used normally as digital I/O, they can momentarily be used as analog inputs. In order to avoid interference from the analog source, it is advisable to buffer the analog input through a voltage follower op amp, however, it is not always necessary. Figure 3A and 3B show some typical configurations. In this application, the analog input is a potentiometer whose wiper is connected through an RC network to channel 0. The RC is necessary in order to smooth out the analog voltage. The RC does contribute to a delay in the sampling time, however the stability of the analog reading is greatly improved.

Software

The analog input is sampled every 20 msec. The digit sinks and the drivers are turned off i.e. Port A is configured as an input and Port B outputs are made low. A 1msec settling time is allowed for the external RC network connected to the analog input to settle and then the A/D conversion is started. The result is read then converted from an 8-bit binary value to a 3-digit BCD value which is then displayed on the 7 Segment LEDs. The Code Listing for Section 3 is in Appendix C.

FIGURE 3A - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT

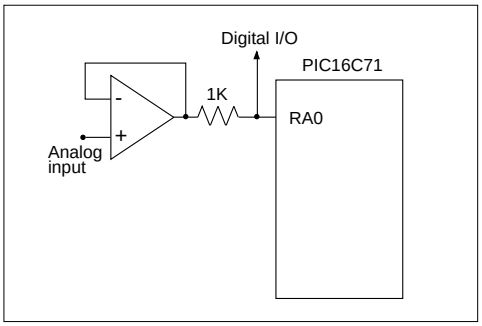
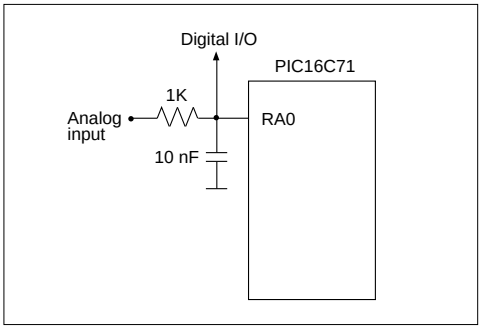


FIGURE 3B - TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT



3

Four Channel Digital Voltmeter with Display and Keyboard

SECTION 4: MULTIPLEXING FOUR 7 SEGMENT LED DISPLAYS WITH A 4X4 KEYPAD AND 4 A/D CHANNELS

Hardware

This section essentially incorporated Sections 1, 2 and 3 to give a complete four channel voltmeter. Figure 4 shows a typical configuration. The analog channels are connected through individual potentiometers to their respective analog inputs and are sampled every 20 msec in a round robin format. The sampling rate can be increased to as fast as once every 5 µs if required. The keypad sampling need not be any faster than once every 20 µs.

Software

The program samples the analog inputs and saves the result in four consecutive locations starting at "ADVALUE", with channel 0 saved at the first location and so on. By default, channel 0 is displayed. If Key 1 is pressed, channel 1 is displayed and so on. Key hits > 3 are ignored. The Code Listing for Section 4 is in Appendix D.

Code Size

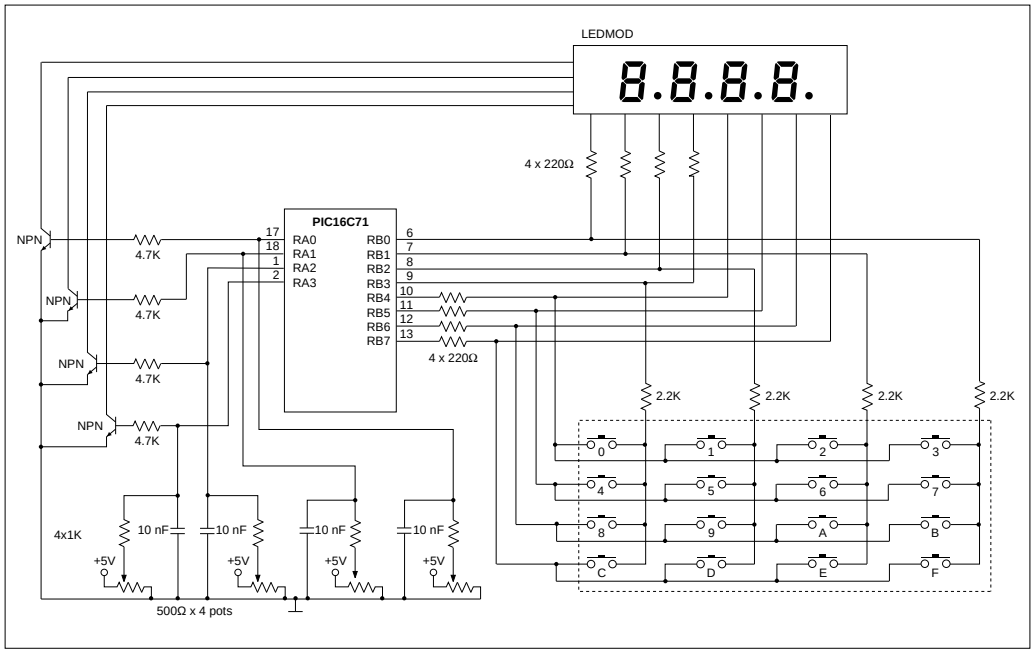
Section 1:	Program Memory:	139
	Data Memory:	6
Section 2:	Program Memory:	207
	Data Memory:	13
Section 3:	Program Memory:	207
	Data Memory:	17
Section 4:	Program Memory:	
	Data Memory:	

Conclusion

The four A/D channels on the PIC16C71 can be multiplexed with digital I/O, thus reducing overall pin counts and improving I/O pin usage in an analog application.

Author: Stan D'Souza, Logic Products Division

FIGURE 4 - FOUR CHANNEL VOLTMETER WITH DISPLAY AND KEYPAD



APPENDIX A

```
MPASM B0.50                                                    PAGE  1



                    ;*****************************************************************
                    ;This program is to demonstrate how to multiplex four 7 segment LED
                    ;digits using a PIC16C71. The four digits will start at 0000 and
                    ;increment at a 1 sec rate up to 9999.
                    ;The LEDs are updated every 5 msec, for a multiplexing rate of 20 msec.
                    ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                    ;
                    ;                                       Stan D'Souza 5/8/93
                    ;*****************************************************************
                           LIST P=16C71, F=INHX8M
                    ;
                           include "picreg.equ"



                    ;
000C                TempC          equ     0x0c           ;temp general purpose regs
000D                TempD          equ     0x0d
000E                TempE          equ     0x0e
000F                Count          equ     0x0f           ;count
0010                MsdTime        equ     0x10           ;most significant Timer
0011                LsdTime        equ     0x11           ;Least significant Timer
0001                OptionReg      equ     1
0002                PCL            equ     2
0026                BcdMsd         equ     26
0027                Bcd            equ     27
                    ;
                                   org     0
0000 2805                          goto    Start          ;skip over interrupt vector
                    ;
                                   org     4
0004 281D                          goto    ServiceInterrupts
                    ;
                    Start
0005 2008                          call    InitPorts
0006 2012                          call    InitTimers
                    loop
0007 2807                          goto    loop
                    ;
                    InitPorts
0008 1683                          bsf     STATUS,RP0     ;select pg 1
0009 3003                          movlw   3              ;make RA0-3 digital I/O
000A 0108                          movwf   ADCON1         ;       /
000B 0205                          clrf    TRISA          ;make RA0-4 outputs
000C 0206                          clrf    TRISB          ;make RB0-7 outputs
000D 1283                          bcf     STATUS,RP0     ;select page 0
000E 0185                          clrf    PORT_A         ;make all outputs low
000F 0186                          clrf    PORT_B         ;       /
0010 1585                          bsf     PORT_A,3       ;enable MSB digit sink
0011 0008                          return
                    ;
                    ;
                    ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                    ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                    ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                    ;end result is that we get a rtcc interrupt every 5msec.
                    InitTimers
0012 0190                          clrf    MsdTime        ;clr timers
0013 0191                          clrf    LsdTime        ;       /
0014 1683                          bsf     STATUS,RP0     ;select pg 1
0015 3084                          movlw   B'10000100'    ;assign ps to rtcc
```

```
0016 0081                       movwf   OptionReg             ;ps = 32
0017 1283                       bcf     STATUS,RP0           ;select pg 0
0018 3020                       movlw   B'00100000'          ;enable rtcc interrupt
0019 008B                       movwf   INTCON               ;
001A 3060                       movlw   .96                  ;preload rtcc
001B 0081                       movwf   RTCC                 ;start counter
001C 0009                       retfie
                        ;
                        ServiceInterrupts
001D 190B                       btfsc   INTCON,RTIF          ;rtcc interrupt?
001E 2822                       goto    ServiceRTCC          ;yes then service
001F 3020                       movlw   B'00100000'          ;else clr rest
0020 008B                       movwf   INTCON
0021 0009                       retfie
                        ;
                        ServiceRTCC
0022 3060                       movlw   .96                  ;initialize rtcc
0023 0081                       movwf   RTCC
0024 110B                       bcf     INTCON,RTIF          ;clr int flag
0025 2028                       call    IncTimer             ;inc timer
0026 2050                       call    UpdateDisplay        ;update display
0027 0009                       retfie
                        ;
                        ;The display is incremented every 200*5msec = 1 Sec.
                        IncTimer
0028 0A0F                       incf    Count,w              ;inc count
0029 3AC8                       xorlw   .200                 ;= 200?
002A 1903                       btfsc   STATUS,Z             ;no then skip
002B 282E                       goto    DoIncTime            ;else inc time
002C 0A8F                       incf    Count
002D 0008                       return
                        DoIncTime
002E 018F                       clrf    Count                ;clr count
002F 0A11                       incf    LsdTime,w            ;get lsd
0030 390F                       andlw   0x0F                 ;mask high nibble
0031 3A0A                       xorlw   0x0a                 ; = 10?
0032 1903                       btfsc   STATUS,Z             ;no then skip
0033 2836                       goto    IncSecondLsd         ;inc next lsd
0034 0A91                       incf    LsdTime              ;else inc timer
0035 0008                       return
                        IncSecondLsd
0036 0E11                       swapf   LsdTime,w            ;get hi in low nibble
0037 390F                       andlw   0x0F                 ;mask hi nibble
0038 3E01                       addlw   1                    ;inc it
0039 0091                       movwf   LsdTime              ;restore back
003A 0E91                       swapf   LsdTime              ;       /
003B 3A0A                       xorlw   0x0a                 ; = 10?
003C 1903                       btfsc   STATUS,Z             ;no then skip
003D 283F                       goto    IncThirdLsd          ;else inc next lsd
003E 0008                       return
                        IncThirdLsd
003F 0191                       clrf    LsdTime
0040 0A10                       incf    MsdTime,w            ;get 3rd lsd
0041 390F                       andlw   0x0F                 ;mask hi nibble
0042 3A0A                       xorlw   0x0a                 ;= 10?
0043 1903                       btfsc   STATUS,Z             ;no then skip
0044 2847                       goto    IncMsd               ;else Msd
0045 0A90                       incf    MsdTime              ;else inc timer
0046 0008                       return
                        IncMsd
0047 0E10                       swapf   MsdTime,w            ;get hi in lo nibble
0048 390F                       andlw   0x0F                 ;mask hi nibble
0049 3E01                       addlw   1                    ;inc timer
004A 0090                       movwf   MsdTime              ;restore back
004B 0E90                       swapf   MsdTime              ;       /
004C 3A0A                       xorlw   0x0a                 ;= 10?
004D 1903                       btfsc   STATUS,Z             ;no then skip
004E 0190                       clrf    MsdTime              ;clr msd
004F 0008                       return
```

```
                        ;
                        ;
                        UpdateDisplay
0050 0805                       movf    PORT_A,w            ;present sink value in w
0051 0185                       clrf    PORT_A              ;disable all digits sinks
0052 390F                       andlw   0x0f
0053 008C                       movwf   TempC               ;save sink value in tempC
0054 160C                       bsf     TempC,4             ;preset for lsd sink
0055 0C8C                       rrf     TempC               ;determine next sink value
0056 1C03                       btfss   STATUS,CARRY        ;c=1?
0057 118C                       bcf     TempC,3             ;no then reset LSD sink
0058 180C                       btfsc   TempC,0             ;else see if Msd
0059 286B                       goto    UpdateMsd           ;yes then do Msd
005A 188C                       btfsc   TempC,1             ;see if 3rdLsd
005B 2866                       goto    Update3rdLsd        ;yes then do 3rd Lsd
005C 190C                       btfsc   TempC,2             ;see if 2nd Lsd
005D 2861                       goto    Update2ndLsd        ;yes then do 2nd lsd
                        UpdateLsd
005E 0811                       movf    LsdTime,w           ;get Lsd in w
005F 390F                       andlw   0x0f                ;       /
0060 286F                       goto    DisplayOut          ;enable display
                        Update2ndLsd
0061 2080                       call    Chk2LsdZero         ;msd = 0 & 2 lsd 0?
0062 1D03                       btfss   STATUS,Z            ;yes then skip
0063 0E11                       swapf   LsdTime,w           ;get 2nd Lsd in w
0064 390F                       andlw   0x0f                ;mask rest
0065 286F                       goto    DisplayOut          ;enable display
                        Update3rdLsd
0066 2088                       call    ChkMsdZero          ;msd = 0?
0067 1D03                       btfss   STATUS,Z            ;yes then skip
0068 0810                       movf    MsdTime,w           ;get 3rd Lsd in w
0069 390F                       andlw   0x0f                ;mask low nibble
006A 286F                       goto    DisplayOut          ;enable display
                        UpdateMsd
006B 0E10                       swapf   MsdTime,w           ;get Msd in w
006C 390F                       andlw   0x0f                ;mask rest
006D 1903                       btfsc   STATUS,Z            ;msd != 0 then skip
006E 300A                       movlw   0x0a
                        DisplayOut
006F 2074                       call    LedTable            ;get digit output
0070 0086                       movwf   PORT_B              ;drive leds
0071 080C                       movf    TempC,w             ;get sink value in w
0072 0085                       movwf   PORT_A
0073 0008                       return
                        ;
                        ;
                        LedTable
0074 0782                       addwf   PCL                 ;add to PC low
0075 343F                       retlw   B'00111111'         ;led drive for 0
0076 3406                       retlw   B'00000110'         ;led drive for 1
0077 345B                       retlw   B'01011011'         ;led drive for 2
0078 344F                       retlw   B'01001111'         ;led drive for 3
0079 3466                       retlw   B'01100110'         ;led drive for 4
007A 346D                       retlw   B'01101101'         ;led drive for 5
007B 347D                       retlw   B'01111101'         ;led drive for 6
007C 3407                       retlw   B'00000111'         ;led drive for 7
007D 347F                       retlw   B'01111111'         ;led drive for 8
007E 3467                       retlw   B'01100111'         ;led drive for 9
007F 3400                       retlw   B'00000000'         ;blank led drive
                        ;
                        ;
                        Chk2LsdZero
0080 2088                       call    ChkMsdZero          ;msd = 0?
0081 1D03                       btfss   STATUS,Z            ;yes then skip
0082 0008                       return                      ;else return
0083 0E11                       swapf   LsdTime,w           ;get 2nd lsd
0084 390F                       andlw   0x0f                ;mask of LSD
0085 1D03                       btfss   STATUS,Z            ;0? then skip
0086 0008                       return
```

```
0087 340A                       retlw   10                      ;else return with 10
                        ;
                        ChkMsdZero
0088 0810                       movf    MsdTime,w               ;get Msd in w
0089 1D03                       btfss   STATUS,Z                ;= 0? skip
008A 0008                       return                          ;else return
008B 340A                       retlw   .10                     ;ret with 10
                        ;

                                end
                        ;
```

