MICROCHIP

AN585

A Real-Time Operating System for PICmicro™ Microcontrollers

Author: Jerry Farmer
Myriad Development Company

INTRODUCTION

Ever dream of having a Real-Time Kernel for the PIC16CXXX family of microcontrollers? Or ever wonder what Multitasking or Threads are all about? Then this article is for you. We will explore how to implement all of the features of a large Real-Time Multitasking Kernel in much less space, with more control, and without the large overhead of existing kernels. By planning ahead, and using the techniques outlined here, you can build your own fast, light, powerful, flexible real-time kernel with just the features needed to get the job done.

Included in this article are two large examples: one on the PIC16C54, and the other on the more powerful PIC16C64. A "Remote Alarm" is implemented on the PIC16C54 as an example of a Non-Preemptive Kernel, with two asynchronous serial input sources capable of running up to 19,200 Baud along with seven sensors needing to be debounced as inputs. One more input line is monitored and causes an internal software recount. For output, this example has an LED that shows eight different internal states of the "Remote Alarm", blinking at different rates and different sequences. Last but not least, is an asynchronous serial output capable of running at 38,400 Baud, passing the inputs to the next remote alarm station. Several short and long timers are included to round out the nine cooperating tasks in this example. Please refer to Figure 2 and Appendix B.

The second example is implemented on an PIC16C64 featuring an interrupt driven Semi-Preemptive Kernel. This example has the serial input and output routines of the first example moved into Interrupt Service Routines (ISR) for more speed and accuracy. The interrupt capabilities of the PIC16C64 will be explored, and a Real-Time Multitasking Kernel framework will be developed. Please refer to Figure 5 and Appendix C.

Why do I Need a Real-Time Kernel?

Real-time design techniques allow the engineer/designer to break-up large, complicated problems into smaller simpler tasks or threads. These more manageable units of code allow faster response to important events, while prioritizing the jobs to be done in a structured well-tested format. The kernel does the job of keeping the time, the peace between tasks, and keeping all the tasks' communication flowing. More activities can be performed in the same amount of time by allowing other tasks to work while other tasks are waiting for some event to occur. Smaller code is also the result of using State-Driven techniques because much information is condensed into the state variables and code structure. If you need an example, look at the PIC16C54's "Remote Alarm" code.

What is Multitasking Anyway?

This is the appearance of several tasks working at the same time. Each task thinks that it owns the CPU, but this appearance is controlled by the kernel. Only one task can be running at a time, but there is undone work that can be done by other tasks not blocked. Multitasking is the orchestration of interrupts, events, communication, shared data, and timing to get a job done. Real-Time Programming is just a bunch of ideas, concepts, and techniques that allow us to divide problems into units of code that are based on units of time, or events that drive a task from one state to another.

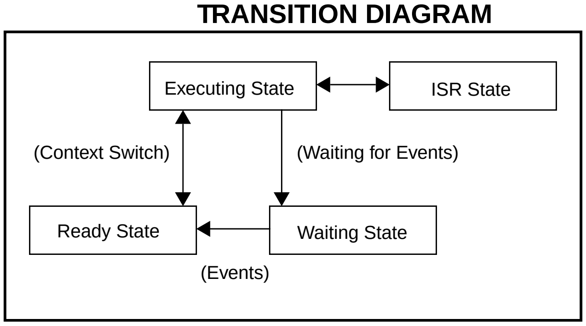
CONCEPTS

We will cover the basic concepts of kernels here so that we are using the same definitions when talking about this difficult topic. This article is a very quick survey on Real-Time Kernel concepts. I hope to get you thinking, reading more, and hopefully writing RT Operating Systems for your current and future projects. Many great books have been written about this very broad and interesting subject. We will refer to some of these books which have a different point of view other than those expressed in this paper.

Critical Section

A critical section is a shared data structure, or a shared resource, or a critical time section of code, or a non-reentrant section of code that can have only one owner that is allowed to view/change/use that section at any one time. These sections must not be interrupted during the update process. They must be protected so that other tasks can not get in and change the pointers/data or modify the hardware at the same time. Remember that if two tasks can get into a critical section, at the same time, then data WILL be corrupted. Make sure that critical sections are small, with time for pending interrupts to get serviced. Not understanding critical sections is where the beginning RT programmers get into the most trouble. Even without interrupts, you must protect variables that are changing over time, such as the byte sized variable `xmt_byte` used in the PIC16C54 example. This variable changes each time the STATE changes for the Serial Out Task. Semaphores, and Disabling Interrupts are two of the techniques used to coordinate between different tasks wanting to control a critical section. Task #4 is devoted to the proper feeding of the shared Serial Out Resource in the PIC16C54 example. Note the use of the binary semaphore “OState_B” to control Task #4, Task #1, and the variable `xmt_byte`. There are several more examples of critical sections in the PIC16C64 example due to the use of interrupts. We disable interrupts for very short time periods to protect these areas. Also in the PIC16C64 example, all critical sections are finished before checking to see if the kernel wants another task to start running instead of the current task. We will discuss in more detail how to protect critical sections later in this article.

FIGURE 1: TASK / PROCESS STATE TRANSITION DIAGRAM



Shared Resources

Data structures, displays, I/O hardware, and non-reentrant routines are a few resource examples. If two or more tasks use these resources, then they are called Shared Resources and you must protect them from being corrupted. They must have only one owner, a way of telling others to wait, and possibly a waiting list for future users of that resource. A rare example of a shared resource is when there exists a critical timing sequence of input and output operations to control some hardware. You must disable interrupts before starting this sequence, and re-enable them upon finishing. Note that Task #1 in the PIC16C64 example is an example of an “non-reentrant” routine that must be finished by the current owner before another task can use it.

Context Switch/Task Switch

When one task takes over from another, the current values of the CPU registers for this running task are saved and the old saved registers for the new task are restored. The new task continues where it left off. This is all done by the Context Switch part of the Real-Time Kernel. Each task usually has a “context switch storage area”. Each task’s SP (Stack Pointer pointing into its own stack) is stored there along with all the other important saved registers. The “Remote Alarm” example does not need to use a context switch because all the important registers are properly freed-up before each task is finished. The PIC16C64 example uses a similar concept, thus keeping the number of saved registers per task way down. We use an old concept called “where I came from”. The variable “FROM” is used to direct the dispatcher to start up the task where it left off. This is because you cannot manipulate the stack in the PIC16CXXX family. This same reason is why we have a “Semi-Preemptive” kernel on the PIC16C64 as an example. By the way, the faster the context switch is done, the better.

Scheduler

The scheduler is that part of the kernel that decides which task will run next. We will talk about several common types in this section. This is where a lot of thinking should be done before starting your new project. By understanding the different kinds of schedulers and what features and problems each type has, you can match your problem to a creatively styled scheduler that meets your needs. For example, the PIC16C54 example shows the recalling of Tasks #1-3 just before a long sequence of code is executed. More creative ways can also be implemented, but be careful to allow all tasks to execute in a timely fashion.

Please see Figure 1. Each task must be in “Ready State" or the "Executing State" to be considered by the scheduler to get temporary control of the CPU next.

Non-Preemptive Kernel

The Non-Preemptive Kernel is also called a “Cooperative Kernel” because the tasks only give-up control when they want/need to in coordination with other tasks, and events. The “Remote Alarm” example uses a Non-Preemptive Kernel type, showing that despite its reputation as being a simple kernel type, a lot can be done with it. The Non-Preemptive Kernel type is well suited for the non-interrupt type PIC16C5Xs. The heart beat of the PIC16C54 example is the internal TMR0 counter crossing over from a high value to a low value of the counter. Use the prescaler to adjust the time units. The very fast tasks continually read the TMR0 directly comparing the delta of time to see if it should fire.

Preemptive Kernel

In a Preemptive Kernel, a running task can be swapped out for a higher priority task when it becomes ready. The Preemptive Kernel relies much more on interrupts as its driving force. The context switch is at the heart of this type of kernel. To implement a true Preemptive Kernel, you must be able to manipulate the stack. This is why we implemented a “Semi-Preemptive” kernel on the PIC16C64, with some of the best features of both types of kernels. We moved some of the tasks in the PIC16C54 example into ISRs to handle the I/Os. This works very well as the ISRs are very short and do most of the real work in this example. The TIMER0 interrupt is the heart beat for the PIC16C64 example. You must have a clock interrupt in order to make a true Preemptive kernel.

Round Robin Scheduler

When the scheduler finds tasks on the ready queue that have the same priorities, the scheduler often uses a technique called Round Robin scheduling to make sure each task gets its day in the sun. This means more housekeeping to get it right. This is part of the creative ways you can tailor the scheduler to fit your needs. In the PIC16C54 example, all tasks will get to run shortly after their appointed time. This means that no task will dominate all others in this simple approach. In the “olden” days of the first Real-Time Operating Systems the term was used to mean the same as “time slicing”. The Preemptive Kernels of today are a major step forward, with their priority schemes, and intertask communication capabilities.

Preemptive vs. Non-Preemptive

The Preemptive Kernel is harder to develop, but is easier to use, and is sometimes used incorrectly. You must spend more upfront time with the Non-Preemptive Kernel but it is better for more cramped microcontrollers. You get much better response time between a cause/event and the response/action for that event with a Non-Preemptive Kernel. The Preemptive Kernel is more predictable in the response times, and can be calculated as to the maximum time to complete a given job. Often the Preemptive Kernel is more expensive.

Reentrancy

In a Preemptive Kernel, two or more tasks may want to use the same subroutine. The problem is that you can not control when a task is swapped out and when another takes its place. Thus, if a subroutine uses only local or passed variables that are stored only in each tasks’ stack, then it is call reentrant or a pure routine. No global variables or hardware may be used in such a pure routine. A way around this reentrancy requirement is to treat the whole subroutine as a critical section.

Appendix D is an example of reentrant code segment as might have been used in the PIC16C54 code example.

Task Priority

Some tasks are not created equal. Some jobs must be done on time or data will be lost. Make the tasks that must get done the highest priority and go down the scale from there. Some kernels make you have a different priority for each task. This is a good idea and requires some thought before coding to make the design work.

Static vs. Dynamic Priorities and Priority Inversions

For most embedded Real-Time Kernels, both static priorities and static tasks are used. Dynamic priorities are sometimes used to solve deadlock and other complex situations that arise from not understanding the problem and not understanding Real-Time Techniques. If the need for dynamic priorities seem to occur, you should relook at how you divided the problem, and divide less so as to include the resources in question under one semaphore. You could divide the problem more to have more tasks not needing two or more resources to complete its job, and have the new tasks talk more together.

As for Dynamic tasks, you should define the problem so as to know, ahead of coding, the continuous use of all tasks. You will need more upfront time in the planning stage to get all tasks talking, but it is well worth it to keep Dynamic Priorities and Dynamic Tasking out of the kernel design.

Priority Inversions is a trick used to get past a poorly designed system by inverting the priorities to allow lower tasks to run that were previously blocked. This is a cheap trick, and is best kept out of a Real-Time Kernel. Use the other techniques outlined in this section to solve this kind of problem.

Semaphores

There are basically two types: binary and counting semaphores. The binary semaphore allows only one owner, and all other tasks wanting access are made to wait. The counting semaphore keeps a list of users that need access. Semaphores can be used in many ways. We will illustrate most of them in the following paragraphs. Note that you can implement counting semaphores using binary semaphores.

Mutual Exclusion

We have touched on the subject of Mutual Exclusion earlier (a method to exclude other tasks from gaining access to critical sections). Mutual Exclusion is the process of excluding others from access to the shared resources. To make a semaphore is a very complicated process. The semaphore’s construction must be atomic. That means that once the process has started, it can not be interrupted until it has saved the name of the new owner. From there on, it knows that no one else can break-in and change owners. We have implemented a binary semaphore using bits and kernel functions to mutually exclude access in the PIC16C54 example.

In the PIC16C64 example, we also disable interrupts to get the same effect. There are at least two good ways of implementing a binary semaphore. The first and oldest way was discovered by a Dutch mathematician named Dekker. We will refer you to a book that talks more about this algorithm. The second method of implementing a binary semaphore was also discovered by another Dutchman named Dijkstra. This method uses the “testandset” type instruction and is much more important to us. We used the `dec & jump if not zero` instruction (see PIC16C64 example).

Deadlock

Deadlock is a condition where two or more tasks own resources that other tasks need to complete there assignment but will not release their own resources until the other tasks release theirs. Talk about cooperation. Please read section, "Static vs. Dynamic Priorities and Priority Inversions" for a discussion about such problems and ways to solve them. The root of such problems is not understanding the original problem.

Synchronization

Semaphores can be used to synchronize tasks so that messages can be passed between them. Also tasks can be started up by semaphores, stopped by semaphores, or started together. They are the foundation blocks for Real-Time Programming. Once you have built a binary semaphore for your kernel, you can build very complex semaphores to synchronize anything. In the PIC16C54 example, data from several sources are passed out the Serial Port Resource. Task #4 synchronizes the other tasks trying to send data out and synchronizes with task #1 to get it done. When task #1 is running, then task #4 can not run until task #1 is ready for more data to send out.

Intertask Communication

We have touched on this topic already, but for large kernels, one can include more complex communication methods to pass data/messages between tasks. Much of the handshaking is done for you inside the kernel. This takes a lot more space and execution speed to implement them in a kernel.

Event Flags

We implemented Event Flags as simple bits having two states (on and off). More info can be stored per Event Flag such as time it was recorded, by who, and who the event belongs to, and whether data was lost.

Message Mailboxes

This is a nice feature to have if you have the ram space. Mailboxes allow the designer to pass messages between tasks, and allows messages to be looked at when the task is ready, and to reply telling the sender that the message was received. One message can be sent to many tasks at the same time.

Message Queues

This again is a very nice feature if you have the execution time, and the ram to implement them. This feature is related to Mailboxes, in that you can store several messages even after reading, to be processed later. If you want to only operate on the highest prioritized messages before handling the rest, this is allowed. You can be very fancy with the Mailboxes and Queues. If you have them, use them.

Interrupts

Interrupts are one of the best inventions to come along for solving Real-Time problems. You can get very quick response to the need, and then go back to what you were doing. The only problem is that they can and do happen at the worst times. That means that you must learn how to turn them on and off to protect your critical sections. Note that before an interrupt can be handled, you must save all important registers so that you can restore them so that the kernel can restart the task where it left off. This is much like the context switch issue, but for interrupts, you must always save and restore. In the PIC16C64 example, the Status, W, and FSR registers are saved in RAM because of the interrupt. The PC register is saved onto the stack by hardware.

Interrupt Latency, Response and Recovery

Interrupt Latency is defined as the largest time period that interrupts are disabled, plus the time it takes for the ISR to start to execute.

The Interrupt Response Time is defined for a Non-Preemptive system as Interrupt Latency plus the “context saving time.” For a Preemptive system, add the execution time for the kernel to record the interrupt.

Interrupt Recovery Time for a Non-Preemptive system is defined as the time to restore the saved context and for the restarting of the task that was interrupted. Interrupt Recovery Time for a Preemptive system is the same as for the Non-Preemptive system plus the time the kernel takes in the scheduler deciding which task to run next. These measurements are how most kernels are compared with each other. The PIC16C64 example does very well in these measurements. That is because of the PIC16CXXX processor and that they are mostly a Non-Preemptive system. You must keep the time you disable interrupts to a minimum in any kernel you write or any task that you write. You should break-up long sequences of instructions to allow for interrupts that are already waiting to execute.

ISR Processing Time

ISR (Interrupt Service Routine) Processing Time is defined as the time an ISR keeps control of the CPU. This amount of time should be short, and if a lot of processing needs to be done in a ISR, then break up the ISR. The new ISR should now just store the new data and return. Next, create a new task and move the extra code from the old ISR into the new task. Remember that the longer you are in one interrupt, the longer you can not answer another pressing interrupt.

Nesting interrupts are where the interrupt with a higher priority can interrupt a lower priority interrupt. Care must be used, as different interrupts may have critical sections too, and disabling interrupts must be used here too to protect critical sections. Nesting of interrupts may not exist on all microcontrollers, such as the PIC16CXXX family.

Non-Maskable Interrupts

On some microprocessors, you can enable/disable selected interrupts, such as on the PICmicro family. This is a great tool to control the flow of data into the system and out. Some systems have what is called Non-Maskable Interrupts. Here you can not turn them off by software masking. These NMIs as they are call for short, are used as clock Ticks, because you do not want problems with complex critical sections on a interrupt that you can not turn off. The PIC16CXXX family does not have any NMIs. NMIs are not as useful as maskable interrupts.

Clock Tick

The Clock Tick, is the heart beat of the system. This is how the kernel keeps time (relative & absolute). This is how the kernel is restarted to see if there is a delay that has finished, so that the task can be moved into the ready state. In the PIC16C54 example, the Timer0 clock is used. In the PIC16C64 example, Timer0 is used. You must have a clock interrupt in order to make a true Preemptive kernel. This is the other reason why we implemented a Non-Preemptive Kernel on the PIC16C54 - no clock interrupt.

ANALYSIS OF CODE EXAMPLES

These sections are the real meat of this article. In these sections we will explain how the concepts are put to practical use line by line in each of the two main examples - PIC16C54 (Appendix C) and PIC16C64 (Appendix D).

We will also examine a short reentrant code example in Appendix B. We will give some ideas on how to expand the examples and how far and how fast the examples can be pushed. Be sure to read both sections on the two examples.

The “Remote Alarm” application has many interesting features. The concept is to have as many tiers of units like a tree feeding into the lower level units the status of each of the larger branches to one central point. Each unit can detect any changes in status before the intruder shuts that unit down, or tampers with it. If any unit’s power or wires connecting it down the tree are cut, the lack of the flow of status and passwords would be noticed in five seconds and reported down the line. The two Serial Input lines per unit receive the status and passwords from it’s two larger branches, checking the data and passing the info down the line by its own Serial Output line. The seven input status lines are debounced in these examples, showing the technique.

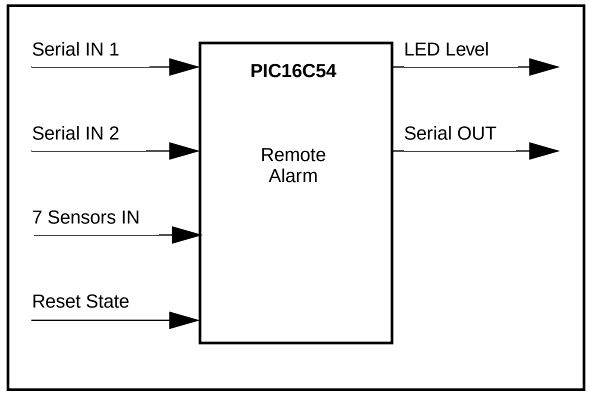
The LED on each unit reports the status at that node as to the importance of its own seven input status lines and the status flowing down the line. The level indication outputted on the LED continues at the last highest level until either a reset is received on the “Reset State” line or five minutes of no new activity on the seven input status lines are received. When either of these two events occur, the level of the LED output is adjusted to the current level of input. Some of the features are changed for this article (Figure 2 and Figure 5).

Another Embedded System use of this type of “Remote Alarm” application is that of placing the unit on the outside of a safe. Hopefully the intruder would be detected before arriving at the unit itself. The continuous stream of status and passwords to the larger unit inside would slow down any simple theft.

PIC16C54 - “Remote Alarm” Example

This example is a cross between a true application and an attempt to show new concepts and some extra features for show. Some of the application specific code has been removed to show more clearly the possibilities of a Real-time Operating System on the PICmicro family. We chose the Baud rate for the Serial output to be twice the speed of the two Serial inputs because it is harder to accurately output a precise Serial Output than it is to monitor Serial inputs.

FIGURE 2: REMOTE ALARM-PIC16C54 EXAMPLE



This example operates at 4 Mhz. By simply increasing the crystal speed to 8 MHz, the two Asynchronous input Serial Baud rates increase from 4800 Baud to 9600 Baud. The Serial Output Baud rate increases from 9600 Baud to 19,200 Baud. By increasing the crystal speed to 16 MHz, it will increase the Baud rates to 19,200 Baud for the two independent Asynchronous inputs, and increase the baud rate for the Asynchronous Serial output to 38,400 Baud. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds. Note that you must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C54 code example. Lines 1-85 are the equates for this program. Lines 88-95 are simple jump tables so as to save some of the precious “first 256 bytes” of each page. The Serial Output Routines - Task #1 are in lines 97-159. Task #7’s subroutines start at line 160 and continue to line 277. In this section, the LED output is controlled. The subroutine QCheck_T123, lines 278-301, is used to allow the checking of Tasks #1-3 to see if they are ready to execute before a long section of code in a slower Task is about to be executed. This is a creative way for the Kernel’s Scheduler to makes sure that the highest Prioritized Tasks get serviced before the less important tasks get executed. Task #2 starts at line 302. This task reads the Serial Input #1 for Asynchronous data. Task #2 can be described as a State Machine for outputting a byte Serially. Task #3 interrupts the code of Task #2 at line 333 and continues until line 362. Task #3 also reads the Serial Input but on input #2. Task #2’s subroutines continue at line 363 and continue until line 423. Task #3’s subroutines continue at line 424 and continue until line 484 is reached. The main or starting code is started at line 485. From that line to line 515, all variables are initialized, and all tasks are initialized at this time also. The Main Loop is started at line 516 and ends at line 665. This is where the real action is done. Each task checks the time to see if the conditions are correct for it to run. The tasks that are not Blocked, and have a job to do now are in a Ready State. In the Main Loop, we check the current state of each task in order of Priority (1-9). If ready, we do a very simple Task

Switch and place that task in the Executing State/Running State. Several time unit changes take place in the Main Loop. Tasks #1-4 use 2 µs as a time base by reading the TMR0 directly. A time unit change takes place at lines 562-575 to 512 µs per unit for Tasks #5-6. Another time unit change takes place for Tasks #7-9, to 131072 µs per unit. For Tasks #5-9, each task counts the time units and compares them to their standard for activation or activity. Task #4 starts at line 538 and finishes at line 561. Task #4 controls the feeding of Task #1 from several other tasks that want data to be outputted. It uses several Semaphores to make sure that Task #1 is not bothered until it is ready for another byte. Task #5 monitors the Level Reset Line, and is always running. It simply resets the status of the LED, to be recalculated in Task #6. Task #5 runs through lines 576-581, and is very short. Lines 582-611 represent Task #6. Here we debounce the seven sensor input lines, leaving the current standard in the variable “Old_RB”. Task #6 asks/Signals Task #4 to output the current standard out the Serial pin. Task #7’s main code is lines 621-628. Task #8 is a five second lack of activity timer, and exists in lines 629-645. If no data comes from either of the two input Serial lines, then Task #8 Signals Task #4 to send a special byte to be outputted by Task #1. This Signals the next “Remote Alarm” of the lack of communication between units. The last task is Task #9. This is a five minute lack of Severe Errors the from Sensor Reset Timer. Lines 646-663 compose Task #9. Subroutine `Do_D_H_E_L` starts at line 667 and continues through to line 692. This routine determines the Highest Error Level, and passes Task #7, the current state, to output on the LED. Lines 693-703, clear the registers #7-1Fh. The “jump at Power-On” code is the last lines 705-706.

The following sections describe in more detail how and what each part of the code does and why. The code segment lines 1-87 are explained in this paragraph. Line 4 tells the MPASM assembler which PICmicro you are using. The include file `PICREG.H` follows with the equates and assignments to make the code more readable and changeable. You should use equates that relate symbols to each other. The Constants — lines 10-12 are the values to change for different Baud rates. They represent the Bit Times for the Baud rates divided by 2 minus some latency factor. You might have to adjust the “Fudge Factor” and other values to fine tune the performance. The value used for the “Fudge Factor” is related to the longest path of code. Lines 21-24 are an experiment that allows a simple name to be associated to a single bit. This allows for easily changeable assignments. Lines 30-54 are the variable assignments. Variables (lines 35-39) are used as time counters. They count the number of units of time, and are compared to literals to see if an Event has just happened. The bits defined in lines 57-64 are used as Binary Semaphores. They keep Critical Sections of data protected. We will see them in action later in the code. The bits defined in lines 67-73 are error flags. They define the current or last error states of the Serial routines, and whether data was lost coming in or out.

The section of equates in lines 76-85 are used to define the different LED activity. They are used by Task #7 to keep the LED blinking. In lines 89-94, we try to save the all important first 256 bytes of any page.

Task #1 outputs a byte Asynchronously over the Serial Output pin. Task #1 is started at line 98. The time units used for Tasks #1-4 are 2µS. We first sample the TMR0 and store the count. When Tasks #1-4 are then allowed to run, they check the difference between the first sample and the current time. If the delta is greater than or equal to the delay, then that Event has just happened. We first check if the state of the Serial Output is zero. We then jump to OStateS to start the outputting of the “Start Bit”. Because any Serial Output timings must be rock solid, we use a trick in lines 101-116 that helps greatly. We check if we are within a certain amount of time BEFORE the deadline and then wait for the time to output another bit. This trick allows us to be within a certain ± amount of time within the expected time to output that bit. With this code, we are about <±8% accurate for the Serial Output. You can only use this trick on the most critical tasks, and only on one. In this section of code, we are constantly checking the delta of time from the “FIRST_TMR0_O” reading and the current reading of TMR0. When we are very close to the output time, we jump to line 117. If we are not even close to the proper time, we exit back to the main loop, so we can check the other timers and tasks. Now look at Figure 4 for a description of the Output Pulses, the “Bit units of Time”, and the associated state numbers. Note that the activities are spread out over time.

The timer Events help to define the different states and their associated output activities. Each Event is handled in a very short, well-defined set of code as Task #1. Lines 117-131, are a quick state jump table. You need to break all Real-Time code into very short segments — in and then out. Each segment is just a few lines long. You do your activity, save status, and increment to the next state. Notice that OState0_7 code is used several times to output all 8 bits. The state variable is used also to count the number of bits already outputted. The time to the next outputting of a bit is calculated and is adjusted to take out the accumulation of errors in lines 151-152. We make sure of a full “Stop Bit” length in the OStateE code. In the OStateL code, we reset the OState variable to zero, and tell the world that we are not outputting now in line 157. This is important because we use that bit (OState_B) to Signal that we need to protect the variable xmt_byte that changes over several states. We also use it to Signal that we are ready for another byte to output. Look at Task #4. See how it uses this Semaphore to full advantage. We have just explained a Critical Segment variable as outlined in the theory sections of this article.

Task #2 reads the Serial Input line 1, running at 4800 Baud. The code structure is very similar to that of Task #1 (Figure 3). Notice that there are more states than the Serial Output Task #1. Once the “Start Bit” is detected, we half step into the “Start Bit” to see if it was a “False Start” or not. We then sample and store the incoming bits to form an 8-bit byte just like Task #1. We sample the “Stop Bit” to see if it is a “Frame Error”. We delay another 1/2 bit to get to the end of the “Stop Bit” if there was an “Frame Error” before resetting Task #1’s state to 0. Otherwise, we reset Task #1’s state to 0, and Signal that we are ready for another “Start Bit”. The just received byte is stored in variable “RCV_Storage”. A check is made to see if we already sent out the last received byte before clobbering the old byte with the new byte.

Task #3 reads the Serial Input line 2, running at 4800 Baud. The code structure is the same as Task #2 Figure 3). The received byte is also put into the same storage variable as Task #2 - “RCV_Storage”. When either Task #2 or Task #3 receives a valid byte, Task #8’s counter is reset. You can up the Baud rate of Task #2 and 3 if you lower the output Baud rate of Task #1. Note that for reading the Serial Input Lines, you can be off by ±15% for each sampling, but not accumulatively.

Task #4 finds the next buffered byte to send out through Task #1. Task #4 also controls the order of which byte goes first over another less important byte of data. It can be said that Task #1 Blocks Task #4 from running. You can think of the Serial Output Line as a Shared Resource. The use of Semaphores here allow the Synchronization of data and actions.

Task #5 monitors the Level Reset Input Line and will reset the LED state variable if the line ever goes low. This task is always in the Ready State. This task is said to simply “pole the input line” when ever it can.

Task #6 debounces the seven sensor input lines, running every 20 ms. The variable “T_20_mS_CO” is incremented every 512 µs (Clock Tick) and is compared to the count needed to equal 20 ms. If it is time, the subroutine QCheck_T123 is called to see if Tasks #1-3 are in the Ready State. If any of the Tasks #1-3 are ready, they are ran and we then continue with Task #6. We compare the current value of the input Port_B to see if it stayed the same from the last reading 20 ms back. If the two readings are the same, then Port_B is considered to be stable and the possibly new value is placed in the variable “Old_RB” to be outputted by Task #1. The subroutine D_H_E_L is called to determine the new LED state. We then check if Task #1 was too busy to output the last sensor status byte, if so then that error is recorded.

FIGURE 3: SERIAL INPUT STATES vs. TIME DIAGRAM

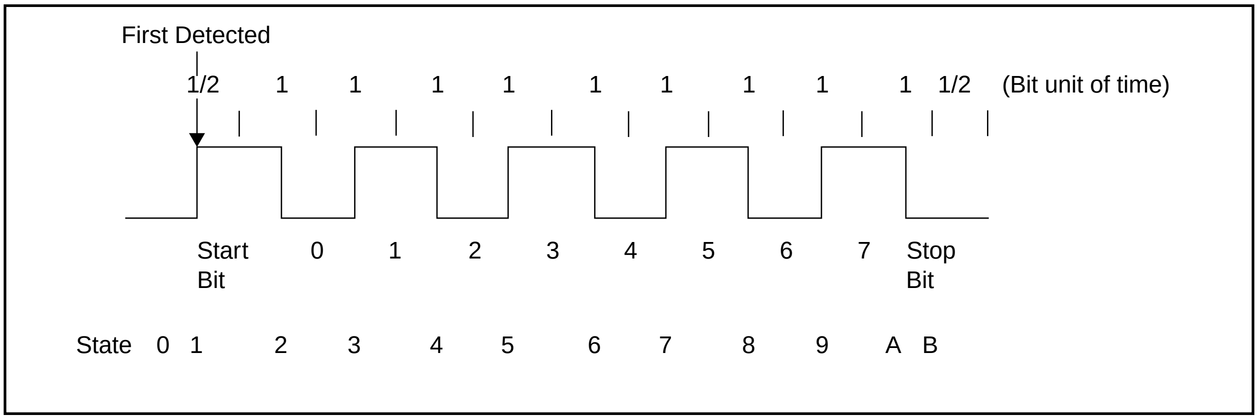
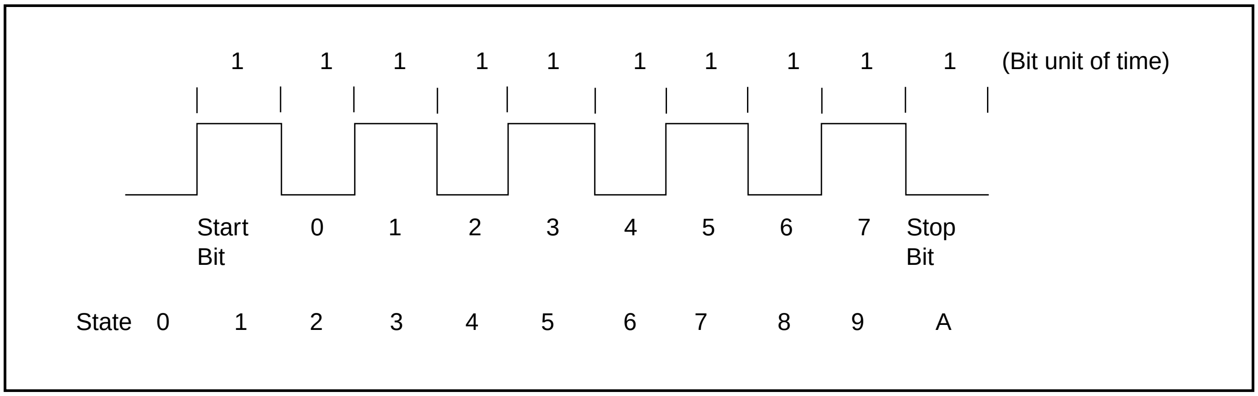


FIGURE 4: SERIAL OUTPUT STATES vs. TIME DIAGRAM



Task #7 outputs the Highest Severity Level Indication on the LED. Do_LED starts at line 161, and continues to 276. This task is also broken into small time units of code. It is constantly checking to see if it is time to switch the on/off condition of the LED. The time units for Task #7 are regulated by the code in lines 613-619. 131072 µS = time unit for Tasks #7-9. Task #7 has many state jump tables so it is included in the first 256 bytes of the first page. Lines 168-175 explain the on and off sequences and offs that represent levels of severity of the input status lines. The variable "LED_Mode" has both Task #7's current state number and the sub-state-number for that state's output sequence.

Task #8 is a 5 second lack of input from either of the two Serial input timers. Tasks #2 and #3 will reset the time counter for Task #8, when either receives a full byte. If the time counter "T_5_S_CO" equals 5 secs, then the LED's state is bumped to the highest, and a special byte is sent down the line to the next "Remote Alarm" unit. The counter variable is reset, and count starts all over. We then check if Task #1 was too busy to output the last special status byte, if so then that error is recorded.

Task #9 measures 5 minutes of calm on the 7 sensor lines and then resets the LED's state. Task #9 needs 16 bits of counter power to record 5 minutes of time. The counter variables are reset after being triggered.

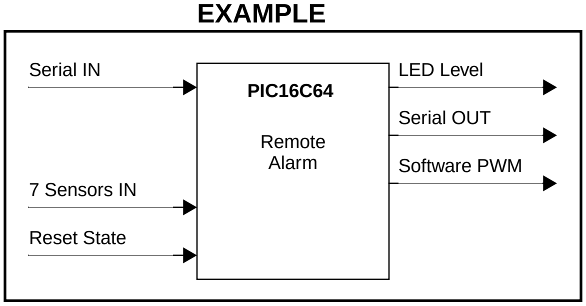
`Do_D_H_E_L` determines the LED's next state based on the 7 sensor input status. This subroutine checks each bit to see if it is active and then checks if a change in the LED's current state needs changing.