Appendix B

```
MPASM B0.50                                                          PAGE  1

                    ;*********************************************************************
                    ;This program is to demonstrate how to multiplex four 7 segment LED
                    ;digits and a 4X4 keypad using a PIC16C71.
                    ;The four digits will start as '0000' and when a key is hit
                    ;it is displayed on the 7 segment leds as a hex value 0 to F. The last
                    ;digit hit is always displayed on the right most led with the rest of
                    ;the digits shifted to the left. The left most digit is deleted.
                    ;The LEDs are updated every 20msec, the keypad is scanned at a rate of 20
msec.
                    ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                    ;
                    ;                                       Stan D'Souza 5/8/93
                    ;*********************************************************************
                           LIST P=16C71, F=INHX8M
                    ;
                           include "picreg.equ"


                    ;
000C                TempC          equ    0x0c           ;temp general purpose regs
000D                TempD          equ    0x0d
000E                TempE          equ    0x0e
0020                PABuf          equ    0x20
0021                PBBuf          equ    0x21
000F                Count          equ    0x0f           ;count
0010                MsdTime        equ    0x10           ;most significant Timer
0011                LsdTime        equ    0x11           ;Least significant Timer
0012                KeyFlag        equ    0x12           ;flags related to key pad
0000                keyhit         equ    0              ;bit 0 -> key-press on
0001                DebnceOn       equ    1              ;bit 1 -> debounce on
0002                noentry        equ    2              ;no key entry = 0
0003                ServKey        equ    3              ;bit 3 -> service key
0013                Debnce         equ    0x13           ;debounce counter
0014                NewKey         equ    0x14
002F                WBuffer        equ    0x2f
002E                StatBuffer     equ    0x2e
0001                OptionReg      equ    1
0002                PCL            equ    2
                    ;
                    ;
                    push           macro
                                   movwf  WBuffer        ;save w reg in Buffer
                                   swapf  WBuffer        ;swap it
                                   swapf  STATUS,w       ;get status
                                   movwf  StatBuffer     ;save it
                                   endm
                    ;
                    pop            macro
                                   swapf  StatBuffer,w   ;restore status
                                   movwf  STATUS         ;       /
                                   swapf  WBuffer,w      ;restore W reg
                                   endm
                    ;
                                   org    0
0000 280D                          goto   Start          ;skip over interrupt vector
                    ;
                                   org    4
                    ;It is always a good practice to save and restore the w reg,
                    ;and the status reg during a interrupt.
                                   push
0004 00AF                          movwf  WBuffer        ;save w reg in Buffer
0005 0EAF                          swapf  WBuffer        ;swap it
0006 0E03                          swapf  STATUS,w       ;get status
0007 00AE                          movwf  StatBuffer     ;save it
0008 2036                          call   ServiceInterrupts
```

```
                        pop
0009 0E2E               swapf   StatBuffer,w        ;restore status
000A 0083               movwf   STATUS              ;       /
000B 0E2F               swapf   WBuffer,w           ;restore W reg

000C 0009               retfie
                ;
                Start
000D 2020               call    InitPorts
000E 202A               call    InitTimers
                loop
000F 1992               btfsc   KeyFlag,ServKey     ;key service pending
0010 2012               call    ServiceKey          ;yes then service
0011 280F               goto    loop
                ;
                ;ServiceKey, does the software service for a keyhit. After a key service,
                ;the ServKey flag is reset, to denote a completed operation.
                ServiceKey
0012 0814               movf    NewKey,w            ;get key value
0013 008E               movwf   TempE               ;save in TempE
0014 0E10               swapf   MsdTime,w           ;move MSD out
0015 39F0               andlw   B'11110000'         ;clr lo nibble
0016 0090               movwf   MsdTime             ;save back
0017 0E11               swapf   LsdTime,w           ;get Lsd
0018 390F               andlw   B'00001111'         ;mask off lsd
0019 0490               iorwf   MsdTime             ;and left shift 3rd
001A 0E11               swapf   LsdTime,w           ;get Lsd again
001B 39F0               andlw   B'11110000'         ;mask off 2nd
001C 040E               iorwf   TempE,w             ;or with new lsd
001D 0091               movwf   LsdTime             ;make Lsd
001E 1192               bcf     KeyFlag,ServKey     ;reset service flag
001F 0008               return
                ;
                InitPorts
0020 1683               bsf     STATUS,RP0          ;select pg 1
0021 3003               movlw   3                   ;make RA0-3 digital I/O
0022 0108               movwf   ADCON1              ;       /
0023 0205               clrf    TRISA               ;make RA0-4 outputs
0024 0206               clrf    TRISB               ;make RB0-7 outputs
0025 1283               bcf     STATUS,RP0          ;select page 0
0026 0185               clrf    PORT_A              ;make all outputs low
0027 0186               clrf    PORT_B              ;       /
0028 1585               bsf     PORT_A,3            ;enable MSB digit sink
0029 0008               return
                ;
                ;
                ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                ;end result is that we get a rtcc interrupt every 5msec.
                InitTimers
002A 0190               clrf    MsdTime             ;clr timers
002B 0191               clrf    LsdTime             ;       /
002C 0192               clrf    KeyFlag             ;clr all flags
002D 1683               bsf     STATUS,RP0          ;select pg 1
002E 3084               movlw   B'10000100'         ;assign ps to rtcc
002F 0081               movwf   OptionReg           ;ps = 32
0030 1283               bcf     STATUS,RP0          ;select pg 0
0031 3020               movlw   B'00100000'         ;enable rtcc interrupt
0032 008B               movwf   INTCON              ;
0033 3060               movlw   .96                 ;preload rtcc
0034 0081               movwf   RTCC                ;start counter
0035 0009               retfie
                ;
                ServiceInterrupts
0036 190B               btfsc   INTCON,RTIF         ;rtcc interrupt?
0037 283B               goto    ServiceRTCC         ;yes then service
0038 018B               clrf    INTCON              ;else clr all int
0039 168B               bsf     INTCON,RTIE
```

```
003A 0008                       return
                        ;
                        ServiceRTCC
003B 3060                       movlw   .96                     ;initialize rtcc
003C 0081                       movwf   RTCC
003D 110B                       bcf     INTCON,RTIF             ;clr int flag
003E 1805                       btfsc   PORT_A,0                ;if msb on then do
003F 2042                       call    ScanKeys                ;do a quick key scan
0040 20A1                       call    UpdateDisplay           ;update display
0041 0008                       return
                        ;
                        ;
                        ;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
                        ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                        ;Debounce for a given keyhit is also taken care of.
                        ;The rate of key scan is 20msec with a 4.096Mhz clock.
                        ScanKeys
0042 1C92                       btfss   KeyFlag,DebnceOn        ;debounce on?
0043 2848                       goto    Scan1                   ;no then scan keypad
0044 0B93                       decfsz  Debnce                  ;else dec debounce time
0045 0008                       return                          ;not over then return
0046 1092                       bcf     KeyFlag,DebnceOn        ;over, clr debounce flag
0047 0008                       return                          ;and return
                        Scan1
0048 208A                       call    SavePorts               ;save port values
0049 30EF                       movlw   B'11101111'             ;init TempD
004A 008D                       movwf   TempD
                        ScanNext
004B 0806                       movf    PORT_B,w                ;read to init port
004C 100B                       bcf     INTCON,RBIF             ;clr flag
004D 0C8D                       rrf     TempD                   ;get correct column
004E 1C03                       btfss   STATUS,C                ;if carry set?
004F 2862                       goto    NoKey                   ;no then end
0050 080D                       movf    TempD,w                 ;else output
0051 0086                       movwf   PORT_B                  ;low column scan line
0052 0000                       nop
0053 1C0B                       btfss   INTCON,RBIF             ;flag set?
0054 284B                       goto    ScanNext                ;no then next
0055 1812                       btfsc   KeyFlag,keyhit          ;last key released?
0056 2860                       goto    SKreturn                ;no then exit
0057 1412                       bsf     KeyFlag,keyhit          ;set new key hit
0058 0E06                       swapf   PORT_B,w                ;read port
0059 008E                       movwf   TempE                   ;save in TempE
005A 2064                       call    GetKeyValue             ;get key value 0 - F
005B 0094                       movwf   NewKey                  ;save as New key
005C 1592                       bsf     KeyFlag,ServKey         ;set service flag
005D 1492                       bsf     KeyFlag,DebnceOn        ;set flag
005E 3004                       movlw   4
005F 0093                       movwf   Debnce                  ;load debounce time
                        SKreturn
0060 2097                       call    RestorePorts            ;restore ports
0061 0008                       return
                        ;
                        NoKey
0062 1012                       bcf     KeyFlag,keyhit          ;clr flag
0063 2860                       goto    SKreturn
                        ;
                        ;GetKeyValue gets the key as per the following layout
                        ;
                        ;                       Col1    Col2    Col3    Col3
                        ;                       (RB3)   (RB2)   (RB1)   (RB0)
                        ;
                        ;Row1(RB4)                 0       1       2       3
                        ;
                        ;Row2(RB5)                 4       5       6       7
                        ;
                        ;Row3(RB6)                 8       9       A       B
                        ;
                        ;Row4(RB7)                 C       D       E       F
```

```
                        ;
                        GetKeyValue
0064 018C                       clrf    TempC
0065 1D8D                       btfss   TempD,3         ;first column
0066 286E                       goto    RowValEnd
0067 0A8C                       incf    TempC
0068 1D0D                       btfss   TempD,2         ;second col.
0069 286E                       goto    RowValEnd
006A 0A8C                       incf    TempC
006B 1C8D                       btfss   TempD,1         ;3rd col.
006C 286E                       goto    RowValEnd
006D 0A8C                       incf    TempC           ;last col.
                        RowValEnd
006E 1C0E                       btfss   TempE,0         ;top row?
006F 2878                       goto    GetValCom       ;yes then get 0,1,2&3
0070 1C8E                       btfss   TempE,1         ;2nd row?
0071 2877                       goto    Get4567         ;yes the get 4,5,6&7
0072 1D0E                       btfss   TempE,2         ;3rd row?
0073 2875                       goto    Get89ab         ;yes then get 8,9,a&b
                        Getcdef
0074 150C                       bsf     TempC,2         ;set msb bits
                        Get89ab
0075 158C                       bsf     TempC,3         ;       /
0076 2878                       goto    GetValCom       ;do common part
                        Get4567
0077 150C                       bsf     TempC,2
                        GetValCom
0078 080C                       movf    TempC,w
0079 0782                       addwf   PCL
007A 3400                       retlw   0
007B 3401                       retlw   1
007C 3402                       retlw   2
007D 3403                       retlw   3
007E 3404                       retlw   4
007F 3405                       retlw   5
0080 3406                       retlw   6
0081 3407                       retlw   7
0082 3408                       retlw   8
0083 3409                       retlw   9
0084 340A                       retlw   0a
0085 340B                       retlw   0b
0086 340C                       retlw   0c
0087 340D                       retlw   0d
0088 340E                       retlw   0e
0089 340F                       retlw   0f
                        ;
                        ;SavePorts, saves the porta and portb condition during a key scan
                        ;operation.
                        SavePorts
008A 0805                       movf    PORT_A,w        ;Get sink value
008B 00A0                       movwf   PABuf           ;save in buffer
008C 0185                       clrf    PORT_A          ;disable all sinks
008D 0806                       movf    PORT_B,w        ;get port b
008E 00A1                       movwf   PBBuf           ;save in buffer
008F 30FF                       movlw   0xff            ;make all high
0090 0086                       movwf   PORT_B          ;on port b
0091 1683                       bsf     STATUS,RP0      ;select page 1
0092 1381                       bcf     OptionReg,7     ;enable pull ups
0093 30F0                       movlw   B'11110000'     ;port b hi nibble inputs
0094 0106                       movwf   TRISB           ;lo nibble outputs
0095 1283                       bcf     STATUS,RP0      ;page 0
0096 0008                       return
                        ;
                        ;RestorePorts, restores the condition of porta and portb after a
                        ;key scan operation.
                        RestorePorts
0097 0821                       movf    PBBuf,w         ;get port n
0098 0086                       movwf   PORT_B
0099 0820                       movf    PABuf,w         ;get port a value
```

```
009A 0085                       movwf   PORT_A
009B 1683                       bsf     STATUS,RP0      ;select page 1
009C 1781                       bsf     OptionReg,7     ;disable pull ups
009D 0205                       clrf    TRISA           ;make port a outputs
009E 0206                       clrf    TRISB           ;as well as PORTB
009F 1283                       bcf     STATUS,RP0      ;page 0
00A0 0008                       return
                        ;
                        ;
                        UpdateDisplay
00A1 0805                       movf    PORT_A,w        ;present sink value in w
00A2 0185                       clrf    PORT_A          ;disable all digits sinks
00A3 390F                       andlw   0x0f
00A4 008C                       movwf   TempC           ;save sink value in tempC
00A5 160C                       bsf     TempC,4         ;preset for lsd sink
00A6 0C8C                       rrf     TempC           ;determine next sink value
00A7 1C03                       btfss   STATUS,CARRY    ;c=1?
00A8 118C                       bcf     TempC,3         ;no then reset LSD sink
00A9 180C                       btfsc   TempC,0         ;else see if Msd
00AA 28B8                       goto    UpdateMsd       ;yes then do Msd
00AB 188C                       btfsc   TempC,1         ;see if 3rdLsd
00AC 28B5                       goto    Update3rdLsd    ;yes then do 3rd Lsd
00AD 190C                       btfsc   TempC,2         ;see if 2nd Lsd
00AE 28B2                       goto    Update2ndLsd    ;yes then do 2nd lsd
                        UpdateLsd
00AF 0811                       movf    LsdTime,w       ;get Lsd in w
00B0 390F                       andlw   0x0f            ;       /
00B1 28BA                       goto    DisplayOut
                        Update2ndLsd
00B2 0E11                       swapf   LsdTime,w       ;get 2nd Lsd in w
00B3 390F                       andlw   0x0f            ;mask rest
00B4 28BA                       goto    DisplayOut      ;enable display
                        Update3rdLsd
00B5 0810                       movf    MsdTime,w       ;get 3rd Lsd in w
00B6 390F                       andlw   0x0f            ;mask low nibble
00B7 28BA                       goto    DisplayOut      ;enable display
                        UpdateMsd
00B8 0E10                       swapf   MsdTime,w       ;get Msd in w
00B9 390F                       andlw   0x0f            ;mask rest
                        DisplayOut
00BA 20BF                       call    LedTable        ;get digit output
00BB 0086                       movwf   PORT_B          ;drive leds
00BC 080C                       movf    TempC,w         ;get sink value in w
00BD 0085                       movwf   PORT_A
00BE 0008                       return
                        ;
                        ;
                        LedTable
00BF 0782                       addwf   PCL             ;add to PC low
00C0 343F                       retlw   B'00111111'     ;led drive for 0
00C1 3406                       retlw   B'00000110'     ;led drive for 1
00C2 345B                       retlw   B'01011011'     ;led drive for 2
00C3 344F                       retlw   B'01001111'     ;led drive for 3
00C4 3466                       retlw   B'01100110'     ;led drive for 4
00C5 346D                       retlw   B'01101101'     ;led drive for 5
00C6 347D                       retlw   B'01111101'     ;led drive for 6
00C7 3407                       retlw   B'00000111'     ;led drive for 7
00C8 347F                       retlw   B'01111111'     ;led drive for 8
00C9 3467                       retlw   B'01100111'     ;led drive for 9
00CA 3477                       retlw   B'01110111'     ;led drive for A
00CB 347C                       retlw   B'01111100'     ;led drive for b
00CC 3439                       retlw   B'00111001'     ;led drive for C
00CD 345E                       retlw   B'01011110'     ;led drive for d
00CE 3479                       retlw   B'01111001'     ;led drive for E
00CF 3471                       retlw   B'01110001'     ;led drive for F
                        ;
                        ;
                               end
                        ;
```

Appendix C

```
MPASM B0.50                                                    PAGE  1


                        ;*********************************************************************
                        ;This program is to demonstrate how to multiplex four 7 segment LED
                        ;and sample ch0 of the a/d in a PIC16C71. The a/d value is displayed
                        ;as a 3 digit decimal value of the a/d input (0 - 255).
                        ;The LEDs are updated every 20msec, the a/d is sampled every 20 msec.
                        ;The RTCC timer is used in internal interrupt mode to generate the 5 msec.
                        ;
                        ;                                       Stan D'Souza 5/8/93
                        ;*********************************************************************
                               LIST P=16C71, F=INHX8M
                        ;
                               include "picreg.equ"



                        ;
0026                    BcdMsd          equ     26
0027                    Bcd             equ     27
000C                    TempC           equ     0x0c            ;temp general purpose regs
000D                    TempD           equ     0x0d
000E                    TempE           equ     0x0e
0020                    PABuf           equ     0x20
0021                    PBBuf           equ     0x21
000F                    Count           equ     0x0f            ;count
0010                    MsdTime         equ     0x10            ;most significant Timer
0011                    LsdTime         equ     0x11            ;Least significant Timer
0012                    ADFlag          equ     0x12            ;flags related to key pad
0005                    ADOver          equ     5               ;bit 5 —> a/d over
002F                    WBuffer         equ     0x2f
002E                    StatBuffer      equ     0x2e
0001                    OptionReg       equ     1
0002                    PCL             equ     2
                        ;
                        push            macro
                        movwf           WBuffer                 ;save w reg in Buffer
                        swapf           WBuffer                 ;swap it
                        swapf           STATUS,w                ;get status
                        movwf           StatBuffer              ;save it
                        endm
                        ;
                        pop             macro
                        swapf           StatBuffer,w            ;restore status
                        movwf           STATUS                  ;       /
                        swapf           WBuffer,w               ;restore W reg
                        endm
                        ;
                        org             0
0000 280D               goto            Start                   ;skip over interrupt vector
                        ;
                        org             4
                        ;It is always a good practice to save and restore the w reg,
                        ;and the status reg during a interrupt.
                        push
0004 00AF               movwf           WBuffer                 ;save w reg in Buffer
0005 0EAF               swapf           WBuffer                 ;swap it
0006 0E03               swapf           STATUS,w                ;get status
0007 00AE       `       movwf           StatBuffer              ;save it

0008 2039               call            ServiceInterrupts
                        pop
0009 0E2E               swapf           StatBuffer,w            ;restore status
000A 0083                     movwf   STATUS          ;       /
```

```
000B 0E2F                       swapf   WBuffer,w               ;restore W reg

000C 0009                       retfie
                        ;
                        Start
000D 2021                       call    InitPorts
000E 202B                       call    InitTimers
000F 2036                       call    InitAd
                        loop
0010 1A92                       btfsc   ADFlag,ADOver           ;a/d over?
0011 2013                       call    UpdateAd                ;yes then update
0012 2810                       goto    loop
                        ;
                        UpdateAd
0013 1C88                       btfss   ADCON0,ADIF             ;a/d done?
0014 0008                       return                          ;no then leave
0015 0809                       movf    ADRES,W                 ;get a/d value
0016 00A1                       movwf   L_byte
0017 01A0                       clrf    H_byte
0018 20AD                       call    B2_BCD
0019 0824                       movf    R2,W                    ;get LSd
001A 0091                       movwf   LsdTime                 ;save in LSD
001B 0823                       movf    R1,W                    ;get Msd
001C 0090                       movwf   MsdTime                 ;save in Msd
001D 1088                       bcf     ADCON0,ADIF             ;clr interrupt flag
001E 1008                       bcf     ADCON0,ADON             ;turn off a/d
001F 1292                       bcf     ADFlag,ADOver           ;clr flag
0020 0008                       return
                        ;
                        ;
                        ;
                        InitPorts
0021 1683                       bsf     STATUS,RP0              ;select pg 1
0022 3003                       movlw   3                       ;make RA0-3 digital I/O
0023 0108                       movwf   ADCON1                  ;       /
0024 0205                       clrf    TRISA                   ;make RA0-4 outputs
0025 0206                       clrf    TRISB                   ;make RB0-7 outputs
0026 1283                       bcf     STATUS,RP0              ;select page 0
0027 0185                       clrf    PORT_A                  ;make all outputs low
0028 0186                       clrf    PORT_B                  ;       /
0029 1585                       bsf     PORT_A,3                ;enable MSB digit sink
002A 0008                       return
                        ;
                        ;
                        ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                        ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                        ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5msec. So the
                        ;end result is that we get a rtcc interrupt every 5msec.
                        InitTimers
002B 0190                       clrf    MsdTime                 ;clr timers
002C 0191                       clrf    LsdTime                 ;       /
002D 1683                       bsf     STATUS,RP0              ;select pg 1
002E 3084                       movlw   B'10000100'             ;assign ps to rtcc
002F 0081                       movwf   OptionReg               ;ps = 32
0030 1283                       bcf     STATUS,RP0              ;select pg 0
0031 3020                       movlw   B'00100000'             ;enable rtcc interrupt
0032 008B                       movwf   INTCON                  ;
0033 3060                       movlw   .96                     ;preload rtcc
0034 0081                       movwf   RTCC                    ;start counter
0035 0009                       retfie
                        ;
                        ;
                        InitAd
0036 30C8                       movlw   B'11001000'             ;init a/d
0037 0088                       movwf   ADCON0
0038 0008                       return
                        ;
```

```
                    ;
                    ServiceInterrupts
0039 190B                   btfsc   INTCON,RTIF             ;rtcc interrupt?
003A 283E                   goto    ServiceRTCC             ;yes then service
003B 018B                   clrf    INTCON
003C 168B                   bsf     INTCON,RTIE
003D 0008                   return
                    ;
                    ServiceRTCC
003E 3060                   movlw   .96                     ;initialize rtcc
003F 0081                   movwf   RTCC
0040 110B                   bcf     INTCON,RTIF             ;clr int flag
0041 1C05                   btfss   PORT_A,0                ;last digit?
0042 2045                   call    SampleAd                ;then sample a/d
0043 2071                   call    UpdateDisplay           ;else update display
0044 0008                   return
                    ;
                    ;
                    SampleAd
0045 205A                   call    SavePorts
0046 204C                   call    DoAd                    ;do a ad conversion
                    AdDone
0047 1908                   btfsc   ADCON0,GO               ;ad done?
0048 2847                   goto    AdDone                  ;no then loop
0049 1692                   bsf     ADFlag,ADOver           ;set a/d over flag
004A 2067                   call    RestorePorts            ;restore ports
004B 0008                   return
                    ;
                    ;
                    DoAd
004C 0186                   clrf    PORT_B                  ;turn off leds
004D 1683                   bsf     STATUS,RP0              ;select pg 1
004E 300F                   movlw   0x0f                    ;make port a hi-Z
004F 0105                   movwf   TRISA                   ;       /
0050 1283                   bcf     STATUS,RP0              ;select pg 0
0051 1408                   bsf     ADCON0,ADON             ;start a/d
0052 307D                   movlw   .125
0053 2056                   call    Wait
0054 1508                   bsf     ADCON0,GO               ;start conversion
0055 0008                   return
                    ;
                    ;
                    Wait
0056 008C                   movwf   TempC                   ;store in temp
                    Next
0057 0B8C                   decfsz  TempC
0058 2857                   goto    Next
0059 0008                   return

                    ;
                    ;SavePorts, saves the porta and portb condition during a key scan
                    ;operation.
                    SavePorts
005A 0805                   movf    PORT_A,w                ;Get sink value
005B 00A0                   movwf   PABuf                   ;save in buffer
005C 0185                   clrf    PORT_A                  ;disable all sinks
005D 0806                   movf    PORT_B,w                ;get port b
005E 00A1                   movwf   PBBuf                   ;save in buffer
005F 30FF                   movlw   0xff                    ;make all high
0060 0086                   movwf   PORT_B                  ;on port b
0061 1683                   bsf     STATUS,RP0              ;select page 1
0062 1381                   bcf     OptionReg,7             ;enable pull ups
0063 30F0                   movlw   B'11110000'             ;port b hi nibble inputs
0064 0106                   movwf   TRISB                   ;lo nibble outputs
0065 1283                   bcf     STATUS,RP0              ;page 0
0066 0008                   return
                    ;
                    ;RestorePorts, restores the condition of porta and portb after a
                    ;key scan operation.
```

```
                        RestorePorts
0067 0821                       movf    PBBuf,w         ;get port n
0068 0086                       movwf   PORT_B
0069 0820                       movf    PABuf,w         ;get port a value
006A 0085                       movwf   PORT_A
006B 1683                       bsf     STATUS,RP0      ;select page 1
006C 1781                       bsf     OptionReg,7     ;disable pull ups
006D 0205                       clrf    TRISA           ;make port a outputs
006E 0206                       clrf    TRISB           ;as well as PORTB
006F 1283                       bcf     STATUS,RP0      ;page 0
0070 0008                       return
                        ;
                        ;
                        UpdateDisplay
0071 0805                       movf    PORT_A,w        ;present sink value in w
0072 0185                       clrf    PORT_A          ;disable all digits sinks
0073 390F                       andlw   0x0f
0074 008C                       movwf   TempC           ;save sink value in tempC
0075 160C                       bsf     TempC,4         ;preset for lsd sink
0076 0C8C                       rrf     TempC           ;determine next sink value
0077 1C03                       btfss   STATUS,CARRY    ;c=1?
0078 118C                       bcf     TempC,3         ;no then reset LSD sink
0079 180C                       btfsc   TempC,0         ;else see if Msd
007A 288C                       goto    UpdateMsd       ;yes then do Msd
007B 188C                       btfsc   TempC,1         ;see if 3rdLsd
007C 2887                       goto    Update3rdLsd    ;yes then do 3rd Lsd
007D 190C                       btfsc   TempC,2         ;see if 2nd Lsd
007E 2882                       goto    Update2ndLsd    ;yes then do 2nd lsd
                        UpdateLsd
007F 0811                       movf    LsdTime,w       ;get Lsd in w
0080 390F                       andlw   0x0f            ;       /
0081 2890                       goto    DisplayOut      ;enable display
                        Update2ndLsd
0082 20A1                       call    Chk2LsdZero     ;msd = 0 & 2 lsd 0?
0083 1D03                       btfss   STATUS,Z        ;yes then skip
0084 0E11                       swapf   LsdTime,w       ;get 2nd Lsd in w
0085 390F                       andlw   0x0f            ;mask rest
0086 2890                       goto    DisplayOut      ;enable display
                        Update3rdLsd
0087 20A9                       call    ChkMsdZero      ;msd = 0?
0088 1D03                       btfss   STATUS,Z        ;yes then skip
0089 0810                       movf    MsdTime,w       ;get 3rd Lsd in w
008A 390F                       andlw   0x0f            ;mask low nibble
008B 2890                       goto    DisplayOut      ;enable display
                        UpdateMsd
008C 0E10                       swapf   MsdTime,w       ;get Msd in w
008D 390F                       andlw   0x0f            ;mask rest
008E 1903                       btfsc   STATUS,Z        ;msd != 0 then skip
008F 300A                       movlw   0x0a
                        DisplayOut
0090 2095                       call    LedTable        ;get digit output
0091 0086                       movwf   PORT_B          ;drive leds
0092 080C                       movf    TempC,w         ;get sink value in w
0093 0085                       movwf   PORT_A
0094 0008                       return
                        ;
                        ;
                        LedTable
0095 0782                       addwf   PCL             ;add to PC low
0096 343F                       retlw   B'00111111'     ;led drive for 0
0097 3406                       retlw   B'00000110'     ;led drive for 1
0098 345B                       retlw   B'01011011'     ;led drive for 2
0099 344F                       retlw   B'01001111'     ;led drive for 3
009A 3466                       retlw   B'01100110'     ;led drive for 4
009B 346D                       retlw   B'01101101'     ;led drive for 5
009C 347D                       retlw   B'01111101'     ;led drive for 6
009D 3407                       retlw   B'00000111'     ;led drive for 7
009E 347F                       retlw   B'01111111'     ;led drive for 8
009F 3467                       retlw   B'01100111'     ;led drive for 9
```

```
00A0 3400                       retlw          B'00000000'    ;blank led drive
                          ;
                          ;
                          Chk2LsdZero
00A1 20A9                       call           ChkMsdZero     ;msd = 0?
00A2 1D03                       btfss          STATUS,Z       ;yes then skip
00A3 0008                       return                        ;else return
00A4 0E11                       swapf          LsdTime,w      ;get 2nd lsd
00A5 390F                       andlw          0x0f           ;mask of LSD
00A6 1D03                       btfss          STATUS,Z       ;0? then skip
00A7 0008                       return
00A8 340A                       retlw          .10            ;else return with 10
                          ;
                          ChkMsdZero
00A9 0810                       movf           MsdTime,w      ;get Msd in w
00AA 1D03                       btfss          STATUS,Z       ;= 0? skip
00AB 0008                       return                        ;else return
00AC 340A                       retlw          .10            ;ret with 10
                          ;
                          ;
                          ;
0026                      count   equ    26
0027                      temp    equ    27
                          ;
0020                      H_byte  equ    20
0021                      L_byte  equ    21
0022                      R0      equ    22                    ; RAM Assignments
0023                      R1      equ    23
0024                      R2      equ    24
                          ;
                          ;
00AD 1003                 B2_BCD  bcf    STATUS,0              ; clear the carry bit
00AE 3010                         movlw  .16
00AF 00A6                         movwf  count
00B0 01A2                         clrf   R0
00B1 01A3                         clrf   R1
00B2 01A4                         clrf   R2
00B3 0DA1                 loop16  rlf    L_byte
00B4 0DA0                         rlf    H_byte
00B5 0DA4                         rlf    R2
00B6 0DA3                         rlf    R1
00B7 0DA2                         rlf    R0
                          ;
00B8 0BA6                         decfsz count
00B9 28BB                         goto   adjDEC
00BA 3400                         RETLW  0
                          ;
00BB 3024                 adjDEC  movlw  R2
00BC 0084                         movwf  FSR
00BD 20C5                         call   adjBCD
                          ;
00BE 3023                         movlw  R1
00BF 0084                         movwf  FSR
00C0 20C5                         call   adjBCD
                          ;
00C1 3022                         movlw  R0
00C2 0084                         movwf  FSR
00C3 20C5                         call   adjBCD
                          ;
00C4 28B3                         goto   loop16
                          ;
00C5 3003                 adjBCD  movlw  3
00C6 0700                         addwf  0,W
00C7 00A7                         movwf  temp
00C8 19A7                         btfsc  temp,3                ; test if result > 7
00C9 0080                         movwf  0
00CA 3030                         movlw  30
00CB 0700                         addwf  0,W
00CC 00A7                         movwf  temp
```

```
00CD 1BA7                       btfsc   temp,7          ; test if result > 7
00CE 0080                       movwf   0               ; save as MSD
00CF 3400                       RETLW   0
                        ;
                        ;

                                end
                        ;
```