`Do_Clear_Regs` clears registers 7-1Fh. It leaves the FSR register zeroed out. This is very important for the PIC16C57 chip.

PIC16C64 - "Remote Alarm64" Example

This example is the same as the PIC16C54 example with a few changes to take advantage of the three timers on the PIC16C64 and interrupts. The second Serial input routine was replaced by an example of a software PWM (Pulse Width Modulation) example. The same code as the PIC16C54 example will run on the PIC16C64 with very few changes using only the TMR0 (TMR0). Be sure to read about the PIC16C54 example, as the comments will not be repeated, except to make a strong point.

FIGURE 5: REMOTE ALARM - PIC16C64 EXAMPLE



This example operates at 4 Mhz. By simply increasing the crystal speeds, you can change the input and output Baud rates just as outlined in the section on the PIC16C54 example's crystal selection. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds.

> **Note:** You must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C64 code example. Lines 1-78 are the equates for this program. Notice that there is no need for jump tables for subroutines to be in the "first 256 bytes" of each page as there was in the PIC16C54 example. Note that the "Reset Vector" is now at code address 0, and the "Interrupt Vector" is at code address 4. Task #1 and 2 have been simplified greatly by using interrupts and timers. For Task #1, we no longer need to use the "trick" any more. It is time to execute once the routines for Task #1 and others are called. The section of code that handles the "Start Bit" (OStateS) lines 106-122 has been changed to setting up TMR2 with its interrupt to trigger the next call to this subroutine. The initial `CALL` to this subroutine was by Task #4, but later calls are due to Timer 2's interrupts. The amount of time until the next interrupt is set by each state's code. This amount is based on the "Bit Unit of Time" which is based on Baud rate and crystal speed. An easy change to the code is to add a software selectable and "changeable on the fly" Baud rate. This is done by having a variable that selects the new Baud rate from the two data tables. One table gets you the Bit Delay value - see line 110. The other data table gets the value to be put into T2CON - see line 107, which selects the Post and Pre-scalers. You may need to adjust the Bit Delay value to take in account the Interrupt Latency. The OStateL state code shuts down Timer2 and its interrupt. See lines 647-676 to understand how we get here by interrupt. Once Timer 2's count equals the count we put into register PR2, we get an interrupt if the following three conditions are true:

1. Not already in an interrupt. When the current interrupt is done, our interrupt will be executed.
2. GIE and PEIE bits are set.
3. TMR2IE bit is set.

Remember to clear the Flag bit as in line 114 before returning from an interrupt. Return from this subroutine will return you back to Task #4 or back to the ISR handle lines 647-676 depending on who called this routine. The Task #7's subroutines are the same as in the PIC16C54 example, lines 151-268. Task #2 is different from the previous example, lines 288-380. First Task #2 uses two interrupts. The INT interrupt on pin RB0/INT is used to detect the "Start Bit". It is very accurate. It is turned off after the detection in I1StateS code. The second interrupt TMR1 is then Enabled in the I1StateS code. Timer1 is then used to cause an interrupt for all the other states for Task #2. Notice that Timer1 has a

16-bit counter and we calculate the amount of Clock Ticks until overflow in lines 329-333. In the state code I1StateL, TMR1 is shut down, and the INT interrupt is now Enabled so as to detect the next input byte. The initializing of the PIC16C64 variable takes place in lines 383-426. The initializing of the tasks take place in lines 427-451. Notice that the last bit to be set is the GIE bit in line 451 after ALL is setup. There are several ways to execute the Task #3-9 code: by Timer0 overflow interrupt, by having the code be in the background as in this example. The trade-offs are many, and too deep for this article. Notice that the subroutine QCheck_T123 is not needed in this method. Timer0 overflow interrupt sets the flag: Time_Bit. The code in lines 454-457 can be considered the “IDLE Task” on some systems. It waits for a Clock Tick from TMR0’s overflow. Task #3 is new, and is a simple 8-bit software PWM. Lines 459-478 show how to have 8 bits of ON, and 8 bits of OFF. This task has two states, on and off. You may add to the code by allowing the Real-Time changing of the 8-bit values under software control. When you change the values in the variables PWM_Out and PWM_In, disable all interrupts by using the following line: `BCF INTCON,GIE`, and enable all interrupts by using the following line: `BSF INTCON,GIE`. The new values will be used at the next transition, thus allowing a smooth change. This task could easily be used in the PIC16C54 example type Kernel. Task #4 is the same except that it calls Task #1’s subroutine to initiate the outputting of a byte. See line 503. Tasks #5-9 are the same as in the PIC16C54 example. The subroutines: `D_H_E_L` and `Clear_Regs` are the same in both examples. The TMR0 (Timer0) Overflow interrupt ISR (Interrupt Service Routine) is lines 641-645. This ISR will set the Time_Bit bit and clear the Flag that caused the interrupt. The Interrupt code lines 647-676 handles the saving of the Context Registers and the restoring of the Context Registers (W, Status, FSR) and by checking the order which interrupts are to be handled first - see lines 656-669. A very important line is 654. You must set the memory page pointers here for the ISR routines! Line 676 is the only place that an interrupt is allowed to return and set the GIE bit (`RETFIE`).

Reentrant example

See Appendix E for the short code segment. This code corresponds to lines 302-332 in the PIC16C54 example. The purpose of reentrant code is to allow two or more tasks to use the same code at the “same time”. See the section about reentrant in the theory section of this article. Notice how the registers 18h-1Bh match the registers 1Ch-1Fh, both starting with the state variable for the two tasks using this routine. Note how Task #2 and Task #3 load a pointer to the state variable for their task before calling DO_I State code. By using the FSR register as a pointer, and incrementing or decrementing the FSR register, you can keep the variables in the two tasks straight even if the two tasks are using different code in the subroutine at any one time. This method is not easy to implement, as can be seen, so use two copies for readability instead, like the PIC16C54 example.

SUMMARY

Now that the PICmicro family of microcontrollers have a way of executing Real-Time Programs, using the techniques outlined in this article, there is very little that PICmicros cannot do! Much more than was ever dreamed before. Many of you will quickly understand and start modifying these examples. Great. That means that we have done our job at Myriad. A few of you may want more help. Great. At Myriad Development Co., we LOVE the PICmicro family.

BIBLIOGRAPHY

Foster, Caxton C.
Real Time Programming - Neglected Topics
Reading, Massachusetts
Addison-Wesley Publishing Company, 1981

Holt, R.C., Graham, G.S., Lazowska, E.D., Scott, M.A.
Structured CONCURRENT PROGRAMMING with Operating Systems Applications
Reading, Massachusetts
Addison-Wesley Publishing Company, 1978

Kaisler, Stephen H.
The Design of Operating Systems for Small Computer Systems
New York, NY
John Wiley & Sons, 1983

Labrosse, Jean J.
uC/OS - The Real-Time Kernel
Lawrence, Kansas
R & D Publications, 1992

Loeliger, R.G.
Threaded Interpretive Languages
Peterborough, NH
BYTE BOOKS, 1981

APPENDIX A:

A Real-Time Vocabulary

ASYNCHRONOUS - An activity that can happen at any moment, at any time.

BLOCKING - The act of wanting to waiting for an EVENT before continuing.

CLOCK TICK - The heart beat that all time is based on.

CONTEXT/TASK SWITCH - Module that saves and restores the states of a task.

CRITICAL SECTION - Section of code or hardware - only one user at a time.

DEADLOCK - That is where two TASKs are waiting for each others resources.

DISPATCHING - The act of starting up a TASK to run from an RT Kernel.

DYNAMIC PRIORITIES - The ability for TASKs to have there PRIORITIES changed.

DYNAMIC TASKING - The creation and the killing of TASKs.

EMBEDDED SYSTEM - An internal system that operates all by itself.

ENABLING/DISABLING INTERRUPTS - Controlling the interrupting processing.

EVENT - Timer, communication, handshaking, interrupts, data, external events.

EVENT FLAGS - The storage of current states or info on what has happened.

INTERRUPT - A hardware event (external/internal) that triggers a jump to the ISR routines to handle that event.

INTERRUPT LATENCY - How long it takes once signaled to start an ISR.

INTERRUPT RECOVERY - How long it takes once interrupted to return back to code.

KERNEL - Module that controls TASKs, INTERRUPTs, and intertask communications.

MAILBOXES - Away to pass data from one TASK to another.

MASKABLE INTERRUPTS - The ability to control whether an ISR is called or not.

MULTITASKING - The act of several TASKs thinking they own the CPU.

MUTUAL EXCLUSION - The act of allowing only ONE owner to a RESOURCE.

NMI - NON-MASKABLE INTERRUPT - Can not be turned off by software.

READY STATE - Referring to a list of TASKs ready (having work to do NOW).

REENTRANT - Code that can be used by several TASKs at the same time.

RESOURCE - Data structures, display, I/O hardware, non-reentrant routines.

RUNNING STATE - Referring to the ONE task owning/using the CPU currently .

SCHEDULER - That part of a kernel that decides which TASK to run next.

SEMAPHORES - A protocol to control RESOURCES, SIGNAL EVENTS, synchronize tasks.

SIGNAL - The act of one task signaling another that something has happened.

STATE MACHINE - An important concept in dividing a job into TASKs & ISRs.

SYNCHRONIZATION - Were TASKs synchronize over data or at a special time.

TASK PRIORITY - Each TASK is ranked as to its importance to getting done.

TASK/THREAD - Code that is defined by a small coherent job/work to be done.

TIME SLICING - The act of giving the same amount of “time” to each TASK to run.

TRAP - A software caused interrupt, useful for system access.

WAITING STATE - Referring to a list of TASKs waiting for an EVENT(s).

Please check the Microchip BBS for the latest version of the source code.
Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe®
(CompuServe membership not required).

APPENDIX B:

```
MPASM 01.40 Released            APP_B.ASM   1-16-1997  17:09:04        PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                    00001         list    p=16C54,t=ON,c=132
                    00002 ;
                    00003 ;*****************************************************************
                    00004 ;
                    00005 ; 'Reentrant Code Example' Designed by Myriad Development Co. - Jerry
Farmer
                    00006 ;       PIC16C54, 4MHz Crystal, WatchDog Timer OFF
                    00007 ;
                    00008 ;       Program:          APP_B.ASM
                    00009 ;       Revision Date:
                    00010 ;                         1-15-97   Compatibility with MPASMWIN 1.40
                    00011 ;
                    00012 ;*******************************************************************
                    00013 ;
                    00014 ; Register Files
  00000018          00015 IState1         equ     18h   ;Serial In #1 State
  00000019          00016 First_TMR0_I1   equ     19h   ;Starting time for next #1 Input event
  0000001A          00017 nbti1           equ     1Ah   ;Next Bit #1 In Time - variable time
  0000001B          00018 rcv_byte_1      equ     1Bh   ;Receive Serial #1 In byte
  0000001C          00019 IState2         equ     1Ch   ;Serial In #2 State
  0000001D          00020 First_TMR0_I2   equ     1Dh   ;Starting time for next #2 Input event
  0000001E          00021 nbti2           equ     1Eh   ;Next Bit #2 In Time - variable time
  0000001F          00022 rcv_byte_2      equ     1Fh   ;Receive Serial #2 In byte
                    00023
                    00024     INCLUDE    <P16C5X.INC>
                    00001         LIST
                    00002 ;P16C5X.INC Standard Header File,Version 3.30 Microchip Technology,Inc.
                    00224         LIST
                    00025
  00000007          00026 temp          EQU     07h   ;Temporary holding register - PIC16C54/56
  00000010          00027 IStateS       EQU     10H
  00000011          00028 IStateS2      EQU     11H
  00000012          00029 IState0_7     EQU     12H
  00000013          00030 IStateE       EQU     13H
  00000014          00031 IStateL       EQU     14H
                    00032
                    00033 ;******  ;Task 2,3 - Asynchronous 2400 Baud Serial Input (LOW=0)
0000                00034 Do_IState
0000 0220           00035               movf    INDF, F   ;if IState2 == 0
0001 0643           00036               btfsc   STATUS,Z  ;  then Do Start Bit
0002 0A10           00037               goto    IStateS
0003 0201           00038               movf    TMR0,W    ;Get current time
0004 0027           00039               movwf   temp
0005 02A4           00040               incf    FSR, F    ;Point to First_TMR0_I(1,2)
0006 0200           00041               movf    INDF,W    ;Get elapsed time; Time Unit = 2 uS
0007 00A7           00042               subwf   temp, F
0008 02A4           00043               incf    FSR, F    ;Point to nbti(1,2)
0009 0200           00044               movf    INDF,W    ;Past time for next input bit ?
000A 0087           00045               subwf   temp,W
000B 0703           00046               btfss   STATUS,0
000C 0A1E           00047               goto    L1
000D                00048 L0
```



```
000D 0C02           00049                movlw   2          ;Point to IState(1,2)
000E 00A4           00050                subwf   FSR, F
000F 0200           00051                movf    INDF,W     ;Get (0-B) mode #
0010 0E0F           00052                andlw   H'0F'      ;Get only mode #
0011 01E2           00053                addwf   PCL, F     ;jump to subroutine
                    00054
0012 0A10           00055                goto    IStateS    ;Serial Start Bit
0013 0A11           00056                goto    IStateS    ;1/2 of Start Bit - see if False Start
0014 0A12           00057                goto    IState0_7 ;Bit 0
0015 0A12           00058                goto    IState0_7 ;Bit 1
0016 0A12           00059                goto    IState0_7 ;Bit 2
0017 0A12           00060                goto    IState0_7 ;Bit 3
0018 0A12           00061                goto    IState0_7 ;Bit 4
0019 0A12           00062                goto    IState0_7 ;Bit 5
001A 0A12           00063                goto    IState0_7 ;Bit 6
001B 0A12           00064                goto    IState0_7 ;Bit 7
001C 0A13           00065                goto    IStateE    ;Serial Stop Bit
001D 0A14           00066                goto    IStateL    ;Last State
001E                00067 L1
001E 0064           00068                clrf    FSR        ;Clear the FSR register
001F 0800           00069                retlw   0
                    00070
                    00071 ;*****
0020                00072 Task_2        ;Task 2 - Asynchronous 2400 Baud Serial Input (LOW=0)
0020 0C18           00073                movlw   IState1     ;Point to IState1
0021 0024           00074                movwf   FSR
0022 0900           00075                call    Do_IState
                    00076 ;*****
0023                00077 Task_3        ;Task 3 - Asynchronous 2400 Baud Serial Input (LOW=0)
0023 0C1C           00078                movlw   IState2     ;Point to IState2
0024 0024           00079                movwf   FSR
0025 0900           00080                call    Do_IState
                    00081
                    00082                  END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX---------- ----------------

All other memory blocks unused.

Program Memory Words Used:    38
Program Memory Words Free:   474


Errors   :      0
Warnings :      0 reported,     0 suppressed
Messages :      0 reported,     0 suppressed
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C:

```
MPASM 01.40 Released            APP_C.ASM   1-16-1997  17:09:32         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                    00001 ;'Remote Alarm' V1.02
                    00002 ;   Designed by Myriad Development Co/- Jerry Farmer
                    00003 ;     PIC16C54, 4MHz Crystal,
                    00004 ;       WatchDog Timer OFF, MPASM instruction set
                    00005 ;
                    00006 ;       Program:          APP_C.ASM
                    00007 ;       Revision Date:
                    00008 ;                         1-15-97  Compatibility with MPASMWIN 1.40
                    00009 ;
                    00010 ;*****************************************************************
                    00011 ;
                    00012         list    p=16C54,t=ON,c=132
                    00013
                    00014         include "P16C5X.INC"
                    00001         LIST
                    00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology,Inc.
                    00224         LIST
                    00015
                    00016 ; Constants
  00000000          00017 INDIR         equ  0         ;Indirect Register
  00000033          00018 OUT_BIT_TIME  equ  33h       ;9600 Baud, 104uS Bit Rate
  00000064          00019 IN_BIT_TIME   equ  64h       ;4800 Baud, 208uS Bit Rate
  00000023          00020 FUDGE_TIME    equ  23h       ;Current Time within a Fudge Factor
                    00021
                    00022 ; B Register Definitions
                    00023 #define Level_Reset     PORTB,  ;Low will cause Past Level to reset
                    00024                                 ;RB.7 - RB.1 == Input from Sensors
  000000FF          00025 RB_TRIS       equ  B'11111111'  ;RB TRIS at INIT State == all input
  00000000          00026 RB_MASK       equ  B'00000000'  ;What is High/Low for RB at INIT State
                    00027
                    00028 ; A Register Definitions - Programmable Inputs
                    00029 #define Serial_IN_1     PORTA,0  ;Serial Input #1 - 8 bits
                    00030 #define LED             PORTA,1  ;LED Output - Level/State Indicator
                    00031 #define Serial_Out      PORTA,2  ;Serial Output - 8 bits + passwords
                    00032 #define Serial_IN_2     PORTA,3  ;Serial Input #2 - 8 bits
                    00033
  000000F9          00034 RA_TRIS       equ  B'11111001' ;RA TRIS at INIT State
  00000000          00035 RA_MASK       equ  B'00000000' ;What is High/Low for RA at INIT State
                    00036
                    00037 ; Register Files
  00000007          00038 temp          equ   07h      ;Temporary holding register - PIC16C54/56
  00000008          00039 Timer_Bits    equ   08h      ;Indicates which Timer(s) are Active = 1
  00000009          00040 Flags         equ   09h      ;Error Flags
  0000000A          00041 LED_Mode      equ   0Ah      ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
  0000000B          00042 OState        equ   0Bh      ;Serial Out State
  0000000C          00043 T_5_M_LO      equ   0Ch      ;5 Min Timer Counter - Low
  0000000D          00044 T_5_M_HI      equ   0Dh      ;5 Min Timer Counter - High
  0000000E          00045 T_5_S_CO      equ   0Eh      ;5 Second Timer - lack of Serial Input
  0000000F          00046 T_20_mS_CO    equ   0Fh      ;20 mS Timer - used for debouncing
  00000010          00047 LED_C         equ   10h      ;LED Counter
  00000011          00048 Last_TMR0     equ   11h      ;Last value of the TMR0
  00000012          00049 First_TMR0_O  equ   12h      ;Starting time for next Output event
  00000013          00050 xmt_byte      equ   13h      ;Serial xmit byte - destroyed in use
  00000014          00051 cc            equ   14h      ;256 * TMR0 time
```