APPENDIX D

```
MPASM 1.00 Released    MPLXAD.ASM   7-15-1994  13:43:14                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001 ;*****************************************************************
                     0002 ;This program is to demonstrate how to multiplex four 7 segment LED
                     0003 ;digits and a 4X4 keypad along with 4 A/D inputs using a PIC16C71.
                     0004 ;The four digits will first display the decimal a/d value of ch0.
                     0005 ;When keys from 0 - 3 are hit the corresponding channel's a/d value
                     0006 ;is displayed in decimal.
                     0007 ;The LEDs are updated every 20mS, the keypad is scanned at a rate of 20 mS.
                     0008 ;All 4 channels are scanned at 20mS rate, so each channel gets scanned
                     0009 ;every 80mS. A faster rate of scanning is possible as required by
                     0010 ;the users application.
                     0011 ;The RTCC timer is used in internal interrupt mode to generate the
                     0012 ;5 mS.
                     0013 ;
                     0014 ;                                        Stan D'Souza 5/8/93
                     0015 ;
                     0016 ;Corrected error in display routine.
                     0017 ;                                        Stan D'Souza 2/27/94
                     0018 ;*****************************************************************
                     0019         LIST P=16C71, F=INHX8M
                     0020 ;
                     0021         include "picreg.equ"
                     0083
                     0084
                     0021
                     0022 ;
000C                 0023 TempC   equ     0x0c            ;temp general purpose regs
000D                 0024 TempD   equ     0x0d
000E                 0025 TempE   equ     0x0e
0020                 0026 PABuf   equ     0x20
0021                 0027 PBBuf   equ     0x21
000F                 0028 Count   equ     0x0f           ;count
0010                 0029 MsdTime equ     0x10            ;most significant Timer
0011                 0030 LsdTime equ     0x11            ;Least significant Timer
                     0031 ;
0012                 0032 Flag    equ     0x12            ;general purpose flag reg
0001                 0033  #define keyhit  Flag,0         ;bit 0 —> key-press on
0002                 0034  #define DebnceOn Flag,1        ;bit 1 -> debounce on
0003                 0035  #define noentry Flag,2         ;no key entry = 0
0004                 0036  #define ServKey Flag,3         ;bit 3 —> service key
0005                 0037  #define ADOver  Flag,4         ;bit 4 —> a/d conv. over
                     0038 ;
0013                 0039 Debnce  equ     0x13            ;debounce counter
0014                 0040 NewKey  equ     0x14
0015                 0041 DisplayCh equ   0x15            ;channel to be displayed
                     0042 ;
0016                 0043 ADTABLE equ     0x16            ;4 locations are reserved here
                     0044                                 ;from 0x16 to 0x19
                     0045 ;
002F                 0046 WBuffer equ     0x2f
002E                 0047 StatBuffer equ  0x2e
0001                 0048 OptionReg equ   1
0002                 0049 PCL     equ     2
                     0050 ;
                     0051 ;
                     0052 push    macro
                     0053         movwf   WBuffer         ;save w reg in Buffer
                     0054         swapf   WBuffer         ;swap it
                     0055         swapf   STATUS,w        ;get status
                     0056         movwf   StatBuffer      ;save it
                     0057         endm
                     0058 ;
```

```
                     0059 pop     macro
                     0060         swapf   StatBuffer,w    ;restore status
                     0061         movwf   STATUS          ;       /
                     0062         swapf   WBuffer,w       ;restore W reg
                     0063         endm
                     0064 ;
                     0065         org     0
0000 280D            0066         goto    Start           ;skip over interrupt vector
                     0067 ;
                     0068         org     4
                     0069 ;It is always a good practice to save and restore the w reg,
                     0070 ;and the status reg during a interrupt.
                     0071         push
0004 00AF               M         movwf   WBuffer         ;save w reg in Buffer
0005 0EAF               M         swapf   WBuffer         ;swap it
0006 0E03               M         swapf   STATUS,w        ;get status
0007 00AE               M         movwf   StatBuffer      ;save it
0008 2052            0072         call    ServiceInterrupts
                     0073         pop
0009 0E2E               M         swapf   StatBuffer,w    ;restore status
000A 0083               M         movwf   STATUS          ;       /
000B 0E2F               M         swapf   WBuffer,w       ;restore W reg
000C 0009            0074         retfie
                     0075 ;
                     0076 Start
000D 203B            0077         call    InitPorts
000E 20EE            0078         call    InitAd
000F 2045            0079         call    InitTimers
                     0080 loop
0010 1992            0081         btfsc   ServKey         ;key service pending
0011 2015            0082         call    ServiceKey      ;yes then service
0012 1A12            0083         btfsc   ADOver          ;a/d pending?
0013 2028            0084         call    ServiceAD       ;yes the service a/d
0014 2810            0085         goto    loop
                     0086 ;
                     0087 ;ServiceKey, does the software service for a keyhit. After a key service,
                     0088 ;the ServKey flag is reset, to denote a completed operation.
                     0089 ServiceKey
0015 1192            0090         bcf     ServKey         ;reset service flag
0016 0814            0091         movf    NewKey,w        ;get key value
0017 3C03            0092         sublw   3               ;key > 3?
0018 1C03            0093         btfss   STATUS,C        ;no then skip
0019 0008            0094         return                  ;else ignore key
001A 0814            0095         movf    NewKey,w
001B 0095            0096         movwf   DisplayCh       ;load new channel
                     0097 ;
                     0098 LoadAD
001C 3016            0099         movlw   ADTABLE         ;get top of table
001D 0715            0100         addwf   DisplayCh,w     ;add offset
001E 0084            0101         movwf   FSR             ;init FSR
001F 0800            0102         movf    0,w             ;get a/d value
0020 00A1            0103         movwf   L_byte
0021 01A0            0104         clrf    H_byte
0022 2106            0105         call    B2_BCD
0023 0824            0106         movf    R2,W            ;get LSd
0024 0091            0107         movwf   LsdTime         ;save in LSD
0025 0823            0108         movf    R1,W            ;get Msd
0026 0090            0109         movwf   MsdTime         ;save in Msd
0027 0008            0110         return
                     0111 ;
                     0112 ;This rountine essentially loads the ADRES value in the table location
                     0113 ;determined by the channel offset. If channel 0 then ADRES is saved
                     0114 ;in location ADTABLE. If channel 1 then ADRES is saved at ADTABLE + 1.
                     0115 ;and so on.
                     0116 ServiceAD
0028 0808            0117         movf    ADCON0,w        ;get adcon0
0029 008C            0118         movwf   TempC           ;save in temp
002A 3008            0119         movlw   B'00001000'     ;select next channel
002B 0708            0120         addwf   ADCON0,w        ;       /
```

```
002C 1A88           0121        btfsc   ADCON0,5        ;if <= ch3
002D 30C1           0122        movlw   B'11000001'     ;select ch0
002E 0088           0123        movwf   ADCON0
                    0124        ;now load adres in the table
002F 3016           0125        movlw   ADTABLE
0030 0084           0126        movwf   FSR             ;load FSR with top
0031 0C8C           0127        rrf     TempC
0032 0C8C           0128        rrf     TempC
0033 0C0C           0129        rrf     TempC,w         ;get in w reg
0034 3903           0130        andlw   3               ;mask off all but last 2
0035 0784           0131        addwf   FSR             ;add offset to table
0036 0809           0132        movf    ADRES,w         ;get a/d value
0037 0080           0133        movwf   0               ;load indirectly
0038 1212           0134        bcf     ADOver          ;clear flag
0039 201C           0135        call    LoadAD          ;load a/d value in display reg.
003A 0008           0136        return
                    0137
                    0138
                    0139
                    0140 ;
                    0141 InitPorts
003B 1683           0142        bsf     STATUS,RP0      ;select pg 1
003C 3003           0143        movlw   3               ;make RA0-3 digital I/O
003D 0088           0144        movwf   ADCON1          ;       /
003E 0185           0145        clrf    TRISA           ;make RA0-4 outputs
003F 0186           0146        clrf    TRISB           ;make RB0-7 outputs
0040 1283           0147        bcf     STATUS,RP0      ;select page 0
0041 0185           0148        clrf    PORT_A          ;make all outputs low
0042 0186           0149        clrf    PORT_B          ;       /
0043 1585           0150        bsf     PORT_A,3        ;enable MSB digit sink
0044 0008           0151        return
                    0152 ;
                    0153 ;
                    0154 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                    0155 ;the rtcc will be incremented every 31.25uS. If rtcc is preloaded
                    0156 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                    0157 ;end result is that we get a rtcc interrupt every 5mS.
                    0158 InitTimers
0045 0190           0159        clrf    MsdTime         ;clr timers
0046 0191           0160        clrf    LsdTime         ;       /
0047 0195           0161        clrf    DisplayCh       ;show channel 0
0048 0192           0162        clrf    Flag            ;clr all flags
0049 1683           0163        bsf     STATUS,RP0      ;select pg 1
004A 3084           0164        movlw   B'10000100'     ;assign ps to rtcc
004B 0081           0165        movwf   OptionReg       ;ps = 32
004C 1283           0166        bcf     STATUS,RP0      ;select pg 0
004D 3020           0167        movlw   B'00100000'     ;enable rtcc interrupt
004E 008B           0168        movwf   INTCON          ;
004F 3060           0169        movlw   .96             ;preload rtcc
0050 0081           0170        movwf   RTCC            ;start counter
0051 0009           0171        retfie
                    0172 ;
                    0173 ServiceInterrupts
0052 190B           0174        btfsc   INTCON,RTIF     ;rtcc interrupt?
0053 2857           0175        goto    ServiceRTCC     ;yes then service
0054 018B           0176        clrf    INTCON          ;else clr all int
0055 168B           0177        bsf     INTCON,RTIE
0056 0008           0178        return
                    0179 ;
                    0180 ServiceRTCC
0057 3060           0181        movlw   .96             ;initialize rtcc
0058 0081           0182        movwf   RTCC
0059 110B           0183        bcf     INTCON,RTIF     ;clr int flag
005A 1805           0184        btfsc   PORT_A,0        ;scan keys every 20 mS
005B 2060           0185        call    ScanKeys        ;when digit 1 is on
005C 1985           0186        btfsc   PORT_A,3        ;scan a/d every 20mS
005D 20F1           0187        call    SampleAd        ;when digit 4 is on
005E 20BF           0188        call    UpdateDisplay   ;update display
005F 0008           0189        return
```

```
                    0190 ;
                    0191 ;
                    0192 ;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
                    0193 ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                    0194 ;Debounce for a given keyhit is also taken care of.
                    0195 ;The rate of key scan is 20mS with a 4.096Mhz clock.
                    0196 ScanKeys
0060 1C92           0197         btfss   DebnceOn        ;debounce on?
0061 2866           0198         goto    Scan1           ;no then scan keypad
0062 0B93           0199         decfsz  Debnce          ;else dec debounce time
0063 0008           0200         return                  ;not over then return
0064 1092           0201         bcf     DebnceOn        ;over, clr debounce flag
0065 0008           0202         return                  ;and return
                    0203 Scan1
0066 20A8           0204         call    SavePorts       ;save port values
0067 30EF           0205         movlw   B'11101111'     ;init TempD
0068 008D           0206         movwf   TempD
                    0207 ScanNext
0069 0806           0208         movf    PORT_B,w        ;read to init port
006A 100B           0209         bcf     INTCON,RBIF     ;clr flag
006B 0C8D           0210         rrf     TempD           ;get correct column
006C 1C03           0211         btfss   STATUS,C        ;if carry set?
006D 2880           0212         goto    NoKey           ;no then end
006E 080D           0213         movf    TempD,w         ;else output
006F 0086           0214         movwf   PORT_B          ;low column scan line
0070 0000           0215         nop
0071 1C0B           0216         btfss   INTCON,RBIF     ;flag set?
0072 2869           0217         goto    ScanNext        ;no then next
0073 1812           0218         btfsc   keyhit          ;last key released?
0074 287E           0219         goto    SKreturn        ;no then exit
0075 1412           0220         bsf     keyhit          ;set new key hit
0076 0E06           0221         swapf   PORT_B,w        ;read port
0077 008E           0222         movwf   TempE           ;save in TempE