```
  00000015         RCV_Storage   equ   15h       ;Long term storage of rcv_byte #1 & 2
  00000016         Old_RB        equ   16h       ;Oldest/Master copy of RB
  00000017         Last_RB       equ   17h       ;Last copy of RB
  00000018         IState1       equ   18h       ;Serial In #1 State
  00000019         First_TMR0_I1 equ   19h       ;Starting time for next #1 Input event
  0000001A         nbti1         equ   1Ah       ;Next Bit #1 In Time - variable time
  0000001B         rcv_byte_1    equ   1Bh       ;Receive Serial #1 In byte
  0000001C         IState2       equ   1Ch       ;Serial In #2 State
  0000001D         First_TMR0_I2 equ   1Dh       ;Starting time for next #2 Input event
  0000001E         nbti2         equ   1Eh       ;Next Bit #2 In Time - variable time
  0000001F         rcv_byte_2    equ   1Fh       ;Receive Serial #2 In byte

                   ; Indicates which Timer(s) are Active = 1 & Flags
                   #define OState_B      Timer_Bits,0;Serial Out Active Bit
                   #define IState1_B     Timer_Bits,1;Serial IN #1 Active Bit
                   #define IState2_B     Timer_Bits,2;Serial IN #2 Active Bit
                   #define T_5_S_B       Timer_Bits,3;5 Second Timer Active Bit
                   #define T_5_M_B       Timer_Bits,4;5 Min Timer Active Bit
                   #define RCV_Got_One_B Timer_Bits,5;Got a NEW Received byte to send out
                   #define RB_NEW_B      Timer_Bits,6;Indicates a change in RB input
                   #define S_5_S_B       Timer_Bits,7;Serial In 5 secs of inactivity

                   ; Error Flags
                   #define FS_Flag_1      Flags,0    ;Serial #1 IN had a False Start Error
                   #define FE_Flag_1      Flags,1    ;Last Serial #1 IN had a Frame Error
                   #define FS_Flag_2      Flags,2    ;Serial #2 IN had a False Start Error
                   #define FE_Flag_2      Flags,3    ;Last Serial #2 IN had a Frame Error
                   #define RCV_Overflow   Flags,4    ;Lost Serial Input Byte - too Slow
                   #define RB_Overflow    Flags,5    ;Lost RB Input Byte - too Slow
                   #define S_5_S_Overflow Flags,6    ;Lost ‘5S Inactivity’ msg - too Slow

                   ;Equates for LED Task #7
                   #define LED_B          LED_Mode,3     ;LED is active
                   #define LED_NEW_B      LED_Mode,7     ;LED has just changed Modes = 1
  00000008         LED_OFF_MODE    equ    B’00001000’    ;LED OFF
  00000089         LED_SEQ1_MODE   equ    B’10001001’    ;LED Sequence 1: .2s On, 1s Off
  0000008A         LED_SEQ2_MODE   equ    B’10001010’    ;LED Sequence 2: 3x(.2s), 1s Off
  0000008B         LED_SEQ3_MODE   equ    B’10001011’    ;LED Sequence 3: 5x(.2s), 1s Off
  0000009C         LED_SLOW_MODE   equ    B’10011100’    ;LED Slow Pulsing - .3 Hz
  0000009D         LED_MEDIUM_MODE equ    B’10011101’    ;LED Medium Pulsing - 1 Hz
  0000009E         LED_FAST_MODE   equ    B’10011110’    ;LED Fast Pulsing - 3 Hz
  0000008F         LED_ON_MODE     equ    B’10001111’    ;LED ON Continuously


                   ; Clear Registers 7-1Fh
0000               Clear_Regs
0000 0BE9                  GOTO    Do_Clear_Regs   ;Save space in first 256 bytes

                   ; Determine the Highest Error Level & Start Task #7 outputing the new
0001               D_H_E_L
0001 0BD2                  GOTO    Do_D_H_E_L      ;Save space in first 256 bytes
                                                   ;Level
                   ;******         ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0002               Do_OState
0002 022B               MOVF    OState, F       ;if OState == 0
0003 0643               BTFSC   STATUS,Z        ;
0004 0A24               GOTO    OStateS         ;then goto Output-Start-Bit
0005 0201               MOVF    TMR0,W          ;Get current time
0006 0027               MOVWF   temp            ; & store in Temporary variable
0007 0212               MOVF    First_TMR0_O,W  ;Get elapsed time; Time Unit = 2 uS
0008 00A7               SUBWF   temp, F         ;Delta of Current Time & Orginal Time
0009 0C23               MOVLW   FUDGE_TIME      ;Take in account processing time to do it
000A 0087               SUBWF   temp,W          ;Time within fudge factor ?
000B 0703               BTFSS   STATUS,C
000C 0A23               GOTO    _0005           ;Not time yet to change States so return
000D 0C33          _0003   MOVLW   OUT_BIT_TIME    ;Past time for next out-bit ?
```

```
000E 0087            00118          SUBWF   temp,W
000F 0603            00119          BTFSC   STATUS,C        ;Do some delaying until it is time
0010 0A15            00120          GOTO    _0004           ;It is now time to out put a bit
0011 0C04            00121          MOVLW   H'04'           ;Account for loop delay
0012 01E7            00122          ADDWF   temp, F
0013 0000            00123          NOP                     ;   make loop delay even
0014 0A0D            00124          GOTO    _0003           ;Wait for exact time to output bit
0015 020B            00125 _0004    MOVF    OState,W        ;Get (0-A) mode #
0016 0E0F            00126          ANDLW   H'0F'           ;Get only mode #
0017 01E2            00127          ADDWF   PCL, F          ;jump to subroutine
0018 0A24            00128          GOTO    OStateS         ;Serial Start Bit
0019 0A2B            00129          GOTO    OState0_7       ;Bit 0
001A 0A2B            00130          GOTO    OState0_7       ;Bit 1
001B 0A2B            00131          GOTO    OState0_7       ;Bit 2
001C 0A2B            00132          GOTO    OState0_7       ;Bit 3
001D 0A2B            00133          GOTO    OState0_7       ;Bit 4
001E 0A2B            00134          GOTO    OState0_7       ;Bit 5
001F 0A2B            00135          GOTO    OState0_7       ;Bit 6
0020 0A2B            00136          GOTO    OState0_7       ;Bit 7
0021 0A31            00137          GOTO    OStateE         ;Serial Stop Bit
0022 0A36            00138          GOTO    OStateL         ;Last State
0023 0800            00139 _0005    RETLW   H'00'
                     00140
0024                 00141 OStateS
0024 0545            00142          BSF     Serial_Out      ;Serial Start Bit
0025 0201            00143          MOVF    TMR0,W          ;Store starting time
0026 0032            00144          MOVWF   First_TMR0_O
0027 0C0D            00145          MOVLW   H'0D'           ;Fudge again
0028 00B2            00146          SUBWF   First_TMR0_O, F
0029 02AB            00147          INCF    OState, F       ;increment to next state
002A 0800            00148          RETLW   H'00'
                     00149
002B                 00150 OState0_7                        ;Bit 0 - 7
002B 0333            00151          RRF     xmt_byte, F     ;Move bit into C from right most bit
002C 0703            00152          BTFSS   STATUS,C        ;
002D 0445            00153          BCF     Serial_Out      ;
002E 0603            00154          BTFSC   STATUS,C        ;
002F 0545            00155          BSF     Serial_Out      ;
0030 0A32            00156          GOTO    OS_End
0031                 00157 OStateE
0031 0445            00158          BCF     Serial_Out      ;Serial Stop Bit
0032 0C33            00159 OS_End   MOVLW   OUT_BIT_TIME    ;Adjust out the cumulation of error
0033 01F2            00160          ADDWF   First_TMR0_O, F
0034 02AB            00161          INCF    OState, F       ;increment to next state
0035 0800            00162          RETLW   H'00'
0036                 00163 OStateL
0036 006B            00164          CLRF    OState          ;Ready to send next byte out
0037 0408            00165          BCF     OState_B        ;Serial Out not active
0038 0800            00166          RETLW   H'00'
                     00167
                     00168 ;******          ;Task #7 - Output Highest Level Indication on LED
0039                 00169 Do_LED
0039 06EA            00170          BTFSC   LED_NEW_B       ;Initialize regs if change in modes
003A 0A4C            00171          GOTO    LED_NEW
003B 02B0            00172          INCF    LED_C, F        ;Inc Counter - Time Unit = 131072 uS
003C 020A            00173          MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 0E07            00174          ANDLW   H'07'           ;Get only mode #
003E 01E2            00175          ADDWF   PCL, F          ;jump to subroutine
003F 0A48            00176          GOTO    LED_OFF         ;LED OFF
0040 0A64            00177          GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 0A67            00178          GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 0A8A            00179          GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 0A50            00180          GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 0A5E            00181          GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 0A61            00182          GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 0A4D            00183          GOTO    LED_ON          ;LED ON Continuously
```

```
0047 0800           _0012   RETLW   H'00'
                    ;------
0048                LED_OFF
0048 0425                   BCF     LED             ;Turn off LED
0049 046A                   BCF     LED_B           ;LED must be off
004A 0070                   CLRF    LED_C           ;Reset Counter - LED_C = 0
004B 0800                   RETLW   H'00'
                    ;------
004C                LED_NEW
004C 04EA                   BCF     LED_NEW_B       ;Done initializing
004D                LED_ON
004D 0525                   BSF     LED             ;Turn on LED
004E 0070                   CLRF    LED_C           ;Reset Counter - LED_C = 0
004F 0800                   RETLW   H'00'
                    ;------
0050                LED_SLOW
0050 0C0C                   MOVLW   H'0C'           ;.3Hz @ 50% Duty
0051 0027                   MOVWF   temp
0052 0207           LED_S   MOVF    temp,W          ;Check LED_C if time, .3Hz @ 50% Duty
0053 0090                   SUBWF   LED_C,W
0054 0743                   BTFSS   STATUS,Z
0055 0A47                   GOTO    _0012
0056 0C10                   MOVLW   H'10'
0057 01AA                   XORWF   LED_Mode, F     ;Switch states
0058 078A                   BTFSS   LED_Mode,4      ;Now make LED same state
0059 0425                   BCF     LED
005A 068A                   BTFSC   LED_Mode,4
005B 0525                   BSF     LED
005C 0070                   CLRF    LED_C           ;Reset LED_C
005D 0800                   RETLW   H'00'
                    ;------
005E                LED_MEDIUM
005E 0C04                   MOVLW   H'04'           ;1Hz @ 50% Duty
005F 0027                   MOVWF   temp
0060 0A52                   GOTO    LED_S           ;Go do it
                    ;------
0061                LED_FAST
0061 0C01                   MOVLW   H'01'           ;3Hz @ 50% Duty
0062 0027                   MOVWF   temp
0063 0A52                   GOTO    LED_S           ;Go do it
                    ;------
0064                LED_SEQ1                        ;.2 ON, 1 OFF
0064 078A                   BTFSS   LED_Mode,4      ;Skip if bit is high
0065 0A76                   GOTO    ON1             ;Go do it
0066 0A82                   GOTO    OFF3            ;Go do it
                    ;------
0067                LED_SEQ2                        ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 020A                   MOVF    LED_Mode,W
0068 0027                   MOVWF   temp
0069 0C30                   MOVLW   H'30'           ;Get sequence # only
006A 0167                   ANDWF   temp, F
006B 03A7                   SWAPF   temp, F         ;swap nibbles
006C 0207                   MOVF    temp,W          ;get nibble for offset
006D 01E2                   ADDWF   PCL, F          ;Table jump calculation
006E 0A76                   GOTO    ON1             ;LED is on, check if time to change
006F 0A7C                   GOTO    OFF2            ;LED is off, check if time to change
0070 0A76                   GOTO    ON1             ;LED is on, check if time to change
0071 0A82                   GOTO    OFF3            ;LED is off, check if time to change
                    ;------
0072                LED_Exit
0072 0C10                   MOVLW   H'10'           ;Inc Seq #
0073 01EA                   ADDWF   LED_Mode, F
0074 0070                   CLRF    LED_C           ;Reset LED_C
0075 0800                   RETLW   H'00'
0076                ON1
0076 0C02                   MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
```

```
0077 0090                SUBWF   LED_C,W
0078 0743                BTFSS   STATUS,Z
0079 0A47                GOTO    _0012
007A 0425                BCF     LED             ;Turn off LED
007B 0A72                GOTO    LED_Exit
007C         OFF2
007C 0C02                MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0090                SUBWF   LED_C,W
007E 0743                BTFSS   STATUS,Z
007F 0A47                GOTO    _0012
0080 0525                BSF     LED             ;Turn on LED
0081 0A72                GOTO    LED_Exit
0082         OFF3
0082 0C08                MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0090                SUBWF   LED_C,W
0084 0743                BTFSS   STATUS,Z
0085 0A47                GOTO    _0012
0086 0525                BSF     LED             ;Turn on LED
0087 0CF0                MOVLW   H'F0'
0088 012A                IORWF   LED_Mode, F     ;Cause (Seq# & NEW) to overflow to 0
0089 0A72                GOTO    LED_Exit
008A         LED_SEQ3      ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 020A                MOVF    LED_Mode,W      ;Get LED info
008B 0027                MOVWF   temp
008C 0C70                MOVLW   H'70'           ;Get sequence # only
008D 0167                ANDWF   temp, F
008E 03A7                SWAPF   temp, F         ;swap nibbles
008F 0207                MOVF    temp,W          ;get nibble for offset
0090 01E2                ADDWF   PCL, F          ;Table jump calculation
0091 0A76                GOTO    ON1             ;LED is on check if time to change
0092 0A7C                GOTO    OFF2            ;LED is off check if time to change
0093 0A76                GOTO    ON1             ;LED is on check if time to change
0094 0A7C                GOTO    OFF2            ;LED is off check if time to change
0095 0A76                GOTO    ON1             ;LED is on check if time to change
0096 0A82                GOTO    OFF3            ;LED is off check if time to change

             ;****               Quick Check of Tasks #1, #2 and #3
0097         QCheck_T123
                         ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0097 0708                BTFSS   OState_B        ;if not outputing now then skip call
0098 0A9A                GOTO    T2
0099 0902                CALL    Do_OState       ;Go Do Task #1

                         ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
009A 0628    T2          BTFSC   IState1_B       ;if already started then call
009B 0A9F                GOTO    _0029
009C 0605                BTFSC   Serial_IN_1     ;if Start bit ? then call
009D 0A9F                GOTO    _0029
009E 0AA0                GOTO    T3
009F 09A7    _0029       CALL    Do_I1State      ;Go Do Task #2

                         ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
00A0 0648    T3          BTFSC   IState2_B       ;if already started then call
00A1 0AA5                GOTO    _0031
00A2 0665                BTFSC   Serial_IN_2     ;if Start bit ? then call
00A3 0AA5                GOTO    _0031
00A4 0800                RETLW   H'00'
00A5 09C2    _0031       CALL    Do_I2State      ;Go Do Task #3
00A6 0800                RETLW   H'00'

             ;******             ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
00A7         Do_I1State
00A7 0238                MOVF    IState1, F      ;if IState1 == 0
00A8 0643                BTFSC   STATUS,Z        ;               then Do Start Bit
00A9 0ADD                GOTO    I1StateS
00AA 0201                MOVF    TMR0,W          ;Get current time
```

```
00AB 0027                  MOVWF   temp
00AC 0219                  MOVF    First_TMR0_I1,W ;Get elapsed time; Time Unit = 2 uS
00AD 00A7                  SUBWF   temp, F
00AE 021A                  MOVF    nbti1,W         ;Past time for next input bit ?
00AF 0087                  SUBWF   temp,W
00B0 0703                  BTFSS   STATUS,C
00B1 0AC1                  GOTO    _0033
00B2 0218                  MOVF    IState1,W       ;Get (0-B) mode #
00B3 0E0F                  ANDLW   H'0F'           ;Get only mode #
00B4 01E2                  ADDWF   PCL, F          ;jump to subroutine
00B5 0ADD                  GOTO    I1StateS        ;Serial Start Bit
00B6 0AE6                  GOTO    I1State2        ;1/2 of Start Bit - see if False Start
00B7 0AEF                  GOTO    I1State0_7      ;Bit 0
00B8 0AEF                  GOTO    I1State0_7      ;Bit 1
00B9 0AEF                  GOTO    I1State0_7      ;Bit 2
00BA 0AEF                  GOTO    I1State0_7      ;Bit 3
00BB 0AEF                  GOTO    I1State0_7      ;Bit 4
00BC 0AEF                  GOTO    I1State0_7      ;Bit 5
00BD 0AEF                  GOTO    I1State0_7      ;Bit 6
00BE 0AEF                  GOTO    I1State0_7      ;Bit 7
00BF 0AF8                  GOTO    I1StateE        ;Serial Stop Bit
00C0 0B03                  GOTO    I1StateL        ;Last State - End of Stop Bit
00C1             _0033
00C1 0800                  RETLW   H'00'

                 ;******                   ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
00C2             Do_I2State
00C2 023C                  MOVF    IState2, F      ;if IState1 == 0
00C3 0643                  BTFSC   STATUS,Z        ;then Do Start Bit
00C4 0B10                  GOTO    I2StateS
00C5 0201                  MOVF    TMR0,W          ;Get current time
00C6 0027                  MOVWF   temp
00C7 021D                  MOVF    First_TMR0_I2,W ;Get elapsed time; Time Unit = 2 uS
00C8 00A7                  SUBWF   temp, F
00C9 021E                  MOVF    nbti2,W         ;Past time for next input bit ?
00CA 0087                  SUBWF   temp,W
00CB 0703                  BTFSS   STATUS,C
00CC 0ADC                  GOTO    _0035
00CD 021C                  MOVF    IState2,W       ;Get (0-B) mode #
00CE 0E0F                  ANDLW   H'0F'           ;Get only mode #
00CF 01E2                  ADDWF   PCL, F          ;jump to subroutine
00D0 0B10                  GOTO    I2StateS        ;Serial Start Bit
00D1 0B19                  GOTO    I2StateS2       ;1/2 of Start Bit - see if False Start
00D2 0B22                  GOTO    I2State0_7      ;Bit 0
00D3 0B22                  GOTO    I2State0_7      ;Bit 1
00D4 0B22                  GOTO    I2State0_7      ;Bit 2
00D5 0B22                  GOTO    I2State0_7      ;Bit 3
00D6 0B22                  GOTO    I2State0_7      ;Bit 4
00D7 0B22                  GOTO    I2State0_7      ;Bit 5
00D8 0B22                  GOTO    I2State0_7      ;Bit 6
00D9 0B22                  GOTO    I2State0_7      ;Bit 7
00DA 0B2B                  GOTO    I2StateE        ;Serial Stop Bit
00DB 0B36                  GOTO    I2StateL        ;Last State - End of Stop Bit
00DC 0800        _0035     RETLW   H'00'

                 ;***              ;Subroutines for Task #2
00DD             I1StateS                          ;Start Bit - Setup timing variables
00DD 0528                  BSF     IState1_B       ;Serial Input Active
00DE 0201                  MOVF    TMR0,W          ;Store starting time
00DF 0039                  MOVWF   First_TMR0_I1
00E0 0C0D                  MOVLW   H'0D'           ;Fudge again
00E1 00B9                  SUBWF   First_TMR0_I1, F
00E2 0C32                  MOVLW   H'32'           ;Time delay = 1/2 bit time
00E3 003A                  MOVWF   nbti1
00E4 02B8                  INCF    IState1, F      ;Increment to next state
00E5 0800                  RETLW   H'00'
```

```
00E6           I1State2                          ;Check if still a Start Bit
00E6 0705               BTFSS   Serial_IN_1      ;False Start Error ?
00E7 0B06               GOTO    FS_Error_1
00E8 0409               BCF     FS_Flag_1        ;Start Bit OK
00E9 021A               MOVF    nbti1,W          ;Adjust out the error
00EA 01F9               ADDWF   First_TMR0_I1, F
00EB 0C64               MOVLW   IN_BIT_TIME      ;Time Delay = full bit time
00EC 003A               MOVWF   nbti1
00ED 02B8               INCF    IState1, F       ;increment to next state
00EE 0800               RETLW   H'00'
00EF           I1State0_7                        ;Bit 0 - 7
00EF 0705               BTFSS   Serial_IN_1      ;Move Input bit into C
00F0 0403               BCF     STATUS,C
00F1 0605               BTFSC   Serial_IN_1
00F2 0503               BSF     STATUS,C
00F3 033B               RRF     rcv_byte_1, F    ;Move C into left most bit
00F4 021A               MOVF    nbti1,W
00F5 01F9               ADDWF   First_TMR0_I1, F ;Adjust out the error
00F6 02B8               INCF    IState1, F       ;increment to next state
00F7 0800               RETLW   H'00'
00F8           I1StateE                          ;Check if we have a proper Stop Bit
00F8 0605               BTFSC   Serial_IN_1      ;Frame Error
00F9 0B09               GOTO    F_Error_1
00FA 0429               BCF     FE_Flag_1        ;Stop Bit OK
00FB 006E               CLRF    T_5_S_CO         ;Reset 5 Sec Timer - got a good byte
                        ;Process the msg Here !
00FC 021B               MOVF    rcv_byte_1,W     ;Make a copy of just received byte
00FD 0035               MOVWF   RCV_Storage
00FE 07A8               BTFSS   RCV_Got_One_B    ;Report Lost data
00FF 0489               BCF     RCV_Overflow
0100 06A8               BTFSC   RCV_Got_One_B
0101 0589               BSF     RCV_Overflow
0102 05A8               BSF     RCV_Got_One_B    ;We Now have a RB Value to go out
0103           I1StateL
0103 0078               CLRF    IState1          ;Ready to receive next byte
0104 0428               BCF     IState1_B        ;Serial In not currently active
0105 0800               RETLW   H'00'
0106           FS_Error_1                        ;False Start - Shut Down Checking
0106 0428               BCF     IState1_B        ;Serial Input NOT Active
0107 0509               BSF     FS_Flag_1        ;False Start Error
0108 0B03               GOTO    I1StateL         ;Start All Over
0109           F_Error_1                         ;Frame Error - Wait for End of Stop Bit
0109 021A               MOVF    nbti1,W          ;Adjust out the error
010A 01F9               ADDWF   First_TMR0_I1, F
010B 0C32               MOVLW   H'32'            ;Time Delay = 1/2 bit time
010C 003A               MOVWF   nbti1
010D 0529               BSF     FE_Flag_1        ;Frame Error for this Byte ?
010E 02B8               INCF    IState1, F       ;Increment to next state
010F 0800               RETLW   H'00'

               ;***              ;Subroutines for Task #3
0110           I2StateS                          ;Start Bit - Setup timing variables
0110 0548               BSF     IState2_B        ;Serial Input Active
0111 0201               MOVF    TMR0,W           ;Store starting time
0112 003D               MOVWF   First_TMR0_I2
0113 0C0D               MOVLW   H'0D'            ;Fudge again
0114 00BD               SUBWF   First_TMR0_I2, F
0115 0C32               MOVLW   H'32'            ;Time delay = 1/2 bit time
0116 003E               MOVWF   nbti2
0117 02BC               INCF    IState2, F       ;Increment to next state
0118 0800               RETLW   H'00'
0119           I2StateS2                         ;Check if still a Start Bit
0119 0765               BTFSS   Serial_IN_2      ;False Start Error ?
011A 0B39               GOTO    FS_Error_2
011B 0449               BCF     FS_Flag_2        ;Start Bit OK
011C 021E               MOVF    nbti2,W          ;Adjust out the error
```

```
011D 01FD                   ADDWF   First_TMR0_I2, F
011E 0C64                   MOVLW   IN_BIT_TIME     ;Time Delay = full bit time
011F 003E                   MOVWF   nbti2
0120 02BC                   INCF    IState2, F      ;increment to next state
0121 0800                   RETLW   H'00'
0122              I2State0_7                        ;Bit 0 - 7
0122 0765                   BTFSS   Serial_IN_2     ;Move Input bit into C
0123 0403                   BCF     STATUS,C
0124 0665                   BTFSC   Serial_IN_2
0125 0503                   BSF     STATUS,C
0126 033F                   RRF     rcv_byte_2, F   ;Move C into left most bit
0127 021E                   MOVF    nbti2,W
0128 01FD                   ADDWF   First_TMR0_I2, F ;Adjust out the error
0129 02BC                   INCF    IState2, F       ;increment to next state
012A 0800                   RETLW   H'00'
012B              I2StateE                          ;Check if we have a proper Stop Bit
012B 0665                   BTFSC   Serial_IN_2     ;Frame Error
012C 0B3C                   GOTO    F_Error_2
012D 0469                   BCF     FE_Flag_2       ;Stop Bit OK
012E 006E                   CLRF    T_5_S_CO        ;Reset 5 Sec Timer - got a good byte
                            ;Process the msg Here !
012F 021F                   MOVF    rcv_byte_2,W    ;Make a copy of just received byte
0130 0035                   MOVWF   RCV_Storage
0131 07A8                   BTFSS   RCV_Got_One_B   ;Report Lost data
0132 0489                   BCF     RCV_Overflow
0133 06A8                   BTFSC   RCV_Got_One_B
0134 0589                   BSF     RCV_Overflow
0135 05A8                   BSF     RCV_Got_One_B   ;We Now have a RB Value to go out
0136              I2StateL
0136 007C                   CLRF    IState2         ;Ready to receive next byte
0137 0448                   BCF     IState2_B       ;Serial In not currently active
0138 0800                   RETLW   H'00'
0139              FS_Error_2
0139 0448                   BCF     IState2_B       ;False Start - Shut Down Checking
013A 0549                   BSF     FS_Flag_2       ;False Start Error
013B 0B36                   GOTO    I2StateL        ;Start All Over
013C              F_Error_2                         ;Frame Error - Wait for End of Stop Bit
013C 021E                   MOVF    nbti2,W         ;Adjust out the error
013D 01FD                   ADDWF   First_TMR0_I2, F
013E 0C32                   MOVLW   H'32'           ;Time Delay = 1/2 bit time
013F 003E                   MOVWF   nbti2
0140 0569                   BSF     FE_Flag_2       ;Frame Error for this Byte ?
0141 02BC                   INCF    IState2, F      ;Increment to next state
0142 0800                   RETLW   H'00'

                  ;******           ;Code Starting point
0143              Main
0143 0C00                   MOVLW   H'00'         ;What is High/Low for RA at INIT State
0144 0025                   MOVWF   PORTA
0145 0C00                   MOVLW   H'00'         ;What is High/Low for RB at INIT State
0146 0026                   MOVWF   PORTB
0147 0CF9                   MOVLW   H'F9'         ;RA TRIS at INIT State
0148 0005                   TRIS    5
0149 0CFF                   MOVLW   H'FF'         ;RB TRIS at INIT State
014A 0006                   TRIS    6
014B 0C00                   MOVLW   H'00'         ;TMR0/2
014C 0002                   OPTION
014D 0900                   CALL    Clear_Regs    ;Clear Registers 7-1F - Same Memory Page
014E 0061                   CLRF    TMR0          ;Start timers

                  ;Initialize Tasks
                                                  ;Task #1 waits for byte to output
                                                  ;Task #2 waits for Serial IN Start Bit
                                                  ;Task #3 waits for Serial IN Start Bit
                                                  ;Task #4 runs when Task 1 is Not
                                                  ;Task #5 is always running
```



```
014F 0206           00514         MOVF    PORTB,W         ;Task #6 is Initialized here
0150 0036           00515         MOVWF   Old_RB
0151 0216           00516         MOVF    Old_RB,W        ;Make all the same initial value
0152 0037           00517         MOVWF   Last_RB
0153 05C8           00518         BSF     RB_NEW_B        ;Tell Task #4: RB byte ready to output
0154 0C08           00519         MOVLW   LED_OFF_MODE
0155 002A           00520         MOVWF   LED_Mode        ;Task #7 is Started
0156 0568           00521         BSF     T_5_S_B         ;Task #8 is Started here
0157 0588           00522         BSF     T_5_M_B         ;Task #9 is Started here
                    00523
                    00524 ; Handle Task & Timer activities - Main Loop
0158                00525 Task_1  ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0158 0708           00526         BTFSS   OState_B        ;if not outputing now then skip call
0159 0B5B           00527         GOTO    Task_2
015A 0902           00528         CALL    Do_OState       ;Go Do Task #1
                    00529
015B                00530 Task_2  ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
015B 0628           00531         BTFSC   IState1_B       ;if already started then call
015C 0B60           00532         GOTO    _0053
015D 0605           00533         BTFSC   Serial_IN_1     ;if Start bit ? then call
015E 0B60           00534         GOTO    _0053
015F 0B61           00535         GOTO    Task_3
0160 09A7           00536 _0053   CALL    Do_I1State      ;Go Do Task #2
                    00537
0161                00538 Task_3 ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
0161 0648           00539         BTFSC   IState2_B       ;if already started then call
0162 0B66           00540         GOTO    _0055
0163 0665           00541         BTFSC   Serial_IN_2     ;if Start bit ? then call
0164 0B66           00542         GOTO    _0055
0165 0B67           00543         GOTO    Task_4
0166 09C2           00544 _0055   CALL    Do_I2State      ;Go Do Task #3
                    00545
0167                00546 Task_4  ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0167 0608           00547         BTFSC   OState_B        ;if outputing now then skip call
0168 0B7D           00548         GOTO    _0059
0169 07A8           00549         BTFSS   RCV_Got_One_B   ;Got a NEW Received byte to send
016A 0B70           00550         GOTO    _0057
016B 0215           00551         MOVF    RCV_Storage,W   ;Send just received byte
016C 0033           00552         MOVWF   xmt_byte
016D 04A8           00553         BCF     RCV_Got_One_B   ;Clear need to send old byte
016E 0508           00554         BSF     OState_B        ;Start Task #1 & Lock Out Others
016F 0B7D           00555         GOTO    _0059
0170 07C8           00556 _0057   BTFSS   RB_NEW_B        ;Indicates a change in RB input
0171 0B77           00557         GOTO    _0058
0172 0216           00558         MOVF    Old_RB,W        ;Send New RB value
0173 0033           00559         MOVWF   xmt_byte
0174 04C8           00560         BCF     RB_NEW_B        ;Clear need to send out newest value
0175 0508           00561         BSF     OState_B        ;Start Task #1 & Lock Out Others
0176 0B7D           00562         GOTO    _0059
0177 07E8           00563 _0058   BTFSS   S_5_S_B         ;Serial In 5 secs of inactivity
0178 0B7D           00564         GOTO    _0059
0179 0CFF           00565         MOVLW   H'FF'           ;Tell of inactivity of Serial In
017A 0033           00566         MOVWF   xmt_byte
017B 04E8           00567         BCF     S_5_S_B         ;Clear need to send msg
017C 0508           00568         BSF     OState_B        ;Start Task #1 & Lock Out Others
                    00569
                    00570        ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
017D 0201           00571 _0059   MOVF    TMR0,W          ;Step-up time units * 512
017E 0027           00572         MOVWF   temp
017F 0211           00573         MOVF    Last_TMR0,W     ;Test to see if it overflowed
0180 0087           00574         SUBWF   temp,W
0181 0703           00575         BTFSS   STATUS,C
0182 0B86           00576         GOTO    Inc_Time
0183 0207           00577         MOVF    temp,W          ;unit error = < |+-512 uS|
0184 0031           00578         MOVWF   Last_TMR0
0185 0B58           00579         GOTO    Task_1
```

```
0186                Inc_Time
0186 0207                   MOVF    temp,W          ;Save current TMR0 into Last_TMR0
0187 0031                   MOVWF   Last_TMR0

0188                Task_5  ;Task #5 - Monitor Level Reset Input Line - Always Running !
0188 0606                   BTFSC   Level_Reset
0189 0B8C                   GOTO    Task_6
018A 0C08                   MOVLW   LED_OFF_MODE    ;Lowest Level Indicator output
018B 002A                   MOVWF   LED_Mode

018C                Task_6  ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
018C 02AF                   INCF    T_20_mS_CO, F   ;Inc Counter - Time Unit = 512 uS
018D 0C27                   MOVLW   H'27'           ;Used to debounce the input
018E 008F                   SUBWF   T_20_mS_CO,W
018F 0743                   BTFSS   STATUS,Z
0190 0BA7                   GOTO    _0065
0191 006F                   CLRF    T_20_mS_CO      ;Reset T_20_mS_CO to start over again

0192 0997                   CALL    QCheck_T123     ;Quick Check of Tasks #1, #2 and #3