0078 2082           0223         call    GetKeyValue     ;get key value 0 - F
0079 0094           0224         movwf   NewKey          ;save as New key
007A 1592           0225         bsf     ServKey         ;set service flag
007B 1492           0226         bsf     DebnceOn        ;set flag
007C 3004           0227         movlw   4
007D 0093           0228         movwf   Debnce          ;load debounce time
                    0229 SKreturn
007E 20B5           0230         call    RestorePorts    ;restore ports
007F 0008           0231         return
                    0232 ;
                    0233 NoKey
0080 1012           0234         bcf     keyhit          ;clr flag
0081 287E           0235         goto    SKreturn
                    0236 ;
                    0237 ;GetKeyValue gets the key as per the following layout
                    0238 ;
                    0239 ;                   Col1    Col2    Col3    Col3
                    0240 ;                   (RB3)   (RB2)   (RB1)   (RB0)
                    0241 ;
                    0242 ;Row1(RB4)           0       1       2       3
                    0243 ;
                    0244 ;Row2(RB5)           4       5       6       7
                    0245 ;
                    0246 ;Row3(RB6)           8       9       A       B
                    0247 ;
                    0248 ;Row4(RB7)           C       D       E       F
                    0249 ;
                    0250 GetKeyValue
0082 018C           0251         clrf    TempC
0083 1D8D           0252         btfss   TempD,3         ;first column
0084 288C           0253         goto    RowValEnd
0085 0A8C           0254         incf    TempC
0086 1D0D           0255         btfss   TempD,2         ;second col.
0087 288C           0256         goto    RowValEnd
0088 0A8C           0257         incf    TempC
```

```
0089 1C8D           0258         btfss   TempD,1         ;3rd col.
008A 288C           0259         goto    RowValEnd
008B 0A8C           0260         incf    TempC           ;last col.
                    0261 RowValEnd
008C 1C0E           0262         btfss   TempE,0         ;top row?
008D 2896           0263         goto    GetValCom       ;yes then get 0,1,2&3
008E 1C8E           0264         btfss   TempE,1         ;2nd row?
008F 2895           0265         goto    Get4567         ;yes the get 4,5,6&7
0090 1D0E           0266         btfss   TempE,2         ;3rd row?
0091 2893           0267         goto    Get89ab         ;yes then get 8,9,a&b
                    0268 Getcdef
0092 150C           0269         bsf     TempC,2         ;set msb bits
                    0270 Get89ab
0093 158C           0271         bsf     TempC,3         ;       /
0094 2896           0272         goto    GetValCom       ;do common part
                    0273 Get4567
0095 150C           0274         bsf     TempC,2
                    0275 GetValCom
0096 080C           0276         movf    TempC,w
0097 0782           0277         addwf   PCL
0098 3400           0278         retlw   0
0099 3401           0279         retlw   1
009A 3402           0280         retlw   2
009B 3403           0281         retlw   3
009C 3404           0282         retlw   4
009D 3405           0283         retlw   5
009E 3406           0284         retlw   6
009F 3407           0285         retlw   7
00A0 3408           0286         retlw   8
00A1 3409           0287         retlw   9
00A2 340A           0288         retlw   0a
00A3 340B           0289         retlw   0b
00A4 340C           0290         retlw   0c
00A5 340D           0291         retlw   0d
00A6 340E           0292         retlw   0e
00A7 340F           0293         retlw   0f
                    0294 ;
                    0295 ;SavePorts, saves the porta and portb condition during a key scan
                    0296 ;operation.
                    0297 SavePorts
00A8 0805           0298         movf    PORT_A,w        ;Get sink value
00A9 00A0           0299         movwf   PABuf           ;save in buffer
00AA 0185           0300         clrf    PORT_A          ;disable all sinks
00AB 0806           0301         movf    PORT_B,w        ;get port b
00AC 00A1           0302         movwf   PBBuf           ;save in buffer
00AD 30FF           0303         movlw   0xff            ;make all high
00AE 0086           0304         movwf   PORT_B          ;on port b
00AF 1683           0305         bsf     STATUS,RP0      ;select page 1
00B0 1381           0306         bcf     OptionReg,7     ;enable pull ups
00B1 30F0           0307         movlw   B'11110000'     ;port b hi nibble inputs
00B2 0086           0308         movwf   TRISB           ;lo nibble outputs
00B3 1283           0309         bcf     STATUS,RP0      ;page 0
00B4 0008           0310         return
                    0311 ;
                    0312 ;RestorePorts, restores the condition of porta and portb after a
                    0313 ;key scan operation.
                    0314 RestorePorts
00B5 0821           0315         movf    PBBuf,w         ;get port b
00B6 0086           0316         movwf   PORT_B
00B7 0820           0317         movf    PABuf,w         ;get port a value
00B8 0085           0318         movwf   PORT_A
00B9 1683           0319         bsf     STATUS,RP0      ;select page 1
00BA 1781           0320         bsf     OptionReg,7     ;disable pull ups
00BB 0185           0321         clrf    TRISA           ;make port a outputs
00BC 0186           0322         clrf    TRISB           ;as well as PORTB
00BD 1283           0323         bcf     STATUS,RP0      ;page 0
00BE 0008           0324         return
                    0325 ;
                    0326 ;
```

```
                     0327 UpdateDisplay
00BF 0805            0328         movf    PORT_A,w        ;present sink value in w
00C0 0185            0329         clrf    PORT_A          ;disable all digits sinks
00C1 390F            0330         andlw   0x0f
00C2 008C            0331         movwf   TempC           ;save sink value in tempC
00C3 160C            0332         bsf     TempC,4         ;preset for lsd sink
00C4 0C8C            0333         rrf     TempC           ;determine next sink value
00C5 1C03            0334         btfss   STATUS,CARRY    ;c=1?
00C6 118C            0335         bcf     TempC,3         ;no then reset LSD sink
00C7 180C            0336         btfsc   TempC,0         ;else see if Msd
00C8 28D6            0337         goto    UpdateMsd       ;yes then do Msd
00C9 188C            0338         btfsc   TempC,1         ;see if 3rdLsd
00CA 28D3            0339         goto    Update3rdLsd    ;yes then do 3rd Lsd
00CB 190C            0340         btfsc   TempC,2         ;see if 2nd Lsd
00CC 28D0            0341         goto    Update2ndLsd    ;yes then do 2nd lsd
                     0342 UpdateLsd
00CD 0811            0343         movf    LsdTime,w       ;get Lsd in w
00CE 390F            0344         andlw   0x0f            ;       /
00CF 28D8            0345         goto    DisplayOut
                     0346 Update2ndLsd
00D0 0E11            0347         swapf   LsdTime,w       ;get 2nd Lsd in w
00D1 390F            0348         andlw   0x0f            ;mask rest
00D2 28D8            0349         goto    DisplayOut      ;enable display
                     0350 Update3rdLsd
00D3 0810            0351         movf    MsdTime,w       ;get 3rd Lsd in w
00D4 390F            0352         andlw   0x0f            ;mask low nibble
00D5 28D8            0353         goto    DisplayOut      ;enable display
                     0354 UpdateMsd
00D6 0E10            0355         swapf   MsdTime,w       ;get Msd in w
00D7 390F            0356         andlw   0x0f            ;mask rest
                     0357 DisplayOut
00D8 20DD            0358         call    LedTable        ;get digit output
00D9 0086            0359         movwf   PORT_B          ;drive leds
00DA 080C            0360         movf    TempC,w         ;get sink value in w
00DB 0085            0361         movwf   PORT_A
00DC 0008            0362         return
                     0363 ;
                     0364 ;
                     0365 LedTable
00DD 0782            0366         addwf   PCL             ;add to PC low
00DE 343F            0367         retlw   B'00111111'     ;led drive for 0
00DF 3406            0368         retlw   B'00000110'     ;led drive for 1
00E0 345B            0369         retlw   B'01011011'     ;led drive for 2
00E1 344F            0370         retlw   B'01001111'     ;led drive for 3
00E2 3466            0371         retlw   B'01100110'     ;led drive for 4
00E3 346D            0372         retlw   B'01101101'     ;led drive for 5
00E4 347D            0373         retlw   B'01111101'     ;led drive for 6
00E5 3407            0374         retlw   B'00000111'     ;led drive for 7
00E6 347F            0375         retlw   B'01111111'     ;led drive for 8
00E7 3467            0376         retlw   B'01100111'     ;led drive for 9
00E8 3477            0377         retlw   B'01110111'     ;led drive for A
00E9 347C            0378         retlw   B'01111100'     ;led drive for b
00EA 3439            0379         retlw   B'00111001'     ;led drive for C
00EB 345E            0380         retlw   B'01011110'     ;led drive for d
00EC 3479            0381         retlw   B'01111001'     ;led drive for E
00ED 3471            0382         retlw   B'01110001'     ;led drive for F
                     0383
                     0384 ;
                     0385 ;
                     0386 InitAd
00EE 30C0            0387         movlw   B'11000000'     ;internal rc for tad
00EF 0088            0388         movwf   ADCON0          ;       /
                     0389         ;note that adcon1 is set in InitPorts
00F0 0008            0390         return
                     0391 ;
                     0392 SampleAd
00F1 20A8            0393         call    SavePorts
00F2 20F8            0394         call    DoAd                    ;do a ad conversion
                     0395 AdDone
```

```
00F3 1908           0396         btfsc   ADCON0,GO              ;ad done?
00F4 28F3           0397         goto    AdDone                 ;no then loop
00F5 1612           0398         bsf     ADOver                 ;set a/d over flag
00F6 20B5           0399         call    RestorePorts           ;restore ports
00F7 0008           0400         return
                    0401 ;
                    0402 ;
                    0403 DoAd
00F8 0186           0404         clrf    PORT_B                 ;turn off leds
00F9 1683           0405         bsf     STATUS,RP0             ;select pg 1
00FA 300F           0406         movlw   0x0f                   ;make port a hi-Z
00FB 0085           0407         movwf   TRISA                  ;       /
00FC 1283           0408         bcf     STATUS,RP0             ;select pg 0
00FD 1408           0409         bsf     ADCON0,ADON            ;start a/d
00FE 307D           0410         movlw   .125
00FF 2102           0411         call    Wait
0100 1508           0412         bsf     ADCON0,GO              ;start conversion
0101 0008           0413         return
                    0414 ;
                    0415 ;
                    0416 Wait
0102 008C           0417         movwf   TempC                  ;store in temp
                    0418 Next
0103 0B8C           0419         decfsz  TempC
0104 2903           0420         goto    Next
0105 0008           0421         return
                    0422
                    0423 ;
                    0424 ;
0026                0425 count  equ     26
0027                0426 temp   equ     27
                    0427 ;
0020                0428 H_byte  equ    20
0021                0429 L_byte  equ    21
0022                0430 R0      equ    22             ; RAM Assignments
0023                0431 R1      equ    23
0024                0432 R2      equ    24
                    0433 ;
                    0434 ;
0106 1003           0435 B2_BCD  bcf    STATUS,0               ; clear the carry bit
0107 3010           0436         movlw  .16
0108 00A6           0437         movwf  count
0109 01A2           0438         clrf   R0
010A 01A3           0439         clrf   R1
010B 01A4           0440         clrf   R2
010C 0DA1           0441 loop16  rlf    L_byte
010D 0DA0           0442         rlf    H_byte
010E 0DA4           0443         rlf    R2
010F 0DA3           0444         rlf    R1
0110 0DA2           0445         rlf    R0
                    0446 ;
0111 0BA6           0447         decfsz count
0112 2914           0448         goto   adjDEC
0113 3400           0449         RETLW  0
                    0450 ;
0114 3024           0451 adjDEC  movlw  R2
0115 0084           0452         movwf  FSR
0116 211E           0453         call   adjBCD
                    0454 ;
0117 3023           0455         movlw  R1
0118 0084           0456         movwf  FSR
0119 211E           0457         call   adjBCD
                    0458 ;
011A 3022           0459         movlw  R0
011B 0084           0460         movwf  FSR
011C 211E           0461         call   adjBCD
                    0462 ;
011D 290C           0463         goto   loop16
```

```
                    0464 ;
011E 3003           0465 adjBCD  movlw   3
011F 0700           0466         addwf   0,W
0120 00A7           0467         movwf   temp
0121 19A7           0468         btfsc   temp,3          ; test if result > 7
0122 0080           0469         movwf   0
0123 3030           0470         movlw   30
0124 0700           0471         addwf   0,W
0125 00A7           0472         movwf   temp
0126 1BA7           0473         btfsc   temp,7          ; test if result > 7
0127 0080           0474         movwf   0               ; save as MSD
0128 3400           0475         RETLW   0
                    0476 ;
                    0477 ;
                    0478 ;
                    0479 ;
                    0480
                    0481         end
                    0482 ;
                    0483
                    0484
                    0485


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X—XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX—— ————————
0140 : ———————— ———————— ———————— ————————

All other memory blocks unused.


Errors   :    0
Warnings :    0
```