0193 0206                   MOVF    PORTB,W         ;Last copy of RB same as Current ?
0194 0097                   SUBWF   Last_RB,W
0195 0643                   BTFSC   STATUS,Z
0196 0B9A                   GOTO    _0062
0197 0206                   MOVF    PORTB,W         ;Store Current RB - diff from Last
0198 0037                   MOVWF   Last_RB
0199 0B9C                   GOTO    _0063
019A 0217           _0062   MOVF    Last_RB,W       ;New Old RB <- same value over 20 mS
019B 0036                   MOVWF   Old_RB
019C 0236           _0063   MOVF    Old_RB, F       ;See if RB is now 0
019D 0643                   BTFSC   STATUS,Z        ;RB == 0 ? then keep timer running
019E 0BA1                   GOTO    _0064
019F 006C                   CLRF    T_5_M_LO        ;Reset 5 Min Timer
01A0 006D                   CLRF    T_5_M_HI        ;  still not zero yet
01A1 0901           _0064   CALL    D_H_E_L         ;Determine the Highest Error Level
01A2 07C8                   BTFSS   RB_NEW_B        ;Check for Lost Data Error
01A3 04A9                   BCF     RB_Overflow
01A4 06C8                   BTFSC   RB_NEW_B
01A5 05A9                   BSF     RB_Overflow
01A6 05C8                   BSF     RB_NEW_B        ;Every 20 mS send Old_RB out

                           ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
01A7 0CF9           _0065   MOVLW   H'F9'           ;RA TRIS - refresh
01A8 0005                   TRIS    5
01A9 0CFF                   MOVLW   H'FF'           ;RB TRIS - refresh
01AA 0006                   TRIS    6
01AB 02F4                   DECFSZ  cc, F           ;Step-up time units * 256
01AC 0B58                   GOTO    Task_1

01AD                Task_7 ;Task 7 - Output Highest Level Indication on LED
01AD 076A                   BTFSS   LED_B           ;Is LED active ?
01AE 0BB1                   GOTO    Task_8

01AF 0997                   CALL    QCheck_T123     ;Quick Check of Tasks #1, #2 and #3

01B0 0939                   CALL    Do_LED          ;Handle LED timing

01B1                Task_8 ;Task #8 - 5 Second Serial Input Lack of Activity Timer
01B1 0768                   BTFSS   T_5_S_B         ;5 Sec Timer Active ?
01B2 0BC0                   GOTO    Task_9
01B3 02AE                   INCF    T_5_S_CO, F     ;Inc Counter - Time Unit = 131072 uS
01B4 0C26                   MOVLW   H'26'           ;Check T_5_S_CO if time
01B5 008E                   SUBWF   T_5_S_CO,W
01B6 0743                   BTFSS   STATUS,Z
01B7 0BC0                   GOTO    Task_9
01B8 006E                   CLRF    T_5_S_CO        ;Reset T_5_S_CO
```

```
01B9 0C8F            00646         MOVLW   LED_ON_MODE     ;Highest Level Indicator output
01BA 002A            00647         MOVWF   LED_Mode
01BB 07E8            00648         BTFSS   S_5_S_B         ;Check if Lost Data Error
01BC 04C9            00649         BCF     S_5_S_Overflow
01BD 06E8            00650         BTFSC   S_5_S_B
01BE 05C9            00651         BSF     S_5_S_Overflow
01BF 05E8            00652         BSF     S_5_S_B         ;Send notice of 5 seconds of inaction
                     00653
01C0                 00654 Task_9 ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
01C0 0788            00655         BTFSS   T_5_M_B         ;5 Min Timer Active ?
01C1 0BD1            00656         GOTO    Task_A
01C2 02AC            00657         INCF    T_5_M_LO, F     ;Inc LO Counter; Time Unit = 131072 uS
01C3 0643            00658         BTFSC   STATUS,Z        ;See if carry needs to be passed on ?
01C4 02AD            00659         INCF    T_5_M_HI, F     ;Inc HI Counter; Time Unit = 131072 uS
01C5 0C08            00660         MOVLW   H'08'           ;#2288<  Check T_5_M_HI if time
01C6 008D            00661         SUBWF   T_5_M_HI,W
01C7 0743            00662         BTFSS   STATUS,Z
01C8 0BD1            00663         GOTO    Task_A
01C9 0CF0            00664         MOVLW   H'F0'           ;#2288>  Check T_5_M_LO if time
01CA 008C            00665         SUBWF   T_5_M_LO,W
01CB 0743            00666         BTFSS   STATUS,Z
01CC 0BD1            00667         GOTO    Task_A
01CD 006C            00668         CLRF    T_5_M_LO        ;Reset T_5_M_LO
01CE 006D            00669         CLRF    T_5_M_HI        ;Reset T_5_M_HI
01CF 0C08            00670         MOVLW   LED_OFF_MODE    ;Lowest Level Indicator output
01D0 002A            00671         MOVWF   LED_Mode
01D1                 00672 Task_A
01D1 0B58            00673         GOTO    Task_1          ;Loop Forever
                     00674
                     00675 ;****
01D2                 00676 Do_D_H_E_L      ; Determine the Highest Error Level & Start Task #7
01D2 0C07            00677         MOVLW   H'07'           ;Check top 7 bits
01D3 0027            00678         MOVWF   temp
01D4 0216            00679         MOVF    Old_RB,W        ;Get copy of 7 debounced Sensor Input
01D5 0037            00680         MOVWF   Last_RB
01D6 0377            00681 _0070   RLF     Last_RB, F      ;Put top bit into C bit
01D7 0603            00682         BTFSC   STATUS,C        ;Check if C bit is set
01D8 0BDE            00683         GOTO    _0072
01D9 02E7            00684         DECFSZ  temp, F         ;Continue to check lesser bits
01DA 0BD6            00685         GOTO    _0070
01DB 0206            00686 _0071   MOVF    PORTB,W         ;Restore current value of RB
01DC 0037            00687         MOVWF   Last_RB
01DD 0800            00688         RETLW   H'00'
01DE 020A            00689 _0072   MOVF    LED_Mode,W      ;Get current Level Indicator
01DF 0E07            00690         ANDLW   H'07'           ;Get only      "       "
01E0 0037            00691         MOVWF   Last_RB         ;Store into a temporary register
01E1 0207            00692         MOVF    temp,W          ;Check if already at this Level
01E2 0097            00693         SUBWF   Last_RB,W
01E3 0603            00694         BTFSC   STATUS,C
01E4 0BDB            00695         GOTO    _0071
01E5 0C88            00696         MOVLW   H'88'           ;Start to build LED_Mode
01E6 0107            00697         IORWF   temp,W          ;Put new Level Indicator into reg
01E7 002A            00698         MOVWF   LED_Mode        ;Store new LED Mode
01E8 0BDB            00699         GOTO    _0071
                     00700
01E9                 00701 Do_Clear_Regs   ; Clear Registers 7-1Fh
01E9 0C1F            00702         MOVLW   H'1F'           ;First regs to clear
01EA 0024            00703         MOVWF   FSR
01EB 0060            00704 Loop_C  CLRF    INDIR           ;Clear reg
01EC 00E4            00705         DECF    FSR, F          ;point to next reg to clear
01ED 0CE7            00706         MOVLW   H'E7'           ;Dec temp, jump if not done
01EE 0084            00707         SUBWF   FSR,W
01EF 0603            00708         BTFSC   STATUS,C
01F0 0BEB            00709         GOTO    Loop_C
01F1 0064            00710         CLRF    FSR             ;Lastly clear FSR reg
01F2 0800            00711         RETLW   H'00'
```



```
                    00712
01FF                00713         ORG     H'1FF'          ;RESET to Main
01FF 0B43           00714         GOTO    Main
                    00715
                    00716         END
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 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX------------X

All other memory blocks unused.

Program Memory Words Used:   500
Program Memory Words Free:    12


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D:

```
MPASM 01.40 Released            APP_D.ASM   1-16-1997  17:10:05         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                    00001         list    p=16C64,t=ON,c=132
                    00002 ;
                    00003 ;*****************************************************************
                    00004 ;
                    00005 ; 'Remote Alarm64' V1.00
                    00006 ;   Designed by Myriad Development Co. - Jerry Farmer
                    00007 ;   PIC16C64, 4MHz Crystal, WatchDog Timer OFF, MPASM instruction set
                    00008 ;   PIC16C54, 4MHz Crystal, WatchDog Timer OFF
                    00009 ;       Program:          APP_D.ASM
                    00010 ;       Revision Date:
                    00011 ;                         1-15-97   Compatibility with MPASMWIN 1.40
                    00012 ;
                    00013 ;*****************************************************************
                    00014 ;
                    00015
                    00016         include "P16C64.INC"
                    00001         LIST
                    00002 ;P16C64.INC Standard Header File,Ver. 1.01 Microchip Technology, Inc.
                    00238         LIST
                    00017
                    00018 ; B Register Definitions
  00000000          00019 Serial_IN_1    equ  0           ;Serial Input #1 - 8 bits - INT pin
                    00020                                 ;RB.7 - RB.1 == Input from Sensors
  000000FF          00021 RB_TRIS        equ  B'11111111' ;RB TRIS at INIT State == all input
  00000000          00022 RB_MASK        equ  B'00000000' ;What is High/Low for RB at INIT State
                    00023
                    00024 ; A Register Definitions - Programmable Inputs
  00000000          00025 Level_Reset    equ  0       ;PORTA.0 - Reset Level Indicator
  00000001          00026 LED            equ  1       ;LED Output - Level/State Indicator
  00000002          00027 Serial_Out     equ  2       ;Serial Output - 8 bits + passwords
  00000003          00028 PWM_Out        equ  3       ;PWM Output - 8 bits ON, 8 bits OFF
                    00029
  000000F1          00030 RA_TRIS        equ  B'11110001'  ;RA TRIS at INIT State
  00000000          00031 RA_MASK        equ  B'00000000' ;What is High/Low for RA at INIT State
                    00032
                    00033 ; Register Files
  00000020          00034 temp           equ  20h  ;Temporary holding register - PIC16C54/56
  00000021          00035 tmp            equ  21h  ;Temporary reg
  00000022          00036 Temp_W         equ  22h  ;Interrupt storage of W
  00000023          00037 Temp_Stat      equ  23h  ;Interrupt storage of STATUS
  00000024          00038 Temp_FSR       equ  24h  ;Interrupt storage of FSR
  00000025          00039 T_B            equ  25h  ;Indicates which Timer(s) are Active = 1
  00000026          00040 FLAGS          equ  26h  ;Error Flags
  00000027          00041 LED_Mode       equ  27h  ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
  00000028          00042 OState         equ  28h  ;Serial Out State
  00000029          00043 IState1        equ  29h  ;Serial In #1 State
  0000002A          00044 cc             equ  2Ah  ;256 * TMR0 time
  0000002B          00045 T_5_M_LO       equ  2Bh  ;5 Min Timer Counter - Low
  0000002C          00046 T_5_M_HI       equ  2Ch  ;5 Min Timer Counter - High
  0000002D          00047 T_5_S_CO       equ  2Dh  ;5 Second Timer - lack of Serial Input
  0000002E          00048 T_20_mS_CO     equ  2Eh  ;20 mS Timer - used for debouncing
  0000002F          00049 T_PWM_CO       equ  2Fh  ;PWM Counter
  00000030          00050 LED_C          equ  30h  ;LED Counter
  00000031          00051 xmt_byte       equ  31h  ;Serial xmit byte - destroyed in use
```

```
  00000032          00052 rcv_byte_1     equ  32h  ;Receive Serial #1 In byte
  00000033          00053 RCV_Storage    equ  33h  ;Long term storage of rcv_byte #1
  00000034          00054 Old_RB         equ  34h  ;Oldest/Master copy of RB
  00000035          00055 Last_RB        equ  35h  ;Last copy of RB
  00000036          00056 PWM_ON         equ  36h  ;PWM ON Counter
  00000037          00057 PWM_OFF        equ  37h  ;PWM OFF Counter
  00000038          00058 PWM_tmp        equ  38h  ;PWM temporary counter
                    00059
                    00060 ; Indicates which Timer(s) are Active = 1 & Flags - T_B
  00000000          00061 OState_B       equ  0    ;Serial Out Active Bit
  00000001          00062 IState1_B      equ  1    ;Serial IN #1 Active Bit
  00000002          00063 T_5_S_B        equ  2    ;5 Second Timer Active Bit
  00000003          00064 T_5_M_B        equ  3    ;5 Min Timer Active Bit
  00000004          00065 RCV_Got_One_B  equ  4    ;Got a NEW Received byte to send out
  00000005          00066 RB_NEW_B       equ  5    ;Indicates a change in RB input
  00000006          00067 S_5_S_B        equ  6    ;Serial In 5 secs of inactivity
  00000007          00068 T_PWM_B        equ  7    ;PWM Activity Bit
                    00069
                    00070 ; Error Flags - FLAGS
  00000000          00071 FS_Flag_1      equ  0    ;Serial #1 IN had a False Start Error
  00000001          00072 FE_Flag_1      equ  1    ;Last Serial #1 IN had a Frame Error
  00000002          00073 RCV_Overflow   equ  2    ;Lost Serial Input Byte - too Slow
  00000003          00074 RB_Overflow    equ  3    ;Lost RB Input Byte - too Slow
  00000004          00075 S_5_S_Overflow equ  4    ;Lost '5S Inactivity' msg - too Slow
  00000005          00076 Time_Bit       equ  5    ;Indicate 512 uS has passed
                    00077
                    00078 ;Equates for LED Task #7 - LED_Mode
  00000003          00079 LED_B           equ  3               ;LED is active - LED_Mode.3
  00000007          00080 LED_NEW_B       equ  7               ;LED has just changed Modes = 1
  00000008          00081 LED_OFF_MODE    equ  B'00001000'     ;LED OFF
  00000089          00082 LED_SEQ1_MODE   equ  B'10001001'     ;LED Sequence 1: .2s On, 1s Off
  0000008A          00083 LED_SEQ2_MODE   equ  B'10001010'     ;LED Sequence 2: 3x(.2s), 1s Off
  0000008B          00084 LED_SEQ3_MODE   equ  B'10001011'     ;LED Sequence 3: 5x(.2s), 1s Off
  0000009C          00085 LED_SLOW_MODE   equ  B'10011100'     ;LED Slow Pulsing - .3 Hz
  0000009D          00086 LED_MEDIUM_MODE equ  B'10011101'     ;LED Medium Pulsing - 1 Hz
  0000009E          00087 LED_FAST_MODE   equ  B'10011110'     ;LED Fast Pulsing - 3 Hz
  0000008F          00088 LED_ON_MODE     equ  B'10001111'     ;LED ON Continuously
                    00089
0000                00090         ORG     0                    ;Reset Vector
                    00091
0000 28F7           00092         GOTO    Main
                    00093
0004                00094         ORG     4                    ;Interrupt Vector
                    00095
0004 29D4           00096         GOTO    Interrupt
                    00097
                    00098 ;******         Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0005                00099 Do_OState
0005 0828           00100         MOVF    OState,W             ;Get (0-A) mode #
0006 390F           00101         ANDLW   H'0F'                ;Get only mode #
0007 0782           00102         ADDWF   PCL, F               ;jump to subroutine
0008 2814           00103         GOTO    OStateS              ;Serial Start Bit
0009 2823           00104         GOTO    OState0_7            ;Bit 0
000A 2823           00105         GOTO    OState0_7            ;Bit 1
000B 2823           00106         GOTO    OState0_7            ;Bit 2
000C 2823           00107         GOTO    OState0_7            ;Bit 3
000D 2823           00108         GOTO    OState0_7            ;Bit 4
000E 2823           00109         GOTO    OState0_7            ;Bit 5
000F 2823           00110         GOTO    OState0_7            ;Bit 6
0010 2823           00111         GOTO    OState0_7            ;Bit 7
0011 2829           00112         GOTO    OStateE              ;Serial Stop Bit
0012 2831           00113         GOTO    OStateL              ;Last State
0013 0008           00114         RETURN
                    00115
0014                00116 OStateS
0014 3000           00117         MOVLW   H'00'                ;Post & Pre 1=1 & OFF
```

```
0015 0092           00118       MOVWF   T2CON
0016 1683           00119       BSF     STATUS,RP0      ;Point to BANK 1
0017 3053           00120       MOVLW   H’68’ - H’15’   ;104uS - 9600 Baud & adjust to Latency
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0018 0092           00121       MOVWF   PR2
0019 1283           00122       BCF     STATUS,RP0      ;Point to BANK 0
001A 0191           00123       CLRF    TMR2            ;Init to 0
001B 108C           00124       BCF     PIR1,TMR2IF     ;Clear Timer 2 Flag so as to start fresh
001C 1505           00125       BSF     PORTA,Serial_Out;Output ‘Serial Start Bit’ starting Now
001D 1512           00126       BSF     T2CON,TMR2ON    ;Start Timer 2
001E 0AA8           00127       INCF    OState, F  ;inc to next state BEFORE allowing interrupt
                    00128
001F 1683           00129       BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0020 148C           00130       BSF     PIE1,TMR2IE     ;Allow for Timer 2 interrupts
0021 1283           00131       BCF     STATUS,RP0      ;Point to BANK 0
0022 0008           00132       RETURN
                    00133
0023                00134 OState0_7                       ;Bit 0 - 7
0023 0CB1           00135       RRF     xmt_byte, F       ;Move bit into C from right most bit
0024 1C03           00136       BTFSS   STATUS,C          ;Output C bit
0025 1105           00137       BCF     PORTA,Serial_Out  ;
0026 1803           00138       BTFSC   STATUS,C          ;
0027 1505           00139       BSF     PORTA,Serial_Out  ;
0028 282A           00140       GOTO    OS_End
0029                00141 OStateE
0029 1105           00142         BCF     PORTA,Serial_Out  ;Serial Stop Bit
002A 108C           00143 OS_End  BCF     PIR1,TMR2IF   ;Clear Timer 2 Flag so as to start fresh
002B 1683           00144         BSF     STATUS,RP0    ;Point to BANK 1
002C 306C           00145         MOVLW   H’68’ + H’4   ;104uS - 9600 Baud & adjust to Latency
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
002D 0092           00146         MOVWF   PR2
002E 1283           00147         BCF     STATUS,RP0    ;Point to BANK 0
002F 0AA8           00148         INCF    OState, F     ;increment to next state
0030 0008           00149         RETURN
0031                00150 OStateL
0031 1683           00151         BSF     STATUS,RP0    ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0032 108C           00152         BCF     PIE1,TMR2IE   ;Do NOT Allow Timer 2 interrupts
0033 1283           00153         BCF     STATUS,RP0    ;Point to BANK 0
                    00154
0034 108C           00155         BCF     PIR1,TMR2IF   ;Clear Timer 2 Flag so as to start fresh
0035 1112           00156         BCF     T2CON,TMR2ON  ;Stop Timer 2
0036 01A8           00157         CLRF    OState        ;Ready to send next byte out
0037 1025           00158         BCF     T_B,OState_B  ;Serial Out not active
0038 0008           00159         RETURN
                    00160
                    00161 ;******  Task #7 - Output Highest Level Indication on LED
0039                00162 Do_LED
0039 1BA7           00163         BTFSC   LED_Mode,LED_NEW_B;Initialize regs if change in modes
003A 284C           00164         GOTO    LED_NEW
003B 0AB0           00165         INCF    LED_C, F        ;Inc Counter - Time Unit = 131072 uS
003C 0827           00166         MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 3907           00167         ANDLW   H’07’           ;Get only mode #
003E 0782           00168         ADDWF   PCL, F          ;jump to subroutine
003F 2848           00169         GOTO    LED_OFF         ;LED OFF
0040 2864           00170         GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 2867           00171         GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 288A           00172         GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 2850           00173         GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 285E           00174         GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 2861           00175         GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 284D           00176         GOTO    LED_ON          ;LED ON Continuously
0047 0008           00177 _0012   RETURN
                    00178 ;------
0048                00179 LED_OFF
```

```
0048 1085             BCF     PORTA,LED         ;Turn off LED
0049 11A7             BCF     LED_Mode,LED_B    ;LED must be off
004A 01B0             CLRF    LED_C             ;Reset Counter - LED_C = 0
004B 0008             RETURN
           ;------
004C       LED_NEW
004C 13A7             BCF     LED_Mode,LED_NEW_B  ;Done initializing
004D       LED_ON
004D 1485             BSF     PORTA,LED         ;Turn on LED
004E 01B0             CLRF    LED_C             ;Reset Counter - LED_C = 0
004F 0008             RETURN
           ;------
0050       LED_SLOW
0050 300C             MOVLW   H'0C'             ;.3Hz @ 50% Duty
0051 00A0             MOVWF   temp
0052 0820  LED_S      MOVF    temp,W            ;Check LED_C if time, .3Hz @ 50% Duty
0053 0230             SUBWF   LED_C,W
0054 1D03             BTFSS   STATUS,Z
0055 2847             GOTO    _0012
0056 3010             MOVLW   H'10'
0057 06A7             XORWF   LED_Mode, F       ;Switch states
0058 1E27             BTFSS   LED_Mode,4        ;Now make LED same state
0059 1085             BCF     PORTA,LED
005A 1A27             BTFSC   LED_Mode,4
005B 1485             BSF     PORTA,LED
005C 01B0             CLRF    LED_C             ;Reset LED_C
005D 0008             RETURN
           ;------
005E       LED_MEDIUM
005E 3004             MOVLW   H'04'             ;1Hz @ 50% Duty
005F 00A0             MOVWF   temp
0060 2852             GOTO    LED_S             ;Go do it
           ;------
0061       LED_FAST
0061 3001             MOVLW   H'01'             ;3Hz @ 50% Duty
0062 00A0             MOVWF   temp
0063 2852             GOTO    LED_S             ;Go do it
           ;------
0064       LED_SEQ1                             ;.2 ON, 1 OFF
0064 1E27             BTFSS   LED_Mode,4        ;Skip if bit is high
0065 2876             GOTO    ON1               ;Go do it
0066 2882             GOTO    OFF3              ;Go do it
           ;------
0067       LED_SEQ2                             ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 0827             MOVF    LED_Mode,W
0068 00A0             MOVWF   temp
0069 3030             MOVLW   H'30'             ;Get sequence # only
006A 05A0             ANDWF   temp, F
006B 0EA0             SWAPF   temp, F           ;swap nibbles
006C 0820             MOVF    temp,W            ;get nibble for offset
006D 0782             ADDWF   PCL, F            ;Table jump calculation
006E 2876             GOTO    ON1               ;LED is on, check if time to change
006F 287C             GOTO    OFF2              ;LED is off, check if time to change
0070 2876             GOTO    ON1               ;LED is on, check if time to change
0071 2882             GOTO    OFF3              ;LED is off, check if time to change
           ;------
0072       LED_Exit
0072 3010             MOVLW   H'10'             ;Inc Seq #
0073 07A7             ADDWF   LED_Mode, F
0074 01B0             CLRF    LED_C             ;Reset LED_C
0075 0008             RETURN
0076       ON1
0076 3002             MOVLW   H'02'             ;Check LED_C if time, .2 sec-on
0077 0230             SUBWF   LED_C,W
0078 1D03             BTFSS   STATUS,Z
0079 2847             GOTO    _0012
```

```
007A 1085                     BCF     PORTA,LED       ;Turn off LED
007B 2872                     GOTO    LED_Exit
007C                    OFF2
007C 3002                     MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0230                     SUBWF   LED_C,W
007E 1D03                     BTFSS   STATUS,Z
007F 2847                     GOTO    _0012
0080 1485                     BSF     PORTA,LED       ;Turn on LED
0081 2872                     GOTO    LED_Exit
0082                    OFF3
0082 3008                     MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0230                     SUBWF   LED_C,W
0084 1D03                     BTFSS   STATUS,Z
0085 2847                     GOTO    _0012
0086 1485                     BSF     PORTA,LED       ;Turn on LED
0087 30F0                     MOVLW   H'F0'
0088 04A7                     IORWF   LED_Mode, F     ;Cause (Seq# & NEW) to overflow to 0
0089 2872                     GOTO    LED_Exit
008A                    LED_SEQ3    ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 0827                     MOVF    LED_Mode,W      ;Get LED info
008B 00A0                     MOVWF   temp
008C 3070                     MOVLW   H'70'           ;Get sequence # only
008D 05A0                     ANDWF   temp, F
008E 0EA0                     SWAPF   temp, F         ;swap nibbles
008F 0820                     MOVF    temp,W          ;get nibble for offset
0090 0782                     ADDWF   PCL, F          ;Table jump calculation
0091 2876                     GOTO    ON1             ;LED is on check if time to change
0092 287C                     GOTO    OFF2            ;LED is off check if time to change
0093 2876                     GOTO    ON1             ;LED is on check if time to change
0094 287C                     GOTO    OFF2            ;LED is off check if time to change
0095 2876                     GOTO    ON1             ;LED is on check if time to change
0096 2882                     GOTO    OFF3            ;LED is off check if time to change

                        ;******       Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
0097                    Do_I1State
0097 0829                     MOVF    IState1,W       ;Get (0-B) mode #
0098 390F                     ANDLW   H'0F'           ;Get only mode #
0099 0782                     ADDWF   PCL, F          ;jump to subroutine
009A 28A7                     GOTO    I1StateS        ;Serial Start Bit
009B 28B8                     GOTO    I1State2        ;1/2 of Start Bit - see if False Start
009C 28C5                     GOTO    I1State0_7      ;Bit 0
009D 28C5                     GOTO    I1State0_7      ;Bit 1
009E 28C5                     GOTO    I1State0_7      ;Bit 2
009F 28C5                     GOTO    I1State0_7      ;Bit 3
00A0 28C5                     GOTO    I1State0_7      ;Bit 4
00A1 28C5                     GOTO    I1State0_7      ;Bit 5
00A2 28C5                     GOTO    I1State0_7      ;Bit 6
00A3 28C5                     GOTO    I1State0_7      ;Bit 7
00A4 28D4                     GOTO    I1StateE        ;Serial Stop Bit
00A5 28DF                     GOTO    I1StateL        ;Last State - End of Stop Bit
00A6 0008                     RETURN

                        ;***          Subroutines for Task #2
00A7                    I1StateS                    ;Start Bit - Setup timing variables
00A7 120B                   BCF     INTCON,INTE     ;Disable detecting changes on INT pin
00A8 108B                   BCF     INTCON,INTF     ;Clear Interrupting Flag
00A9 3000                   MOVLW   H'00'           ;Internal Clk, Pre 1=1 & OFF
00AA 0090                   MOVWF   T1CON
00AB 018E                   CLRF    TMR1L     ;Calculate (0 - #) of counts until roll-over
00AC 304A                   MOVLW   H'68' - H'1E' ;208us/2 = 4800 Baud & adjust to Latency
00AD 028E                   SUBWF   TMR1L, F
00AE 018F                   CLRF    TMR1H
00AF 038F                   DECF    TMR1H, F        ;H'FF'
00B0 100C                   BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00B1 1410                   BSF     T1CON,TMR1ON    ;Start Timer 1
00B2 0AA9                   INCF    IState1, F ;inc to next state BEFORE allowing interrupts
```

```
00B3 14A5                BSF     T_B,IState1_B   ;Serial Input Active

00B4 1683                BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
00B5 140C                BSF     PIE1,TMR1IE     ;Allow for Timer 1 interrupts
00B6 1283                BCF     STATUS,RP0      ;Point to BANK 0
00B7 0008                RETURN
00B8               I1State2                      ;Check if still a Start Bit
00B8 1C06                BTFSS   PORTB,Serial_IN_1  ;False Start Error ?
00B9 28E9                GOTO    FS_Error_1
00BA 1026                BCF     FLAGS,FS_Flag_1 ;Start Bit OK
00BB 1010                BCF     T1CON,TMR1ON    ;Stop Timer 1
00BC 018E                CLRF    TMR1L
00BD 30AA                MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00BE 028E                SUBWF   TMR1L, F
00BF 018F                CLRF    TMR1H
00C0 038F                DECF    TMR1H, F        ;H'FF'
00C1 100C                BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00C2 1410                BSF     T1CON,TMR1ON    ;Start Timer 1
00C3 0AA9                INCF    IState1, F      ;increment to next state
00C4 0008                RETURN
00C5               I1State0_7                    ;Bit 0 - 7
00C5 1C06                BTFSS   PORTB,Serial_IN_1  ;Move Input bit into C
00C6 1003                BCF     STATUS,C
00C7 1806                BTFSC   PORTB,Serial_IN_1
00C8 1403                BSF     STATUS,C
00C9 0CB2                RRF     rcv_byte_1, F   ;Move C into left most bit
00CA 1010                BCF     T1CON,TMR1ON    ;Stop Timer 1
00CB 018E                CLRF    TMR1L
00CC 30AA                MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00CD 028E                SUBWF   TMR1L, F
00CE 018F                CLRF    TMR1H
00CF 038F                DECF    TMR1H, F        ;H'FF'
00D0 100C                BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00D1 1410                BSF     T1CON,TMR1ON    ;Start Timer 1
00D2 0AA9                INCF    IState1, F      ;increment to next state
00D3 0008                RETURN
00D4               I1StateE                         ;Check if we have a proper Stop Bit
00D4 1806                BTFSC   PORTB,Serial_IN_1  ;Frame Error
00D5 28EC                GOTO    F_Error_1
00D6 10A6                BCF     FLAGS,FE_Flag_1    ;Stop Bit OK
00D7 01AD                CLRF    T_5_S_CO           ;Reset 5 Sec Timer - got a good byte
                         ;Process the msg Here !
00D8 0832                MOVF    rcv_byte_1,W       ;Make a copy of just received byte
00D9 00B3                MOVWF   RCV_Storage
00DA 1E25                BTFSS   T_B,RCV_Got_One_B  ;Report Lost data
00DB 1126                BCF     FLAGS,RCV_Overflow
00DC 1A25                BTFSC   T_B,RCV_Got_One_B
00DD 1526                BSF     FLAGS,RCV_Overflow
00DE 1625                BSF     T_B,RCV_Got_One_B ;We Now have a RB Value to go out
00DF               I1StateL
00DF 1010                BCF     T1CON,TMR1ON      ;Stop Timer 1
00E0 1683                BSF     STATUS,RP0        ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
00E1 140C                BSF     PIE1,TMR1IE       ;Allow for Timer 1 interrupts
00E2 1283                BCF     STATUS,RP0        ;Point to BANK 0
00E3 100C                BCF     PIR1,TMR1IF       ;Clear Timer 1 Flag so as to start fresh

00E4 01A9                CLRF    IState1           ;Ready to receive next byte
00E5 10A5                BCF     T_B,IState1_B     ;Serial In not currently active

00E6 108B                BCF     INTCON,INTF       ;Clear Interrupting Flag
00E7 160B                BSF     INTCON,INTE       ;Enable detecting changes on INT pin
00E8 0008                RETURN
00E9               FS_Error_1    ;False Start - Shut Down Checking
00E9 10A5                BCF     T_B,IState1_B     ;Serial Input NOT Active
```

```
00EA 1426           00376       BSF     FLAGS,FS_Flag_1    ;False Start Error
00EB 28DF           00377       GOTO    I1StateL           ;Start All Over
00EC                00378 F_Error_1      ;Frame Error - Wait for End of Stop Bit
00EC 14A6           00379       BSF     FLAGS,FE_Flag_1    ;Frame Error for this Byte ?
00ED 0AA9           00380       INCF    IState1, F         ;Increment to next state
00EE 1010           00381       BCF     T1CON,TMR1ON       ;Stop Timer 1
00EF 018E           00382       CLRF    TMR1L
00F0 304A           00383       MOVLW   H'68' - H'1E'      ;208us/2 = 4800 Baud & adjust to Latency
00F1 028E           00384       SUBWF   TMR1L, F
00F2 018F           00385       CLRF    TMR1H
00F3 038F           00386       DECF    TMR1H, F           ;H'FF'
00F4 100C           00387       BCF     PIR1,TMR1IF        ;Clear Timer 1 Flag so as to start fresh
00F5 1410           00388       BSF     T1CON,TMR1ON       ;Start Timer 1
00F6 0008           00389       RETURN
                    00390
                    00391 ;******        Code Starting point
00F7                00392 Main
00F7 0183           00393       CLRF    STATUS
00F8 0184           00394       CLRF    FSR
00F9 0181           00395       CLRF    TMR0               ;Clear Timer0
00FA 3000           00396       MOVLW   H'00'              ;What is High/Low for RA at RESET State
00FB 0085           00397       MOVWF   PORTA
00FC 0086           00398       MOVWF   PORTB
00FD 0087           00399       MOVWF   PORTC
00FE 0088           00400       MOVWF   PORTD
00FF 0089           00401       MOVWF   PORTE
0100 018A           00402       CLRF    PCLATH
0101 3060           00403       MOVLW   H'60'      ;/GIE,PEIE,T0IE,/INTE,/RBIE,/T0IF,/INTF,/RBIF
0102 008B           00404       MOVWF   INTCON
0103 018C           00405       CLRF    PIR1             ;Timer 2 Flag cleared
0104 018E           00406       CLRF    TMR1L
0105 018F           00407       CLRF    TMR1H
0106 0190           00408       CLRF    T1CON            ;Timer 1 OFF until ready for input
0107 0191           00409       CLRF    TMR2
0108 0192           00410       CLRF    T2CON            ;Timer 2 OFF until have byte to output
0109 0193           00411       CLRF    SSPBUF
010A 0194           00412       CLRF    SSPCON
010B 0195           00413       CLRF    CCPR1L
010C 0196           00414       CLRF    CCPR1H
010D 0197           00415       CLRF    CCP1CON
                    00416
010E 1683           00417       BSF     STATUS,RP0       ;Point to BANK 1
010F 3040           00418       MOVLW   H'40'   ;TMR0/2 & Interrupt on Rising edge of INT
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0110 0081           00419       MOVWF   OPTION_REG       ;Load OPTION reg
0111 30F1           00420       MOVLW   RA_TRIS
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0112 0085           00421       MOVWF   TRISA
0113 30FF           00422       MOVLW   H'FF'            ;RB TRIS at RESET State
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0114 0086           00423       MOVWF   TRISB
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0115 0087           00424       MOVWF   TRISC
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0116 0088           00425       MOVWF   TRISD
0117 3007           00426       MOVLW   H'07'            ;PSPMODE=0
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0118 0089           00427       MOVWF   TRISE
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0119 018C           00428       CLRF    PIE1             ;Timer 2 Interrupt disabled
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011A 148E           00429       BSF     PCON,NOT_POR
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011B 0192           00430       CLRF    PR2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011C 0193           00431       CLRF    SSPADD
```

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011D 0194           00432       CLRF    SSPSTAT
                    00433
011E 1283           00434       BCF     STATUS,RP0  ;Point to BANK 0
011F 21BF           00435       CALL    Clear_Regs  ;Clear Registers 20-7F, A0-C0 Memory Pages
                    00436
                    00437 ;Initialize Tasks
                    00438 ;Task #1 waits for byte to output
                    00439 ;Task #2 waits for Serial IN Start Bit
0120 3031           00440       MOVLW   H’31’      ;Task #3 is initialized for square pulses
0121 00B6           00441       MOVWF   PWM_ON          ;   “   25 mS ON
0122 3031           00442       MOVLW   H’31’           ;   “   Period = 50 mS, DS= 50%
0123 00B7           00443       MOVWF   PWM_OFF         ;   “  25 mS OFF
0124 01AF           00444       CLRF    T_PWM_CO        ;   “
0125 0836           00445       MOVF    PWM_ON,W        ;move PWM_tmp,PWM_ON
0126 00B8           00446       MOVWF   PWM_tmp         ; “
0127 1585           00447       BSF     PORTA,PWM_Out   ;Start Outputting ON
0128 17A5           00448       BSF     T_B,T_PWM_B     ;   “
                    00449                               ;Task #4 runs when Task 1 is Not
                    00450                               ;Task #5 is always running
0129 0806           00451       MOVF    PORTB,W         ;Task #6 is Initialized here
012A 00B4           00452       MOVWF   Old_RB
012B 0834           00453       MOVF    Old_RB,W        ;Make all the same initial value
012C 00B5           00454       MOVWF   Last_RB
012D 16A5           00455       BSF     T_B,RB_NEW_B    ;Tell Task #4: RB byte ready to output
012E 3008           00456       MOVLW   LED_OFF_MODE
012F 00A7           00457       MOVWF   LED_Mode        ;Task #7 is Started
0130 1525           00458       BSF     T_B,T_5_S_B     ;Task #8 is Started here
0131 15A5           00459       BSF     T_B,T_5_M_B     ;Task #9 is Started here
                    00460
0132 178B           00461       BSF     INTCON,GIE      ;Enable Global Interrupts
                    00462
                    00463 ; Handle Task & Timer activities - Main Loop done in background
0133                00464 Inc_Time      ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
0133 1EA6           00465       BTFSS   FLAGS,Time_Bit ;Idle Task - wait until 512 uS has gone by
0134 2933           00466       GOTO    Inc_Time
0135 12A6           00467       BCF     FLAGS,Time_Bit ;Reset for next indicator
                    00468                                 ;from TMR0 Interrupt
0136                00469 Task_3       ;Task #3 - PWM, Period = (PWM_ON + PWM_OFF) * 512uS
0136 1FA5           00470       BTFSS   T_B,T_PWM_B       ;if NOT outputting now then skip call
0137 2947           00471       GOTO    Task_4
0138 0AAF           00472       INCF    T_PWM_CO, F       ;Inc count of time
0139 0838           00473       MOVF    PWM_tmp,W         ;cjne T_PWM_CO,PWM_tmp,Task_4
013A 022F           00474       SUBWF   T_PWM_CO,W        ; “
013B 1D03           00475       BTFSS   STATUS,Z          ; “
013C 2947           00476       GOTO    Task_4            ; “
013D 01AF           00477       CLRF    T_PWM_CO          ;Reset timer
013E 1985           00478       BTFSC   PORTA,PWM_Out
013F 2944           00479       GOTO    T3_1
0140 1585           00480       BSF     PORTA,PWM_Out     ;Change Output State
0141 0836           00481       MOVF    PWM_ON,W          ;move PWM_tmp,PWM_ON
0142 00B8           00482       MOVWF   PWM_tmp           ; “
0143 2947           00483       GOTO    Task_4
0144                00484 T3_1
0144 1185           00485       BCF     PORTA,PWM_Out    ;Change Output State
0145 0837           00486       MOVF    PWM_OFF,W        ;mov PWM_tmp,PWM_OFF
0146 00B8           00487       MOVWF   PWM_tmp          ; “
                    00488
0147                00489 Task_4  ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0147 1825           00490       BTFSC   T_B,OState_B    ;if outputting now then skip call
0148 295E           00491       GOTO    Task_5
0149 1E25           00492       BTFSS   T_B,RCV_Got_One_B  ;Got a NEW Received byte to send
014A 2950           00493       GOTO    _0057
014B 0833           00494       MOVF    RCV_Storage,W      ;Send just received byte
014C 00B1           00495       MOVWF   xmt_byte
014D 1225           00496       BCF     T_B,RCV_Got_One_B  ;Clear need to send old byte
```

```
014E 1425                   BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
014F 295D                   GOTO    T4_S
0150 1EA5             _0057 BTFSS   T_B,RB_NEW_B       ;Indicates a change in RB input
0151 2957                   GOTO    _0058
0152 0834                   MOVF    Old_RB,W           ;Send New RB value
0153 00B1                   MOVWF   xmt_byte
0154 12A5                   BCF     T_B,RB_NEW_B       ;Clear need to send out newest value
0155 1425                   BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
0156 295D                   GOTO    T4_S
0157 1F25             _0058 BTFSS   T_B,S_5_S_B        ;Serial In 5 secs of inactivity
0158 295E                   GOTO    Task_5
0159 30FF                   MOVLW   H'FF'              ;Tell of inactivity of Serial In
015A 00B1                   MOVWF   xmt_byte
015B 1325                   BCF     T_B,S_5_S_B        ;Clear need to send msg
015C 1425                   BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
015D                  T4_S                             ;Start Task #1
015D 2005                     CALL    Do_OState

015E                  Task_5   ;Task #5 - Monitor Level Reset Input Line - Always Running !
015E 1805                     BTFSC   PORTA,Level_Reset
015F 2962                     GOTO    Task_6
0160 3008                     MOVLW   LED_OFF_MODE     ;Lowest Level Indicator output
0161 00A7                     MOVWF   LED_Mode

0162                  Task_6 ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
0162 0AAE                     INCF    T_20_mS_CO, F    ;Inc Counter - Time Unit = 512 uS
0163 3027                     MOVLW   H'27'            ;Used to debounce the input
0164 022E                     SUBWF   T_20_mS_CO,W
0165 1D03                     BTFSS   STATUS,Z
0166 297C                     GOTO    _0065
0167 01AE                     CLRF    T_20_mS_CO       ;Reset T_20_mS_CO to start over again
0168 0806                     MOVF    PORTB,W          ;Last copy of RB same as Current ?
0169 0235                     SUBWF   Last_RB,W
016A 1903                     BTFSC   STATUS,Z
016B 296F                     GOTO    _0062
016C 0806                     MOVF    PORTB,W          ;Store Current RB - diff from Last
016D 00B5                     MOVWF   Last_RB
016E 2971                     GOTO    _0063
016F 0835             _0062   MOVF    Last_RB,W        ;New Old RB <- same value over 20 mS
0170 00B4                     MOVWF   Old_RB
0171 08B4             _0063   MOVF    Old_RB, F        ;See if RB is now 0
0172 1903                     BTFSC   STATUS,Z         ;RB == 0 ? then keep timer running
0173 2976                     GOTO    _0064
0174 01AB                     CLRF    T_5_M_LO         ;Reset 5 Min Timer
0175 01AC                     CLRF    T_5_M_HI         ;  still not zero yet
0176 21A8             _0064   CALL    D_H_E_L          ;Determine the Highest Error Level
0177 1EA5                     BTFSS   T_B,RB_NEW_B     ;Check for Lost Data Error
0178 11A6                     BCF     FLAGS,RB_Overflow
0179 1AA5                     BTFSC   T_B,RB_NEW_B
017A 15A6                     BSF     FLAGS,RB_Overflow
017B 16A5                     BSF     T_B,RB_NEW_B     ;Every 20 mS send Old_RB out

                             ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
017C                  _0065
017C 1683                     BSF     STATUS,RP0       ;Point to BANK 1
017D 30F1                     MOVLW   RA_TRIS          ;RA TRIS - refresh
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
017E 0085                     MOVWF   TRISA
017F 30FF                     MOVLW   H'FF'            ;RB TRIS - refresh
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0180 0086                     MOVWF   TRISB
0181 1283                     BCF     STATUS,RP0       ;Point to BANK 0
0182 0BAA                     DECFSZ  cc, F            ;Step-up time units * 256
0183 2933                     GOTO    Inc_Time

0184                  Task_7  ;Task 7 - Output Highest Level Indication on LED
```

```
0184 1DA7           00561         BTFSS   LED_Mode,LED_B   ;Is LED active ?
0185 2987           00562         GOTO    Task_8
0186 2039           00563         CALL    Do_LED           ;Handle LED timing
                    00564
0187                00565 Task_8 ;Task #8 - 5 Second Serial Input Lack of Activity Timer
0187 1D25           00566         BTFSS   T_B,T_5_S_B      ;5 Sec Timer Active ?
0188 2996           00567         GOTO    Task_9
0189 0AAD           00568         INCF    T_5_S_CO, F      ;Inc Counter - Time Unit = 131072 uS
018A 3026           00569         MOVLW   H'26'            ;Check T_5_S_CO if time
018B 022D           00570         SUBWF   T_5_S_CO,W
018C 1D03           00571         BTFSS   STATUS,Z
018D 2996           00572         GOTO    Task_9
018E 01AD           00573         CLRF    T_5_S_CO         ;Reset T_5_S_CO
018F 308F           00574         MOVLW   LED_ON_MODE      ;Highest Level Indicator output
0190 00A7           00575         MOVWF   LED_Mode
0191 1F25           00576         BTFSS   T_B,S_5_S_B      ;Check if Lost Data Error
0192 1226           00577         BCF     FLAGS,S_5_S_Overflow
0193 1B25           00578         BTFSC   T_B,S_5_S_B
0194 1626           00579         BSF     FLAGS,S_5_S_Overflow
0195 1725           00580         BSF     T_B,S_5_S_B      ;Send notice of 5 seconds of inaction
                    00581
0196                00582 Task_9 ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
0196 1DA5           00583         BTFSS   T_B,T_5_M_B      ;5 Min Timer Active ?
0197 29A7           00584         GOTO    Task_A
0198 0AAB           00585         INCF    T_5_M_LO, F      ;Inc LO Counter; Time Unit = 131072 uS
0199 1903           00586         BTFSC   STATUS,Z         ;See if carry needs to be passed on ?
019A 0AAC           00587         INCF    T_5_M_HI, F      ;Inc HI Counter; Time Unit = 131072 uS
019B 3008           00588         MOVLW   H'08'            ;#2288<  Check T_5_M_HI if time
019C 022C           00589         SUBWF   T_5_M_HI,W
019D 1D03           00590         BTFSS   STATUS,Z
019E 29A7           00591         GOTO    Task_A
019F 30F0           00592         MOVLW   H'F0'            ;#2288>  Check T_5_M_LO if time
01A0 022B           00593         SUBWF   T_5_M_LO,W
01A1 1D03           00594         BTFSS   STATUS,Z
01A2 29A7           00595         GOTO    Task_A
01A3 01AB           00596         CLRF    T_5_M_LO         ;Reset T_5_M_LO
01A4 01AC           00597         CLRF    T_5_M_HI         ;Reset T_5_M_HI
01A5 3008           00598         MOVLW   LED_OFF_MODE     ;Lowest Level Indicator output
01A6 00A7           00599         MOVWF   LED_Mode
01A7                00600 Task_A
01A7 2933           00601         GOTO    Inc_Time         ;Loop Forever
                    00602
                    00603 ;****            Determine the Highest Error Level & Start Task #7
01A8                00604 D_H_E_L
01A8 3007           00605         MOVLW   H'07'            ;Check top 7 bits
01A9 00A0           00606         MOVWF   temp
01AA 0834           00607         MOVF    Old_RB,W         ;Get copy of 7 debounced Sensor Input
01AB 00A1           00608         MOVWF   tmp
01AC 0DA1           00609 _0070   RLF     tmp, F           ;Put top bit into C bit
01AD 1803           00610         BTFSC   STATUS,C         ;Check if C bit is set
01AE 29B4           00611         GOTO    _0072
01AF 0BA0           00612         DECFSZ  temp, F          ;Continue to check lesser bits
01B0 29AC           00613         GOTO    _0070
01B1 0806           00614 _0071   MOVF    PORTB,W          ;Restore current value of RB
01B2 00A1           00615         MOVWF   tmp
01B3 0008           00616         RETURN
01B4 0827           00617 _0072   MOVF    LED_Mode,W       ;Get current Level Indicator
01B5 3907           00618         ANDLW   H'07'            ;Get only      "       "
01B6 00A1           00619         MOVWF   tmp              ;Store into a temporary register
01B7 0820           00620         MOVF    temp,W           ;Check if already at this Level
01B8 0221           00621         SUBWF   tmp,W
01B9 1803           00622         BTFSC   STATUS,C
01BA 29B1           00623         GOTO    _0071
01BB 3088           00624         MOVLW   H'88'            ;Start to build LED_Mode
01BC 0420           00625         IORWF   temp,W           ;Put new Level Indicator into reg
01BD 00A7           00626         MOVWF   LED_Mode         ;Store new LED Mode
```

```
01BE 29B1           00627        GOTO    _0071
                    00628
                    00629 ;******        Clear Registers 20-7Fh, A0-C0
01BF                00630 Clear_Regs
01BF 307F           00631        MOVLW   H'7F'           ;First regs to clear in Bank 0
01C0 0084           00632        MOVWF   FSR
01C1 0180           00633 Loop_C1 CLRF   INDF            ;Clear reg
01C2 0384           00634        DECF    FSR, F          ;point to next reg to clear
01C3 3020           00635        MOVLW   H'20'           ;Dec temp, jump if not done
01C4 0204           00636        SUBWF   FSR,W
01C5 1803           00637        BTFSC   STATUS,C
01C6 29C1           00638        GOTO    Loop_C1
                    00639
01C7 30C0           00640        MOVLW   H'C0'           ;First regs to clear in Bank 1
01C8 0084           00641        MOVWF   FSR
01C9 0180           00642 Loop_C2 CLRF   INDF            ;Clear reg
01CA 0384           00643        DECF    FSR, F          ;point to next reg to clear
01CB 30A0           00644        MOVLW   H'A0'           ;Dec temp, jump if not done
01CC 0204           00645        SUBWF   FSR,W
01CD 1803           00646        BTFSC   STATUS,C
01CE 29C9           00647        GOTO    Loop_C2
01CF 0184           00648        CLRF    FSR             ;Lastly clear FSR reg
01D0 0008           00649        RETURN
                    00650
                    00651 ;******        TMR0 IRS - Set Time_Bit for background tasks
01D1                00652 Do_Inc_Time
01D1 16A6           00653        BSF     FLAGS,Time_Bit  ;Tell background tasks of overflow
01D2 110B           00654        BCF     INTCON,T0IF     ;Clear for next overflow
01D3 0008           00655        RETURN
                    00656
                    00657 ;******        Handle Interrupts Here
01D4                00658 Interrupt
01D4 00A2           00659 PUSH:  MOVWF   Temp_W
01D5 0E03           00660        SWAPF   STATUS,W
01D6 00A3           00661        MOVWF   Temp_Stat
01D7 0804           00662        MOVF    FSR,W
01D8 00A4           00663        MOVWF   Temp_FSR
01D9 1283           00664        BCF     STATUS,RP0     ;Point to BANK 0 - Very IMPORTANT !!!!!
                    00665
01DA 188C           00666        BTFSC   PIR1,TMR2IF
01DB 2005           00667        CALL    Do_OState      ;Go Do Task #1 - all states
                    00668
01DC 18A5           00669        BTFSC   T_B,IState1_B  ;INTF will set even if INTE is cleared
01DD 29E0           00670        GOTO    I1
01DE 188B           00671        BTFSC   INTCON,INTF
01DF 2097           00672        CALL    Do_I1State     ;Go Do Task #2 - 0 state only
01E0                00673 I1:
01E0 180C           00674        BTFSC   PIR1,TMR1IF
01E1 2097           00675        CALL    Do_I1State     ;Go Do Task #2 - 1-B states
                    00676
01E2 190B           00677        BTFSC   INTCON,T0IF
01E3 21D1           00678        CALL    Do_Inc_Time    ;Go Inc Time_Bit every 512uS
                    00679
01E4 0824           00680 POP:   MOVF    Temp_FSR,W
01E5 0084           00681        MOVWF   FSR
01E6 0E23           00682        SWAPF   Temp_Stat,W
01E7 0083           00683        MOVWF   STATUS
01E8 0EA2           00684        SWAPF   Temp_W, F
01E9 0E22           00685        SWAPF   Temp_W,W
01EA 0009           00686        RETFIE                 ;Return from Interrupt
                    00687
                    00688        END
```



```
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 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX----- ----------------

All other memory blocks unused.

Program Memory Words Used:   488
Program Memory Words Free:  1560


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :    19 reported,     0 suppressed
```

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology India
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hongiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore
Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France
Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Müchen, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleone
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81-4-5471- 6166 Fax: 81-4-5471-6122

5/8/97

MICROCHIP