NOTES:

AN591

Apple® Desktop Bus (ADB™)

Author: Rob McCall - WFT Electronics

INTRODUCTION

The purpose of this application note is to introduce a PIC16CXX based ADB interface which can be used as a basis for the development of custom ADB devices. This application note describes the hardware involved, a general purpose ADB protocol handler and an example application task. The example software application supports a single key keyboard to the Macintosh® computer (see Figure 1).

OVERVIEW

ADB licensing from Apple Computer.

Described as a peripheral bus used on almost all Macintoshs (except for the Macintosh 128, 512k, and Plus) for keyboards, mice, etc.

Communication between the ADB task and the application task takes place using several flags. The flags indicate whether there is data received that needs to be sent to the Macintosh, or if data from the Macintosh needs to be sent by the application.

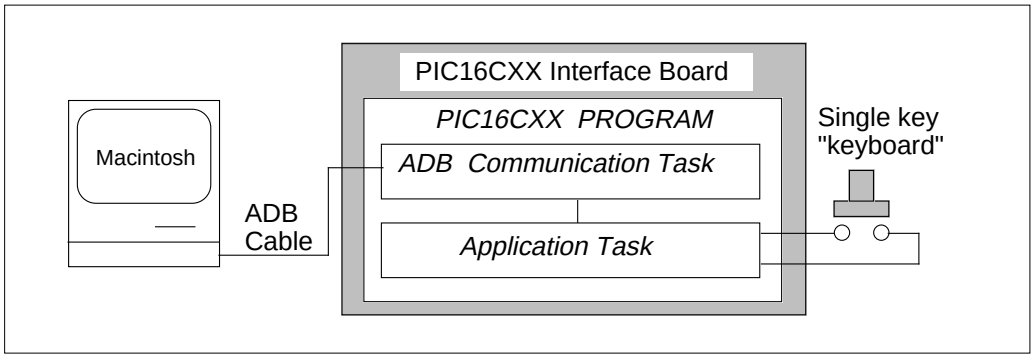
EXPLANATION OF ADB TECHNOLOGY

The ADB is an asynchronous pulse-width communication protocol supporting a limited number of devices. All devices share a single I/O wire in a multi-drop master/slave configuration in which any slave devices may request service. This accomplished through a wired OR negative logic arrangement.

The ADB cable is composed of four wires: +5v, gnd, ADB signal, and power-on (of the Macintosh). The signal wire communicates ADB input and output using an open collector type signal. The number of devices is limited by the addressing scheme and a maximum current draw of 500mA.

Every ADB device has a default address at start-up assigned by Apple. If there are device address conflicts, the protocol supports the reassignment of device addresses at start-up. The software in the PIC16CXX discussed here is designed to easily modify the device address to make the PIC16CXX appear as another ADB device for testing and development.

FIGURE 1: BLOCK DIAGRAM OF FUNCTIONALITY



No device issues commands, except the host. However, devices are permitted to request service during specific time intervals in the signal/Command protocol. A Service Request is referred to as an "**Srq**" The signal protocol communication is accomplished by pulling the ADB line low for various time intervals.

The host controls the flow of data through issuance of specific signal sequences and by issuing several types of Commands. The basic Command types are Talk, Listen, Flush, or Reserved. Each Command has a component called a "Register" indicator which specifies the storage area affected by the Command type. The following is a summary explanation of the each of the Commands. The complete specifications are available from Apple, as listed in the Resources section.

PROTOCOL ASSUMPTIONS

The ADB protocol is defined with a number of general assumptions about its use. These assumptions have driven the general philosophy of the communication sequences. It is assumed that the devices on the ADB are used for human input and each are used one at a time, such as a keyboard and a mouse. It is also assumed that the transfer time from one device to another is relatively slow. This does not mean that the protocol is limited to these assumptions but rather that the protocol is optimized towards this type of use. This is made very evident in the host polling logic, where the host continues to poll the last device communicated with until another device issues an Srq. Consequently, if another device issues an Srq, the device being communication with (or the host) may need to retransmit.

ADB Elements:

The ADB protocol has two components, a signal protocol and a Command/data protocol. These two elements are intertwined. The signal protocol is differentiated in most cases by timing periods during which the ADB signal is low. The Apple ADB specification allows +/- 3% tolerance timing of the signals from the host and +/- 30% by the devices. The signals are:

- Reset: signal low for 3 ms.
- Attention: signal low for 800 µs.
- Sync: signal high for 70 µs.
- Stop-to-Start-Time (**Tlt**): signal high for 160 to 240 µs.
- Service Request (**Srq**): signal low for 300 µs.

After device initialization, in general all communication through the ADB is accomplished through the following event sequence initiated by the host: **Attention** Signal, **Sync** Signal, **Command Packet**, **Tlt** signal, and **Data Packet** transfer. Depending upon the Command, the device may or may not respond with a data packet. Service Requests are issued by the devices during a very specific time at the end the reception of the Command packet.

The Command packets and the Data packets are the constructs used to communicate the digital information. The method of representing data bits is accomplished in a signal timing construct called a **bit cell**. Each **bit cell** is a 100 µs period. Data 1's and 0's are defined by the proportions of the bit cell time period when the line is low and then high. A "1" bit is represented by the line low for 35 µs, and high for 65 µs. Conversely, A "0" bit is represented by the line low for 35 µs, and high for 65 µs (see Figure 3).

FIGURE 2: TYPICAL TRANSACTION WITH COMMAND AND DATA

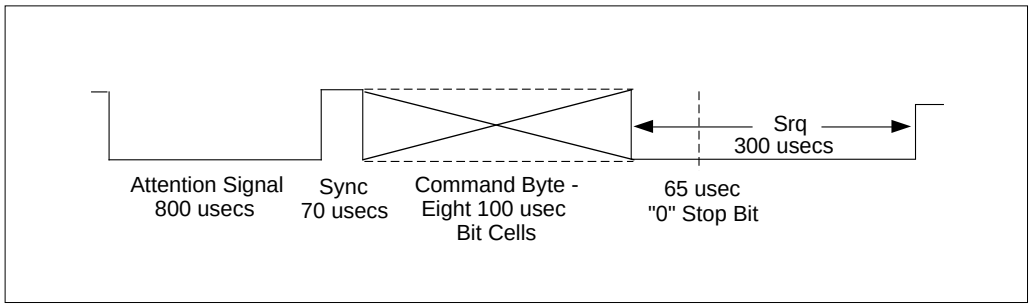
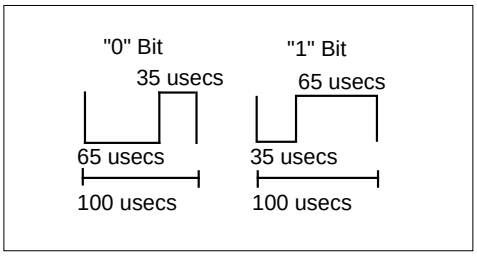


FIGURE 3: BIT CELLS



The **Command Packet**, received from the host, follows an Attention signal and a Sync signal. It consists of an 8 bit Command Byte and a "0" Command Stop Bit. The Command Byte may be broken down into two nibbles. The upper nibble is a four bit unique Device Address. The lower nibble is defined as a Global or Reserved Command for all devices, or a Talk, Listen, or Flush Command for a specific device. Also contained in the lower nibble is a "Register" designator which further details the Command. The importance of the Command Stop bit cell is that **Srq**'s can only be issued by a device to the host during the Command Stop Bit cell low time if the device address is not for the device wishing service, The Host controls when Srq's are allowed through the Command protocol. The Tlt signal and Data Packet transfer, which are part of every Command packet signal sequence, are overridden if an Srq is issued by any device.

A **Data Packet** is the data sent to, or received from, the host. Its length is variable from 2 to 8 bytes. The structure is a "1" Start Bit, followed by 2 to 8 bytes, ending with a "0" Stop Bit. The Apple ADB documentation refers to the data packet sent or requested as Device Data "Registers". This does not necessarily indicate a specific place in memory. In this PIC16CXX implementation, each Data Register has been limited to two PIC16CXX register bytes. The ADB specification allows each Data Register to hold between two and eight bytes. They are referenced in the Command byte as "register" as 0, 1, 2, or 3. Data Register 3 has special significance. It holds the special status information bits (such as whether Srq's are allowed), the Device Address, and the Device Handler ID. Commands are further defined by the "register id" sent in the Command data packet.

For example, if the Host issues the Command in binary of 0010 1100, it would be interpreted as "Device 2, Talk Register 0". The complete definition of the Commands and data registers are described in detail in the ADB specifications supplied by Apple.

PIC16CXX ADB PROTOCOL PROGRAM EVENT SEQUENCE:

Overview

At power-on the host will generate a Reset signal. The purpose of Reset is to initialize the devices on the ADB line. This includes determining the addresses of each device, and resolving device address conflicts if there are any. Once the device addresses are determined, each device waits to be Commanded or issues an Srq if it requires service from the host and is not being addressed by the host. After Reset processing the ADB Protocol Task monitors the ADB line for the Attention/Sync/Command signal sequence. The PIC16CXX program differentiates the signal timing.

Note: The signal detection routines check to see if the Application Task needs service after each event and after the falling edge of the Attention signal is detected.

Command interpretation is accomplished during the low signal time of the Stop Bit cell of the Command packet. Response to the Command must occur after the minimum time of the Stop to Start time period (**Tlt**), which is 160 usec. but before the max Tlt time of 240 usecs. When a device has issued an Srq, it waits to be addressed by the host. If the next Command received is not for the device, it issues the Srq again. The normal response to an Srq will be a Talk Command from the host.

Detailed Description

START-UP

Upon start-up, the Reset routine is executed, looking for the ADB line to be high. When the line is high, an initialization routine is executed during which registers are cleared or loaded with default values. The only exception is a register for generating a random address used in the address conflict resolution process.

RESET

During a Reset condition, default values are loaded, such as the Default Device Address and Handler ID (a piece of information used by the host to identify the type of device). More than one device may have the same address. There is a sequence of events to resolve address conflicts described in the Implementation section. The host assigns a unique address to each device. The Reset condition only takes place once, during start-up, except under unusual conditions, such as testing this program.

ATTENTION ROUTINE

When the Reset routine is complete, the Attention Signal routine is executed, looking for the line to go low and then high. This low time is monitored to be within range of the Attention Signal Timing. If the timing is below the minimum threshold, the routine aborts to start over again looking for the line to go low as the beginning of the Attention Signal. If the low time is exceeded, the routine aborts to the Reset Signal routine.

SYNC SIGNAL ROUTINE

When the line transitions to high, the Sync Signal routine looks for the line to go low as the start of the first bit of the Command Byte. If the Sync high time is exceeded, the routine aborts to the Attention Signal.

COMMAND ROUTINE

The Command routine detects and decodes the next 8 bit cells as the Command Byte. The routine must first determine if the device address is for the device. If the routine determines that the device address in the Command matches the stored device address it may do one of two things; issue an Srq to the host by holding the line low, or go on to check if the Command is Global to all devices. If Global, the routine determines the specific Command and executes the routine for that Global Command. After execution of the Command routine it then goes back to look for the Attention Signal.

When a device is addressed, it determines whether the Command is to Talk, Listen, or Flush data, for the specified Data Register number. If the Command is for Data Register 3, there are special considerations, described for this program in the Implementation section below. If the Command is to Flush, the routine clears the data in the specified Register. The ADB specification defines the action of the Flush Command to be device specific. For a Talk Command or Listen Command, the device then waits for the Tlt signal. When the Command is to Talk, the device sends the data bytes from the specified register and a Data Stop Bit after the Tlt minimum time. For a Listen Command, the device receives data for the specified Register.

When the data has been Flushed, Sent, or Received, the program the device then goes to look for the Attention signal again.

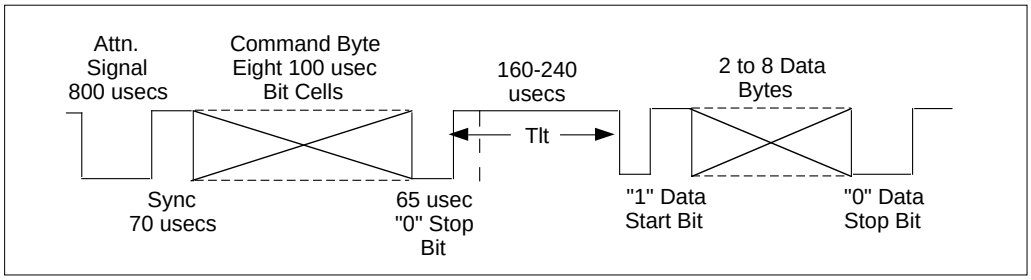
Notes: 1. In this PIC16CXX program, the Application Task is serviced before looking for the Attention signal.

2. If at any time the line is low or high outside of timing ranges, the program aborts to check if an Attention or Reset signal has been issued by the Host. In the case of sending Data, the program goes first to the Collision routine.

SENDING DATA TO THE HOST

Data is sent only in response to a Talk Command. For every data bit cell, the line is tested to go high at the proper time. If the line is still low, a collision has occurred. When a collision is detected, a collision flag is set, and the program aborts to look for a Command signal sequence.

FIGURE 4: TYPICAL TRANSACTION WITH SERVICE REQUEST



IMPLEMENTATION

Hardware:

The hardware of this circuit is fairly simple. The circuit is powered via the +5v and Gnd wires of the ADB cable. The ADB I/O wire is connected to port RA0 with a pull-up resistor to 5v. The T0CKI pin is tied to Gnd. The Master Clear pin is tied to 5v.

This circuit uses a 4Mhz crystal as a timing reference, but higher values may be substituted. The software is designed to accommodate higher frequencies.

A push-button switch is used as the single key of the "keyboard". One side is connected to port RB1 with a pull-up resistor to 5v, and the other side to Gnd. An LED is used to indicate that the 'key' has been pressed, with the positive side connected to port RB0 and the negative side to Gnd.

Software:

There are two sections of the program designated as "Application Tasks", setup to switch between a protocol support task for the ADB signal decode and processing, and another task, the Application Task, in this case a single key "keyboard" routine. The ADB protocol task has priority. The first section of the code is the ADB protocol task, the second section is the Application task, "Keyboard". The two tasks communicate through flags which indicate that data needs to be sent, or that data has been received.

The Keyboard Task is run at two times; 1) during the Attention Signal, 2) between the end of the Data Stop Bit and the beginning of the Attention Signal. The Keyboard Tasks is given up to 500 usecs during the Attention Signal, and 900 usecs during the time between the end of the Data Stop Bit and the beginning of the Attention Signal. It is important to note here that the other tasks MUST NOT AFFECT TMR0 or the ADB time variable that the Attention Signal is using to keep track of the RTCC.

Timing:

Timing of is accomplished by first loading a constant into a time variable. This constant represents the maximum limit for the current routine, which may not necessarily be the maximum timing range for the current Signal. The TMR0 value is loaded into the working register, and subtracted from the time variable. The Carry bit of the Status register is tested to see if it is set or cleared. If the bit is clear, the current timing limit has been exceeded. Further action is taken based on this status. It is important that the constant not be allowed very close to 255, or rollover may occur, giving inaccurate results. The prescaler is applied to the TMR0 as necessary.

The following are the timing ranges used by this program for ADB signals:

Reset	Greater Than 824 usecs
Attention	776-824 usecs
Sync	72 usecs
Bit Cell	Up to 104 usecs
1 Bit low time	< 50 usecs
0 Bit low time	> 50 < 72 usecs
Stop bit	0 Bit
Stop to Start (Tlt)	140-260 usecs
Service Request (Srq)	300 usecs

Note: The range of values given for 0 bit, 1 bit, and Tlt timing are slightly wider than those given in the ADB specification.

How Address Conflicts are Resolved:

During the start-up process the host sends a "Talk Register 3" Command to each device address, and waits for a response. When a device recognizes that the Host issued a "Talk Register 3" Command, it responds by sending a random address. During the transfer of each bit cell of the random address the signal line is monitored for the expected signal level. If the signal is not what is expected there is an address conflict. If the address is sent successfully the host will respond with a Listen Command to that device with a new Device Address for that device to move to. The device then only responds to Commands at the new Address.

If there is a conflict, where two devices have the same default address, and respond at the same time, the device that finds the line low when it expects it to be high, immediately stops transmitting because it has determined that a collision has occurred. The device which detected the collision marks its address as unmovable and therefore ignores the address move Command, a Listen register 3 Command, The device maintains the unmovable address condition until it has executed a successful response to the talk register 3 Command.

The host continues sending a Talk Register 3 Command at the same address until there is a time-out and no device responds. This is how conflicts are resolved when more than one device has the same address; for example, if two keyboards are connected.

Program Sequence:

Words in parenthesis accompanying the TITLES are Labels of procedures corresponding in the code.

Start-up / IDLE (Start)

Start by setting the ADB pin on Port A and the Switch Pin on Port B as inputs, and tri-stating the rest of Port A and B as outputs.

INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH (Reset)

Look for the line to be high, and when it is, clear or initialize registers to default values.

LOOK FOR ATTENTION OR RESET (AttnSig)

Look for the line to go low, when it does, clear TMR0 and time how long it is low. An Attention Signal has occurred when the line goes high between 776 and 824 usecs. If the low time is measured less than 776 usecs, another signal has occurred and the program aborts, looking for the Attention Signal again. When the low time is measured greater than 824 usecs, the program interprets this timing as a Reset Signal. The program starts over again, waiting for the line to be high, and when it is, performs a Reset initialization.

Note: The keyboard task is performed during the attention signal (Task_2).

LOOK FOR SYNC SIGNAL (SyncSig; calls Srq)

The Sync Signal is the high time between the rising edge of the Attention Signal and the falling edge of the first bit of the Command.

GET THE COMMAND (Command; calls Get_Bit)

Look for the Command; a combination of eight 0 and 1 bits. The MSB is sent first. This is achieved by calling a the Get_Bit routine, which checks whether the maximum Bit Cell time is exceeded, if not, it looks for the rising edge at the end of the bit. When the bit is received, it is rotated into a variable, and the end of the bit cell is expected. When the falling edge of the next bit is detected, the routine clears TMR0 and returns to Command, which calls Get_Bit again until all 8-bits of the Command have been received.

ISSUE A SERVICE REQUEST IF NECESSARY (Srq)

If data needs to be sent to the Host, a Service Request (Srq) is issued by holding the line low while the Stop Bit is being received, during the Stop-to-Start time (Tlt) between the end of the Command Stop bit and the beginning of the Data Start Bit

LOOK FOR STOP BIT (CmdStop)

Look for the Stop Bit (a 0 bit of 65 usecs) that comes after the last Command Byte.

INTERPRET THE COMMAND (AddrChk)

After the Command has been received, determine if the Address belongs to this Device. If the Address is not for this Device, determine if the Command is global for all Devices and if so, do that Command. If this is not a Global/Reserved Command, call the Service Request (Srq) routine to see if an Srq should to be issued to the Host, and do so if necessary, then return to get the Attn Signal. If the Address is for this Device determine whether it is a Talk, Listen, or Flush Command, and go to the specified Command routine.

SENDING DATA (Talk; calls Tlt)

If the Command was interpreted to be a Talk Command addressed to this Device, call the Stop-to-Start Time (Tlt) routine. When the Tlt routine has completed, determine if this is a Talk Register 3 Command. If so, return a Random Address as part of the two bytes sent to the Host. If this is not a Talk Register 3 Command, determine if Data needs to be sent. If so, send the Data Start Bit (a '1'), two bytes of Data from the indicated register, and a Stop Bit (a '0'). If not, abort to the Attention Signal. If at any time the transmission of Data is interrupted, abort to the Collision routine. Only after a complete transmission should the flags be cleared indicating a successful transmission.

Note: The ADB Specification indicates data may be between two and eight bytes long. The limitations of the PIC16C54/55/56 parts allow only two bytes of data to be sent by this program due to limited register space. If more than two bytes of data must be sent, use the PIC16C57.

RECEIVING DATA (Listen; calls Tlt)

If the Command was interpreted to be a Listen Command addressed to this Device, call the Stop-to-Start Time (Tlt) routine. When the Tlt routine has completed, receive the rest of the Data Start Bit, 2 Data Bytes, and Data Stop Bit. When the Data has been received, determine whether this is a Listen Register 3 Command. If this is a Listen Register 3 Command, interpret what the Command is. If this is a conditional Address change Command, determine if this Device's Address is moveable at this time. If not, abort to the Attention Signal. If so, change the Device to the new Address and go run the Second Application Task. If this is not a Listen Register 3 Command, move the Data into the specified register and go run the Second Application Task.

LOOK FOR THE STOP TO START TIME (Tlt)

After the Command and Stop Bit, the Talk or Listen routines call the Tlt routine. Tlt looks for the line to go low. If the line went low before the Min. Tlt Time, see if this is a Talk Command. If this is a Talk Command, abort to the Collision routine. If this is a Listen Command, abort to the Attention Signal.

If the Min. Tlt time passes and the line is high, see if the Talk routine called the Tlt, if so, go wait for until the middle of the Tlt, then return to the Talk routine to send the Data Start Bit, Data Bytes, and Stop Bit. If at any time the line goes low during the Tlt and the Talk routine called it, abort to the Collision routine.

If the Listen routine called Tlt, look for the line to go low as the beginning of the Data Start Bit. When the line goes low, return for the rest of the Start Bit. If the line doesn't go low before the Max. Tlt time is up, abort to the Attention Signal.

THE KEYBOARD TASK IS PERFORMED BETWEEN THE END OF THE DATA STOP BIT AND THE ATTENTION SIGNAL (Task_2)

The Keyboard Task checks to see if the key has been pressed. When the key is pressed, flags are set to indicate this and an LED is turned on until the key has been debounced. The flags allow the key to be debounced, Srq(s) to be sent to the Host, and indicate to the Talk routine that Data needs to be sent. Two bytes of data are loaded into Register 0 representing a key-down code and a flag is set indicating to the ADB task that data needs be sent to the host. When the key-down codes have been sent, the key up codes are loaded into Register 0. When the key-up codes have been sent and the key has been debounced, the flags are cleared. The final routine of Task_2 decides whether to return to the beginning or middle of the Attention Signal.

FIGURE 5: APPLE DESKTOP BUS PIC16CXX FLOWCHART

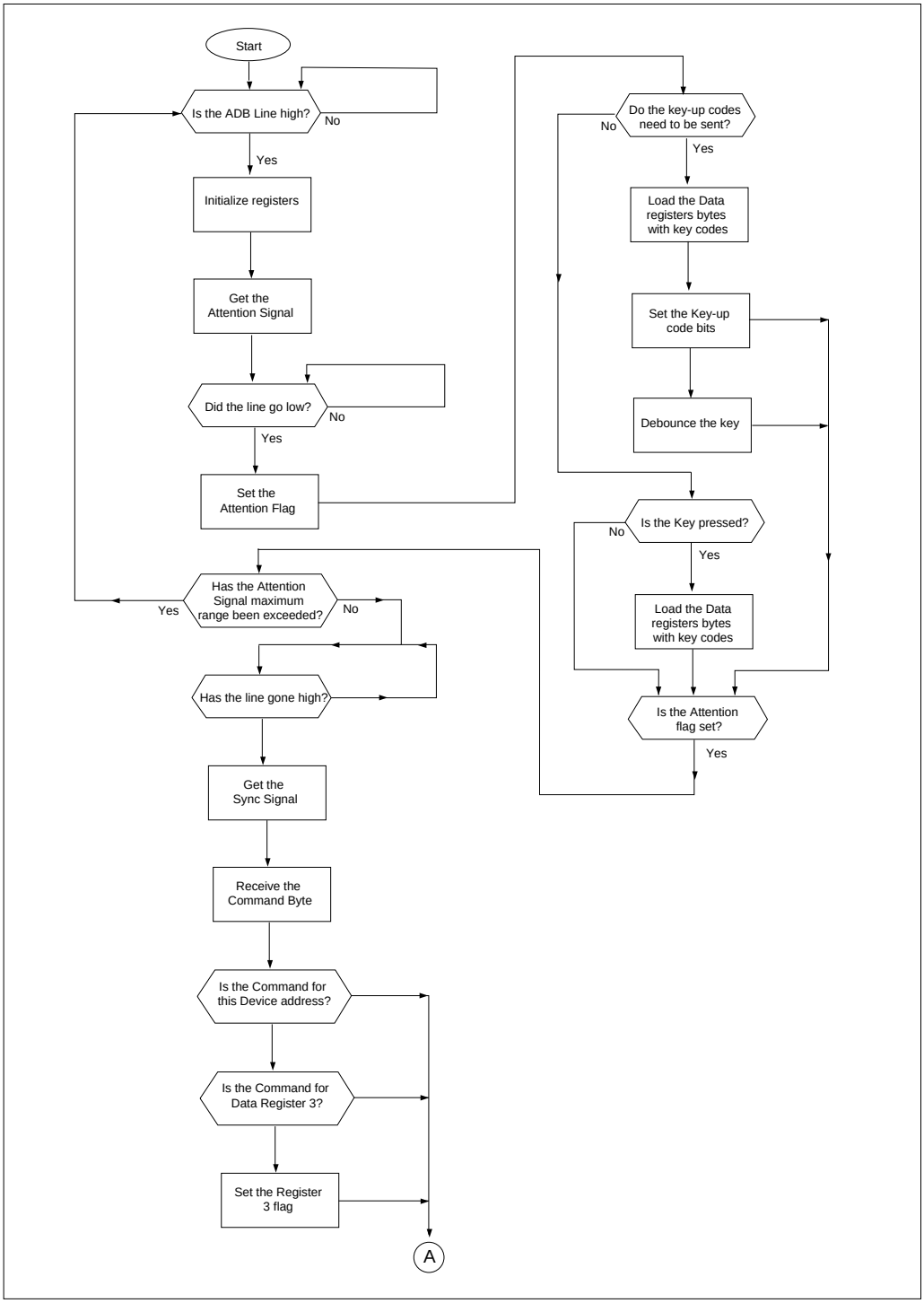
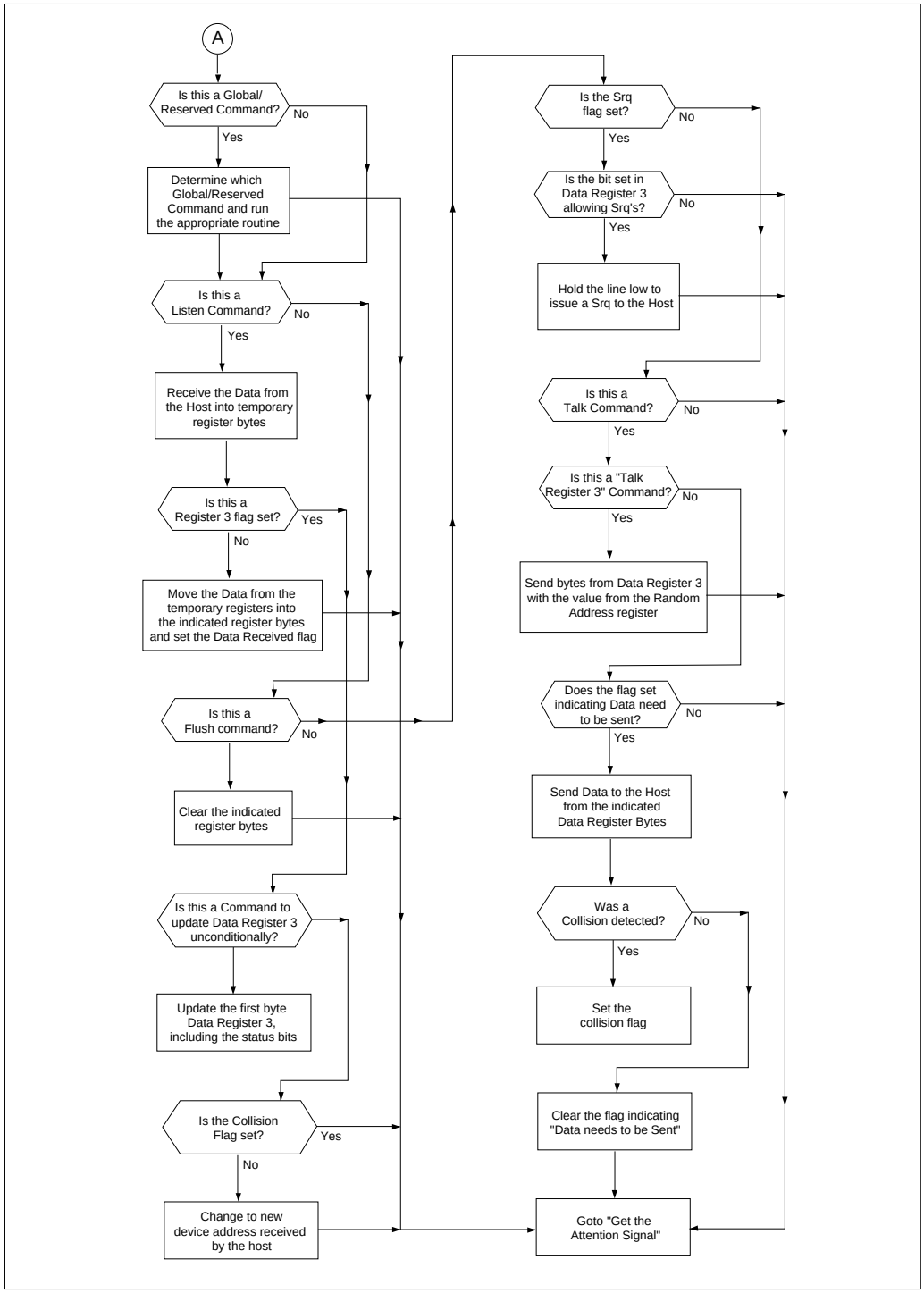


FIGURE 5 (CONT.): APPLE DESKTOP BUS PIC16CXX FLOWCHART

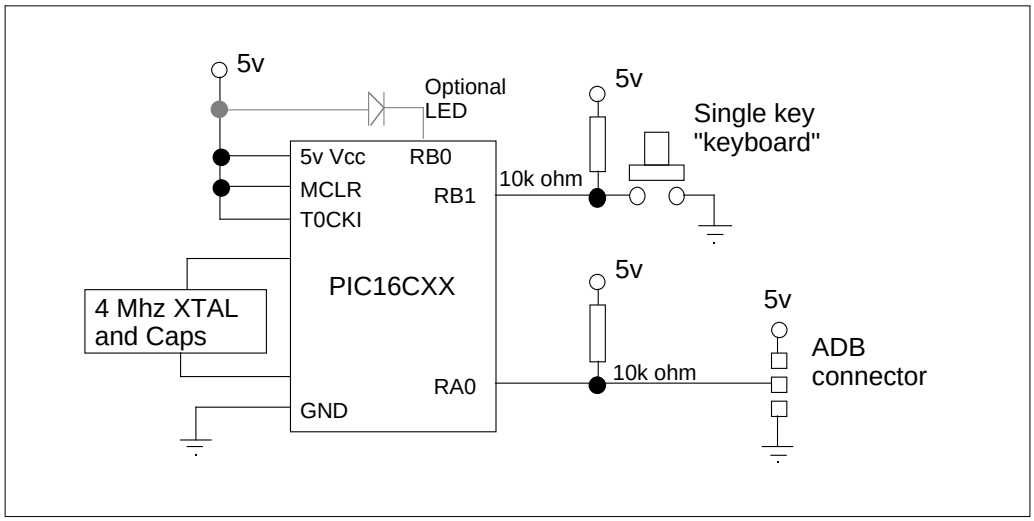


SUGGESTIONS ABOUT MODIFYING THE CODE

1) If high crystal frequencies are used, a divider equate at the beginning of the timing section of the equates allows an easy adaptation for all established timing definitions.

2) The second application task may occur as a communication task with another PIC16CXX chip by using the three other i/o lines on Port A, although code for this has not yet been written to test this. Two of the lines would be used as ready-to-send (one for each PIC16CXX). The third would be used as a data line, using low signals as 0 bits, and high signals as 1 bits. Additionally, all eight lines on Port B may be used as well.

FIGURE 6: SIMPLE SCHEMATIC OF THE TEST BOARD



RESOURCES

Apple Publications and Support Software

MacTech Magazine (formerly MacTutor) is a publication dedicated to supporting the Macintosh. They have had several articles regarding the Apple Desktop Bus. They publish a CD-ROM that contains all of their articles from 1984 to 1992. Also, single disks are available (ask for #42).

MacTech Magazine can be contacted at:

P.O. Box 250055
Los Angeles, CA 90025-9555
310 575-4343 FAX 310 575-0925
Applelink: MACTECHMAG
Internet: info@xplain.com

Apple licenses the ADB technology. They can be contacted at:

20525 Mariani Ave.
Cupertino, CA 95014
Attn: Software Licensing

- Apple Keyboard, extended, specification drawing #062-0168-A.
- Apple Desktop specification drawing # 062-0267-E.
- Apple Desktop connector, plug, Mini DIN drawing #519-032X-A.
- Engineering Specification, Macintosh transceiver interface, ADB drawing #062-2012-A.
- Apple keyboard, specification drawing #062-0169-A.
- Developer CD series, Tool Chest Edition, August 1993 contains:
    - Folder = Tool Chest: Devices and Hardware: Apple Desktop Bus
    - ADB Analyzer
    - ADB Parser (most complete environment)
    - ADB Lister
    - ADB ReInit
    - ADB Tablet code samples

WFT Electronics offers free assistance in procuring necessary ADB info. Contact Gus Calabrese, Rob McCall, Dave Evink at:

4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.orgA

AUTHOR / CREDITS

Rob McCall developed the majority of the PIC16CXX ADB code. He also wrote most of the application note. Gus Calabrese, Dave Evink, and Curt Apperson supported this effort. Dave works with Gus, Rob, and Curt in developing a variety of embedded processor products.

Contact Gus Calabrese, Rob McCall, Dave Evink, Curt Apperson at:

WFT Electronics
4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.org

3

NOTES: