

# Creating a Multitasking Kernel for the COP8 Microcontroller

National Semiconductor  
Application Note 1099  
Richard F. Zarr  
March 1998



## INTRODUCTION

Everyone is familiar with multi-tasking operating systems such as Microsoft™ Windows® and UNIX®. These systems have the advantages of executing several tasks at once allowing a single CPU to do the job of several. These systems however, use very fast and sophisticated computers to run optimally. When it comes to the world of microcontrollers however, it seems very unlikely that one can take advantage of the features and benefits of a multitasking system. This is not necessarily true. With a little care and understanding of the limits of microcontrollers, a full-featured multitasking kernel (MTK) can be created. This MTK can allow many separate tasks to share the microcontroller's CPU and on-board peripherals and provide the added bonus of reduced time to market and lower development cost. In this application note, we will discuss the various types of multitasking systems, the best choice for a microcontroller, and how to implement an MTK in assembly language for the new COP8SGR microcontroller — a true system on a chip.

## MULTITASKING SYSTEMS

The word "Multitasking" or "Multithreading" can have several meanings, but they generally refer to performing more than one task at a time. In reality, multitasking operating systems generally are sharing the CPU among many tasks. This sharing is done so fast that it provides the illusion that the computer is actually doing more than one thing at a time. This sharing technique assumes a single processor. Some computers however, actually have many processors in parallel and the operating system assigns an individual processor to each task. In the case of the COP8, there is only the single CPU and its peripherals to share among many tasks. This requires a mechanism to "switch" which task is actually running on the CPU. There are several task-switching methods in use today, two of which are "Preemptive" and "Cooperative".

The "Preemptive" method shown in *Figure 1* allows a task, such as calculating the number pi, to execute for some pre-determined time. At the end of that time, the operating system or MTK suspends that task's ownership of the computer and enables the next task in a queue. The order in the queue can often be modified to allow higher priority tasks to get service faster. The problem with preemptive operating systems is that they require a great deal of resources for switching tasks. Since the task is stopped without warning — and usually in the middle of something complex, a complete, separate memory stack and variable space must be maintained for each task that is running. This also includes saving the address of the last instruction executed, and all the register states before the task was stopped. This is required to enable the operating system to restore the previous state of the task when it's time to start it again. This typically happens about every millisecond, but can happen faster or slower depending on real-time requirements. A characteristic of this type of task switching is that every task gets an equal amount of the system's CPU.

The "Cooperative" switching method shown in *Figure 2* handles tasks differently. The cooperative operating system uses a special understanding between the program task and itself to facilitate the sharing of resources. This "understanding" is as follows. Any task has full access to the CPU resource for as long as it requires, but must return the resource within a reasonable amount of time. Each thread must run from beginning to end, keeping track of its own state and then returning to the operating system. This implies that if the task does not return the CPU resource (i.e. gets stuck in an infinite loop), the system crashes or locks up. Sound familiar? The first versions of Microsoft Windows (Windows 3.x) used this scheme. If a program locked up, only an interrupt (such as the familiar ALT-CTRL-DEL sequence) could get the operating system's attention. Unix and Windows NT however, use a fully preemptive task switching method, which is much more robust.

Which method should be used for a microcontroller? The preemptive method is robust, but carries a heavy penalty in resources. The cooperative method uses much fewer resources, but can still lock up. A simple solution is to increase the robustness of the cooperative method by adding some of the qualities of the preemptive method. We'll call this method the "Supervised Cooperative" method and this will be the basis for COP8 based MTK. By adding some "watch-dog" features to the operating system, each task thread can be monitored for its usage of the CPU resource. If a task takes too long, it can be stopped and restarted — a nice feature.

## THE SUPERVISED COOPERATIVE MULTITASKING KERNEL

The kernel used in this application note will use the supervised cooperative method of task switching. Since the COP8SGR has a watch-dog timer (Timer T0) that is always running, an interrupt can be generated periodically to check the status of each task. Every 4096 cycle of the instruction cycle clock, Timer T0 can generate an interrupt. At 10 MHz, that is about every 4 milliseconds. By using this "Tick", the operating system can "watch" threads execute. If a task is stuck, or locked up in a loop due to a coding error or other failure, the operating system can recover. Here's how it works. Every time a task is switched, a counter is loaded with a user selectable value (1–255). This number is the maximum task time allowed by the operating system. A zero value disables this feature by turning off the timer T0 interrupt. To calculate the worst case time, use the following calculation:

$$T_{max} = n * T_{cycle} * 4096 \quad \text{where } n \text{ is the value passed to } \text{osSetMaxTaskTime}$$

If a task locks up in this example operating system, the OS transfers control to the first task loaded — the MAIN task. This task should have a message-processing loop to handle operating system messages from timers, UARTs, and the operating system itself. These messages are passed back to the loop in the accumulator on entry. If there are no mes-

UNIX® is a registered trademark of AT & T Bell Laboratories.  
Microsoft™ is a trademark of Microsoft Corporation.  
Windows® is a registered trademark of Microsoft Corporation.

sages for the message loop, the accumulator will be null (zero). If a task locks up, the main loop will receive a **mTaskFailure** message. The application can identify the task by looking at the 'B' register in the message loop. The main task can then restart it without affecting the other running tasks. To prevent locking up again, the OS stops the

failed task. This requires the application to restart it implicitly. Task 1 (the first task added) is always the main task. If the main task fails, the OS will automatically restart the microcontroller since this is a fatal failure. A sample Main Task message loop is shown below.

```

Main:
    IFEQ A, #mNone           ; Any messages?
    JP   MainContinue       ; No, continue with other main activities
    IFEQ A, #mTime          ; Was it a timer message
    JP   MainTimer          ; Yes, go service the timer tick.
    IFEQ A, #mUART          ; Was it a UART message?
    JP   MainUART           ; Yes, go service the UART
    IFEQ A, #mTaskFailure   ; Did a task lock up?
    JP   MainRestartTask    ; Yes, go fix it

MainContinue:

;   (Do other main stuff here)

    JP   MainExit           ; Done, return to OS

MainTimer:
    LD   A, B               ; 'B' holds which timer ticked...

;   (Do stuff with timer tick)

    JP   MainExit           ; Done, return to OS

MainUART:
    LD   A, B               ; 'B' holds UART action required...

;   (Do UART stuff)

    JP   MainExit           ; Done, return to OS

MainRestartTask:
    LD   A, B               ; 'B' holds which timer ticked...
    IFEQ A, #2              ; Task 2?
    JSR   StartTask2        ; Yes, Reload data for task 2 and restart it.
    IFEQ A, #3              ; Task 3?
    JSR   StartTask3        ; Yes, Reload data for task 3 and restart it.

;   (Continue checking for other failed tasks here)

MainExit:
    RET                     ; Return to OS

```

### SOME CONCEPTS

We've already discussed the idea of a task or thread, but let's review the idea again. A "Task" or "Thread" is a distinct piece of code (software) that handles a specific task. This task might be scanning a keyboard for input, updating a multiplexed display, communicating on a network with other devices, reading external or internal peripherals for data, and much more. The task structure for the supervised cooperative OS requires the code to execute completely in very few passes — that is, very little looping. Loops require time, and hog the CPU resource. These tasks look very much like sub-routines in that they are called by the OS, execute a function, and return back to the OS when completed. While the task is running however, no other task can have the CPU resource.

Another concept is the "Callback". Callbacks are a method of dynamically allocating resources such as timers, UARTs or other on-board microcontroller peripherals, and providing a direct handler for the resource — circumventing the main message loop. This concept works like this. A special piece

of code is written to handle a specific task. This code's address is passed to the operating system as a location to "call" when this function is needed. This feature provides the benefit of not requiring the main task message loop to process the interrupt and figure out what happened. The interrupt is simply routed directly to the service routine. Callbacks can also augment or override a default function, or provide a completely new function. Let's examine an example of a callback implementation.

A timer resource can be dynamically allocated using a callback. The application can specify an interval for a timer, and a routine to call when each time tick occurs. The routine is "Called Back" after each tick of the timer. The timer can also be used once to time a specific event without "looping" to burn time. The timer is created using **osSetTimer**, which also assigns a callback routine when the time period ends. The callback in turn stops the timer and frees the timer resource using **osKillTimer**. The callback routine can then continue its operation following the time delay. This also illus-

trates the idea of a "Resource Pool". The example MTK will use this concept to manage the COP8SGR UART and TIMER resources. Feel free to modify the MTK to extend this idea for handling other resources as well.

#### USING THE MULTITASKING KERNEL

The COP8SGR MTK architecture is shown in *Figure 3*. The COP8SGR has 4 pages of memory, each of which is 128 bytes long for a total of 512 bytes. They are selected by writing the page number into the segment register "S". The COP8 uses the last 16 bytes of PAGE 0 for registers, so they cannot be used for data storage. There is a single stack that is located in PAGE 0, however it is available in any page when using the **POP** and **PUSH** instructions. This is very important for this MTK in that the stack is used for passing information between the operating system and the application.

Initialize:

```
LD      A, #LOW(Main)      ; Add to task list
PUSH    A                  ; LSB First
LD      A, #HIGH(Main)    ;
PUSH    A                  ; MSB Second
JSR     osAddTask          ; Go add it...
X       A, hMainTask       ; Save the task handle.
```

This should be done for each of the tasks the user needs to run. Each time a task gets control, the entry point is always the same. In the above example, the entry point was called MAIN. Each task is responsible for keeping track of itself and using as little time as required to perform its function. Each

We will use page 1 for the operating system, but any page other than zero will work fine. This reserves 128 bytes for the operating system. The COP8SGR has 3 general purpose timers (T1–T3), and a watchdog timer (T0), as well as a full-featured UART. The operating system will control access to these resources to allow sharing among many threads.

The initialization routine sets up all the COP8SGR memory and internal registers after reset and then calls a user setup routine called "INITIALIZE". This is the first routine that gets control in the users code. Here the user should setup all the tasks that are initially active (more can be created later). This is done by pushing information onto the stack and calling the **osAddTask** routine. Here's what the code looks like to add a task called "MAIN".

task is in effect "looping" along as the operating system passes control to it. Each task also must be started. This enters the task into the switching queue to await execution. This is accomplished as follows.

```
LD      A, hMyTask         ; Tell OS which task to start
JSR     osStartTask        ; Start the task
```

At the end of the INITIALIZE routine, the code should start the MTK as follows.

```
JSR     osStart
```

This begins the multitasking kernel and begins the task switching. An error recovery routine should follow this jump subroutine command. If the OS has a fatal error, it will return and execute instructions following this call. This allows the user to select a method of restarting the processes in the case of a catastrophic code failure. This is equivalent to the BLUE SCREEN OF DEATH seen on many PCs.

Once the OS has begun, each task will be executed in turn. The entry point for each routine will be the values passed to the OS by the **osAddTask** call. Resource callbacks will be executed whenever that resource requires attention—such as a received character in the UART RX Buffer. The application can suspend any task by calling **osStopTask**. Again, feel free to modify the MTK code given in this application note to add features you may need.

#### ALLOCATING RESOURCES

To acquire a microcontroller resource such as a timer, the application code should use the operating system as opposed to using the resource directly. What's the advantage

of that? Other tasks cannot tell if a particular resource is in use, the operating system can. For the case of the on-board timers, of which the COP8SGR has 3, the MTK has a special operating system routine that requests a timer resource. Since all the general purpose timers are identical, it doesn't matter which timer a task uses. Therefore, the operating system decides which timer a task will get. This way, if another task needed a timer for awhile, and it received timer 1, the current task requesting a timer would get timer 2. Timer allocation (as well as UART and other peripheral allocation) is done using "handles". Handles are nothing more than a number that indicates which entry in a table belongs to a particular task. The handle allows both the task and OS to know which items belong to whom. The handle is returned by the operating system if the task received the timer (or peripheral). If not, the handle will be NULL or zero. Otherwise it will be a number greater than 0. This number must be kept in order to free the resource when a task has finished using it. For instance, to request a timer a task would do the following.

```
LD      A, #cPeriodLo      ; Setup time period
PUSH    A                  ; Low order first
LD      A, #cPeriodHi      ; followed by high order
PUSH    A
LD      A, #LOW(MyTimerCallBack) ; Setup call back
PUSH    A
LD      A, #HIGH(MyTimerCallBack)
PUSH    A
JSR     osSetTimer         ; Go get a timer resource
IFEQ    A, #0              ; Was the resource available?
```

```

JP      GetTimerFailure      ; No, bail and try later
X       A, hMyTimer          ; Yes, save the timer handle
(Continue)

```

To release the timer resource after the task is finished with it, the code should call the Kill Timer routine as follows.

```

LD      A, hMyTimer          ; Free my timer
JSR     osKillTimer

```

This routine will return zero in the accumulator if successful and non-zero if it fails. An invalid handle here can cause chaos. It may stop and release a timer being used by some other task that's critical to the application. Always store handles locally to the task. That is, keep a special variable that only that task uses for keeping the timer handle. No global variables for handles!

The timer allocation routine for this MTK is quite simple. It only allows for creating callbacks on a periodic basis. The general purpose timers in the COP8SGR however, are much more sophisticated. They have the ability to perform pulse width modulation (PWM) and capture the time between events as well. A more sophisticated callback handler would allow a mode select a particular timer and setup the timer for PWM operation, capture mode, or a periodic time. The routines in this application note can easily be expanded to include these features. Feel free to experiment and either modify these timer resource allocation routines or create your own.

#### WRITING TASKS

Obviously, the user of this MTK needs to add application specific code to make it useful. The code is basically made up of a collection of tasks that the application needs to perform. These tasks, as mentioned earlier, can handle many

functions such as updating a multiplexed display, or scanning a keyboard. To write these task follow these simple rules.

1. Write the task as a subroutine — a single entry and exit point. This improves maintainability as well.
2. Keep tasks specific — perform single operations per task.
3. Avoid loops — The longer a task holds a resource such as the CPU, the less efficient the OS becomes.
4. Use callbacks for timers, UARTs, and other OS supported on-board peripherals.

A sample task that illustrates these points well is multiplexing an LED display. This example will use 4 common anode 7 segment displays. The D PORT of the COP8SGR will be used to drive the multiplexed segments directly, and the lower 4 F PORT bits will be used to drive PNP transistors to supply anode current to the 7 segment displays (8 segments including the decimal point). See *Figure 4* for the connection diagram. A callback from a timer will be used to maintain a constant refresh on the display. The period only needs to be about 4 mS, which provides a total refresh time of about 16 milliseconds for 4 digits. This is fast enough that the human eye will not see any flickering of the display. The PNP transistors require the pin to go low to turn on, so the code in the following example deals with these pins as active low or inverted — notice the digit drive table entries are inverted as well.

The variables in RAM for this routine are as follows

```

bDigitCount .DSB 1 ; Provide a counter to keep track of which digit is in use
bDispBuffer .DSB 4 ; Provide a buffer for the characters

```

The timer will be assigned in the INITIALIZE routine.

```

LD      A, #cRefreshLo      ; Setup time period to refresh the display
PUSH    A
LD      A, #cRefreshHi      ; Low order first
PUSH    A
LD      A, #LOW(ServiceDisplay) ; Setup a call back to service the display
PUSH    A
LD      A, #HIGH(ServiceDisplay)
PUSH    A
JSR     osSetTimer           ; Go get a timer resource for this function
IFEQ    A, #0                ; Was the resource available?
JP      GetTimerFailure      ; No, bail and try later
X       A, hRefreshTimer     ; Yes, save the timer handle
(Continue...)                ; Continue to initialize other stuff...

```

The service routine for refreshing the 7 segment displays looks like this.

ServiceDisplay:

```

LD      A, PORTFD            ; Get current value of port F data
OR      A, #X'0F             ; Turn off all digits while updating (active low)
X       A, PORTFD            ; Write new value to port F data

LD      A, bDigitCount        ; Get current digit (0-3)
ADD     A, bDispBuffer
X       A, B                  ; Use 'B' to point to buffer
LD      A, [B]                ; Get the character to display
XOR     A, #X'FF              ; Invert for active low drive
X       A, PORTD              ; Write new value

LD      A, bDigitCount        ; Get digit count to turn on new digit

```

```

    ADD    A, DGTable      ; Use the count to select the digit table entry
    LAID   ; Load the accumulator with the digit data
    AND    A, PORTFD      ; Add what's already on the upper port pins
    X      A, PORTFD      ; Write to F PORT data

    LD     A, bDigitCount  ; Get digit count and update
    INC    A               ; Point to next digit
    IFGT   A, #cMaxDigits-1 ; Check if past last digit
    LD     A, #0           ; Yes, Reload to beginning
    X      A, bDigitCount  ; Save back for next pass
    RET    ; Return to OS

DGTable:
    .DB    B'11111110     ; Digit 1
    .DB    B'11111101     ; Digit 2
    .DB    B'11111011     ; Digit 3
    .DB    B'11110111     ; Digit 4

```

As this above example illustrates, the task uses a callback from the timer and updates a new digit each time the routine gets called by the operating system. It uses a single entry and exit point, is brief and handles a single function. These rules can be broken with care. For example, a task could have a single entry point and multiple exit points, however, it complicates the maintenance of the software.

Another example task would be to scan a keyboard for a button closure. If a button is pressed, the task should load a

variable with the scan code for that key. A scan code of 0xFF could be a no-key-pressed state. An example of this would be to use the L PORT of the COP8SGR for 16 keys connected in a matrix to the port. The lower 4 bits will be the drivers (active low) and the upper 4 will be the inputs (pulled-up). When a button is pushed it will short one of 4 lower port pins to one of the upper 4. See *Figure 5*. The task code would look like this.

```

ScanKeys:
    LD     rRowCount, #4    ; Scan 4 row (OK for only 4 passes in MTK)
    LD     bRowSel, #01     ; Use memory location for row select.
    LD     PORTLC, #X'0F    ; Lower 4 bits outputs, upper 4 bits inputs
    LD     PORTLD, #X'FF    ; Pull upper bits high, Clear driver bits
    LD     B, #01           ; Row to scan

ScanKeysLoop:
    LD     A, bRowSel       ; Get scan row
    XOR    A, #X'FF        ; Turn on (active low) scan row
    X      A, PORTLD        ; Send to port data
    LD     A, PORTLP        ; Check the pins on port L for change
    XOR    A, #X'FF        ; Reverse the polarity (Famous sci-fi quote)
    AND    A, #X'F0        ; Strip unused bits
    IFGT   A, #0           ; Hit?
    JP     ScanKeyHit       ; Yes, go see which key it was
    LD     A, bRowSel       ; Update the row pointer
    RC     ; Clear the carry flag - just in case
    RLC    A               ; Rotate left to next row
    X      A, bRowSel       ; Save it back
    DRSZ   rRowCount       ; Check the next row
    JP     ScanKeysLoop     ; Loop back
    LD     bScanCode, #X'FF ; Scan code for no key pressed.
    JP     ScanKeysExit     ; No keys pressed... return to OS

ScanKeyHit:
    IFBIT  4, A             ; Column 1?
    LD     bScanCode, #0    ; Yes, must
    IFBIT  5, A             ; Column 2?
    LD     bScanCode, #1    ; Yes, must
    IFBIT  6, A             ; Column 3?
    LD     bScanCode, #2    ; Yes, must
    IFBIT  7, A             ; Column 4?
    LD     bScanCode, #3    ; Yes, must

;
    LD     A, #4           ; Find row
    SUB    A, rRowCount    ; by subtraction
    RC     ; Reset the carry flag
    RLC    A               ; Multiply X2
    RLC    A               ; Multiply X2 (X4)
    ADD    A, bScanCode    ; Add to scan code
    X      A, bScanCode    ; Save it back

```

```

ScanKeysExit:
    LD    PORTLD, #X'00    ; Turn off all outputs to port
    RET                                ; Return to OS

```

Each time the MTK OS calls this routine, the keyboard is scanned. If a key is pressed, its scan code (0–15) is returned in the bScanCode variable. If not, the NO-KEY-PRESSED valued is loaded instead. Note here that the routine completes its function in a single call, much like a subroutine (which it is in reality — a subroutine of the application code). It also uses very little time to scan the entire keyboard. To reduce the time even further, each entry into the routine could

scan a single row instead of the entire keyboard. This is why the current row was kept in a local variable. The changes to implement this are quite simple. Instead of looping within the task, you simply loop to the exit routine. Also the row counter must be preloaded elsewhere, not in the beginning of the routine since it is updated each pass of the task. The changes would be as follows.

```

DRSZ    rRowCount        ; Check the next row
JP      ScanKeysExit     ; Wait for next entry to check next row...
LD      rRowCount, #4     ; Load new row count for next fresh start
LD      bRowSel, #01      ; Use memory location for row select.
LD      bScanCode, #X'FF  ; Scan code for no key pressed.
JP      ScanKeysExit     ; No keys pressed... return to OS

ScanKeyHit:
LD      rRowCount, #4     ; Reload the row counter
LD      bRowSel, #01      ; Use memory location for row select.

(CODE THE SAME FROM HERE)

```

## CONCLUSIONS

There are many benefits gained by using a multitasking kernel such as that shown here. Time to market can be reduced by creating software tasks that can be used over and over. Once the kernel is stable, new applications can be created in a very short time. Also, sophisticated applications that require many functions to work together and simultaneously can be easily implemented. Use the MTK code in this application note as a starting point to add your own handlers or to develop your own application. Multitasking on the COP8 has never been easier. Enjoy...

## OVERVIEW OF OPERATING SYSTEM ROUTINES

This section provides an overview of all the MTK operating system routines. This includes the routines that are accessible by the application code as well as those used internally by the operating system. All OS routines begin with the lower case letters "os". This will make it easy when looking at your code to determine if the routine belongs to the application or the operating system. These routines are grouped by function.

### Routine: **osPUSH**

Use: This routine pushes the 'A' register onto the OS data stack

Entry: Accumulator contains data byte to push

Returns: Accumulator contains original data (Like real PUSH)

Comments: Used for temporary application storage. This function provides another stack that can be used by the application for holding data.

### Routine: **osPOP**

Use: This routine pops data from the OS data stack into the 'A' register.

Entry: (none)

Returns: Accumulator contains popped data from OS data stack

Comments: Used for temporary application storage.

### Routine: **osSetMaxTaskTime**

Use: This routine sets the maximum time any task

can run (See text).

Entry: Accumulator holds time selection if greater than 0. If zero, function disabled.

Returns: Accumulator non-zero if successful, zero if fails

Comments: Use this function to increase the robustness of the application. Prevents task lock-up.

### Routine: **osAddTask**

Use: This routine will add a task entry point to the task list.

Entry: The stack (SP) holds the entry point address. Push LSB of the task entry point address first and then the MSB. Next call *osAddTask*.

Returns: Accumulator returns assigned task number if successful, zero if the task was not added.

Comments: Use this routine to add the tasks to the task list. The application must register all task threads by calling this routine.

Routine:	<b>osStartTask</b>	Returns:	(None)
Use:	This routine will start a task thread running.	Comments:	(Used by OS only). This routine is used as a safety net in the case a timer or other callback is unassigned and is executed. This will immediately return to the OS with no harm done.
Entry:	Accumulator contains the task number to start.		
Returns:	Accumulator is zero if successful, non-zero if failed.		
Comments:	For a task to execute, this routine must be called. Otherwise, the task is skipped and never executed.		
Routine:	<b>osStopTask</b>	Routine:	<b>osSetTimer</b>
Use:	This routine will stop a task thread from running.	Use:	This routine Start a timer with a call back routine.
Entry:	Accumulator contains the task number to stop.	Entry:	Stack has all data pushed on in this order: 1. Push LSB of Timer value 2. Push MSB of Timer value 3. Push LSB of call back routine address (0 if main task) 4. Push MSB of call back routine address (0 if main task)
Returns:	Accumulator is zero if successful, non-zero if failed.	Returns:	Accumulator holds handle number for timer (1-n) if successful Accumulator is zero (0) if resource unavailable
Comments:	Use this routine to suspend a task from running.	Comments:	If the callback is set to zero, Task 1 (first task added) will get the callback with the accumulator set to the system message "mTimer", and 'B' with the timer handle. This facilitates a single routine to handle all operating system messages.
Routine:	<b>osStart</b>	Routine:	<b>osKillTimer</b>
Use:	This routine will start the operating system	Use:	This routine stops a timer and frees its resource.
Entry:	(None required)	Entry:	Accumulator contains the non-zero timer handle.
Returns:	(Should not return unless fatal system error)	Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	This routine should be called after all tasks are added and started. Following the call to this routine, the application should contain an error handler to restart the system in the case of a fatal operating system error. This can occur if a call is made without the proper number of data values on the stack or other bad things...	Comments:	When a task is done using a timer resource, it must release this resource by calling this routine.
Routine:	<b>osMain</b>	Routine:	<b>osSignal</b>
Use:	This routine is the main loop of the kernel. All dispatching of tasks is done from here. Any error handling is also done here.	Use:	This routine will move a task to the top of the queue.
Entry:	(None required)	Entry:	Accumulator contains the task number to signal next.
Returns:	(Never unless really bad things have happened...)	Returns:	Accumulator is zero if successful, non-zero if failed.
Comments:	(Used by OS only)	Comments:	This is useful in real time conditions when a previous event requires faster service. For example, use this routine in a UART service callback to move the parser to the top of the list. It will then be executed next following the current task.
Routine:	<b>osGetTaskAddress</b>	Routine:	<b>osSetUART</b>
Use:	Get Task Address Subroutine	Use:	This routine will capture a UART to a task and setup the baud rate, framing, data bits, and buffers.
Entry:	Accumulator holds task-1	Entry:	'A' holds COM channel request (1 = UART 1, 2 = UART 2, etc.) Data is pushed on the stack as follows: 1. PUSH Configuration Byte (See <i>Tables 1, 2</i> ) 2. PUSH LSB of UART RX Call Back routine 3. PUSH MSB of UART RX Call Back routine
Returns:	B points to task entry address record	Returns:	If successful, accumulator will return the handle to the UART (1-n).
Comments:	(Used by OS only)		
Routine:	<b>osGetFirstTask</b>		
Use:	Gets the first running task number (first to start)		
Entry:	(None)		
Returns:	First running task number (1-n) in 'A' if successful, zero if it fails		
Comments:	(Used by OS only)		
Routine:	<b>osGetNextTask</b>		
Use:	Gets the next running task number (next to start)		
Entry:	(None)		
Returns:	next running task number (1-n) in 'A' if successful, zero if it fails		
Comments:	(Used by OS only)		
Routine:	<b>osUnassigned</b>		
Use:	Unassigned Callback Idle routine		
Entry:	(None)		

If failure, accumulator will be zero (0).

Comments: If the callback address is zero (0), Task 1 (first task added) will receive the callback with the accumulator set to the system message mUART and the 'B' with the UART handle. This facilitates a single routine to handle all operating system messages.

TABLE 1. UART Configuration Byte

Bit	Description
0-3	Baud Rate — See BAUD Rate Table
4	Data Bits (0=7, 1=8)
5	Stop Bits (0=1, 1=2)
6	Parity (0=none, 1=On)
7	Parity Type (0=Odd, 1=Even)

TABLE 2. Baud Rate Bits

Bits 3-0	Baud Rate (Bits per Second)
0000	110
0001	134.5
0010	150
0011	300
0100	600
0101	1200
0110	2400
0111	4800
1000	7200

Bits 3-0	Baud Rate (Bits per Second)
1001	9600
1010	19200
1011	Reserved — Defaults to 9600
1100	Reserved — Defaults to 9600
1101	Reserved — Defaults to 9600
1110	Reserved — Defaults to 9600
1111	Reserved — Defaults to 9600

Routine: **osUARTSend**

Use: This routine adds a byte to the transmit queue of the open UART resource.

Entry: Data byte is on the stack and 'A' holds the handle of the open UART resource.

Returns: If successful, returns zero in 'A'.  
If failure, return non-zero in 'A'.

Comments: Use this routine to send data. The transmit queue has two pointers — a read and write pointer. If the pointers are different, the UART transmit routine reads data from the transmit queue and updates the read pointer until they are the same once again. Once they are the same, transmissions stop.

Routine: **osKillUART**

Use: Frees a UART resource

Entry: 'A' holds the handle to the open UART resource

Returns: If successful, returns zero in 'A'.

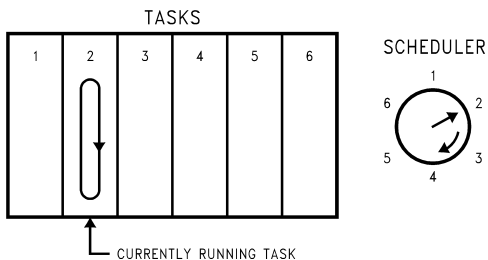
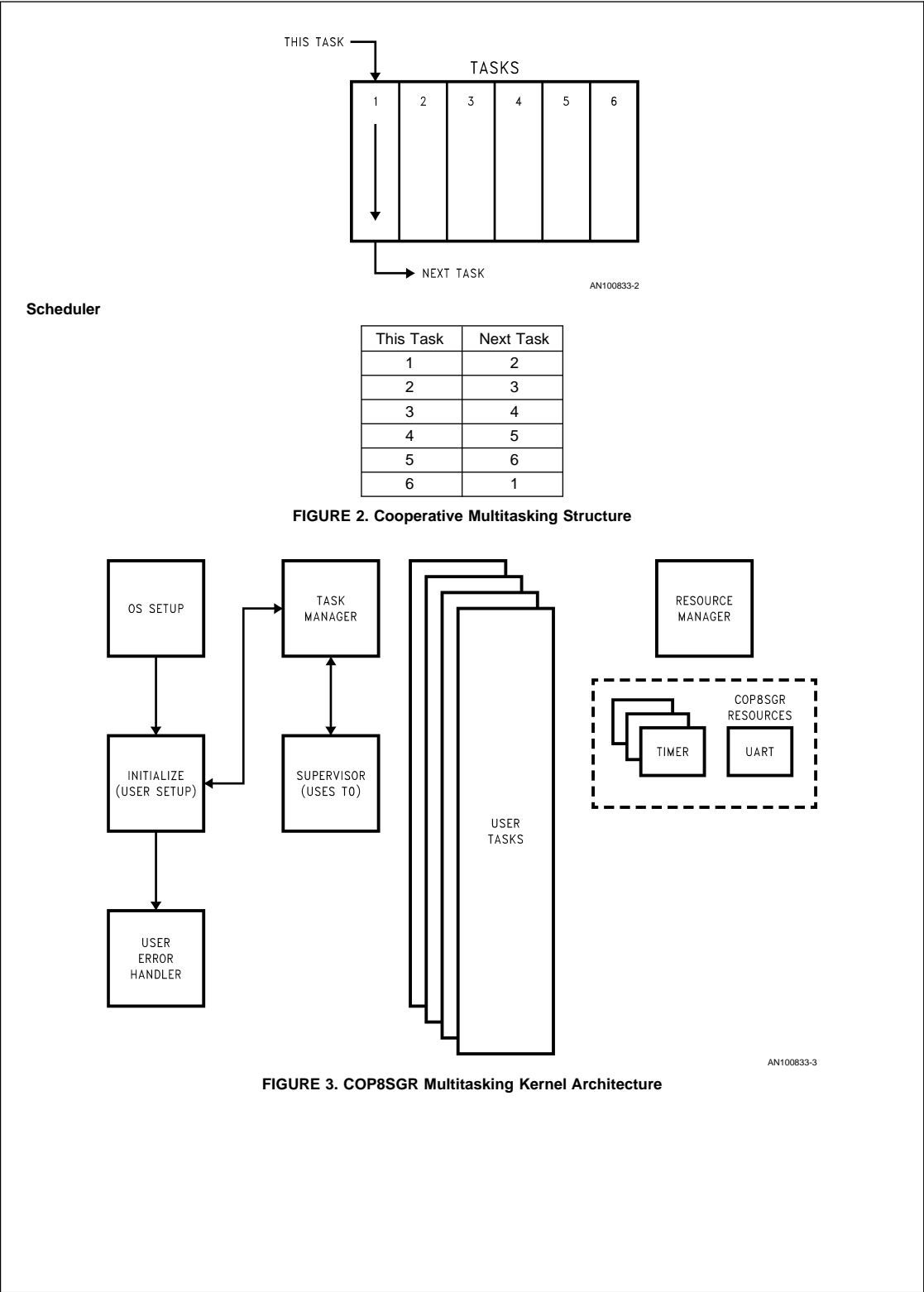
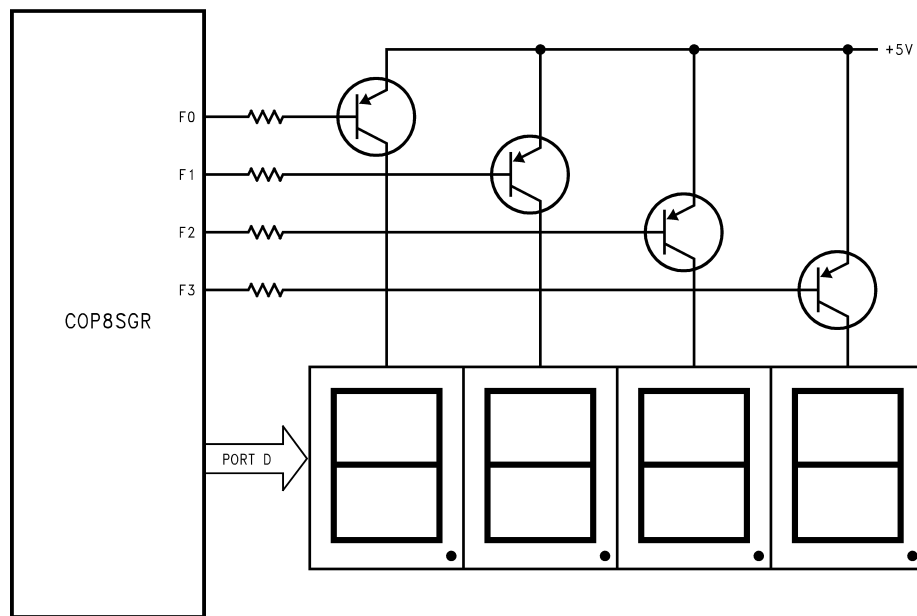


FIGURE 1. Preemptive Multitasking Structure

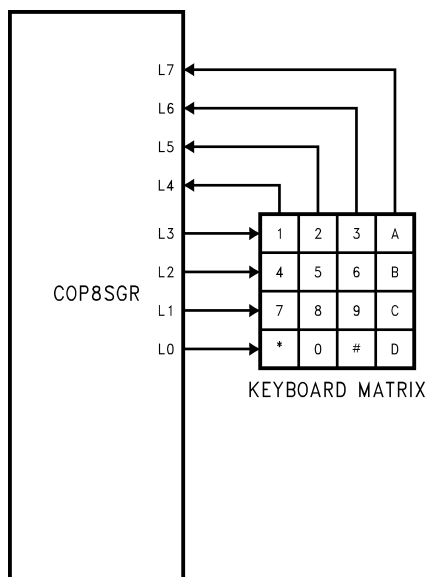






AN100833-4

FIGURE 4. Multiplexed 7 Segment Display Task Hookup



AN100833-5

FIGURE 5. Keyboard Scanning Task Hookup

```

1          .TITLE MTK , 'Multi Tasking Kernal'
2          ;*****
3          ; FILENAME: MTK88815.ASM
4          ; Multi-Tasking Kernal for simple multithreaded applications
5          ; Copyright (C) 1996, 1997 National Semiconductor Corporation
6          ;*****
7          ;
8          ;
9          ; Comment
10         ;
11         ;
12         ;
13         ;
14         ;
15         ;
16         ;
17         ;
18         ;
19         ;
20         ;*****
21         ; Notes:
22         ; A) Naming conventions:
23         ; The following naming methods are used through out this source code.
24         ; This makes classifying the variables easier when looking through
25         ; the listing. It also helps prevent association errors since the
26         ; assembler cannot tell you if you've associated an incorrect variable
27         ; with an operation. For example, if you wrote the following:
28         ;
29         ; LD A, xyz ; Get the data in variable xyz
30         ;
31         ; and you made the mistake that xyz was in effect a constant, the
32         ; assembler would load the location in memory that was pointed to by
33         ; the constant. This means you might get something like this:
34         ;
35         ; LD A, 3 ; Get the data in variable xyz
36         ;
37         ; instead of the intended variable location. This naming convention
38         ; can help prevent this. Here is the same thing using the names. I
39         ; have defined two things, one is the variable name and size and the
40         ; other is it's default value. They can be named almost the same thing
41         ; helping to keep track of what's what!
42         ;
43         ; LD A, cXYZ ; Load the default value for xyz
44         ; X A, bXYZ ; Save it into the variable
45         ;
46         ; Here's my definitions used in most of the source code I write.
47         ; Feel free to adopt this methodology or create your own.
48         ;
49         ; Variable prefix naming conventions:
50         ;
51         ; a : ARRAY (Group of bytes. Example: aData)
52         ; b : BYTE VALUE (Single byte value. Example: bData)
53         ; w : WORD (Variable defined as DSW. Example: wTest)
54         ; f : FLAG BIT (Bit within byte. Example: fWorking)
55         ; c : CONSTANT (ROM defined value Example: cValue)
56         ; p : POINTER (Byte long pointer into RAM. Example: pBuffer)
57         ; pc : POINTER CONSTANT (Pointer into EEPROM. Example: pcMaxMinV)
58         ; r : REGISTER (CPU register R0-RA Example: rCounter)
59         ; io : I/O PIN (I/O pin on COP8 controller Example: ioData)
60         ; i : INPUT PIN (Input pin on COP8 controller Example: iSwitch)
61         ; o : OUTPUT PIN (Output pin on COP8 controller Example: oLED1)
62         ;
63         ;*****
64         .INCLD COP888EG.INC ; Example 888 device for demo of MTK
65         ;*****
66         ;
67         ; CONSTANTS

```

```

68          ;
69          ; Example
70          ; cConstant = n ; Setup up constants like this
71          ;
72          ;*****
73
74      006F      cTopOfRAM      =      0x6F ; Top of RAM for COP87L88EG
75      007F      cTopOfSysRAM   =      0x7F ; Top of OS RAM for COP87L88EG
76      0001      cSystemSegment =      1   ; Current system RAM segment
77      0003      cRecSize       =      3   ; Size of task records in bytes
78      000A      cMaxTasks      =      10   ; Maximum number of tasks that can be loaded
79
80      0060      cBaudPrescale   =      0x60 ; UART Prescaler value 6.5 (01100) for 10MHz
81      0010      cCOM1TxBufLen  =      16   ; COM1 Transmitter FIFO length
82      0010      cCOM1RxBufLen  =      16   ; COM1 Receiver FIFO Length
83
84          ;*****
85          ;
86          ; OPERATING SYSTEM MESSAGES
87          ;
88          ;*****
89
90      0000      mNone           =      X'00 ; Idle message
91      0001      mTimer          =      X'01 ; Timer tick / service
92      0002      mUART           =      X'02 ; UART Service
93      000F      mTaskFailure     =      X'0F ; Task monitor service
94
95      0010      mUser           =      X'10 ; Beginning of user messages
96
97      0000      .SECT OSRAM, SEG ; Located in segment 1 of the COP87L88EG
98          ;*****
99          ;
100         ; VARIABLES
101         ;
102         ; Example:
103         ; wWord : .DSW 1 ; Word format for 16 bit location
104         ; bByte : .DSB 1 ; Byte format for 8 bit location
105         ;
106         ;*****
107      0000      aStore          :      .DSB 8 ; Local store for working with data
108      0008      pStack          :      .DSB 1 ; OS Data Stack Pointer
109
110      0009      bTask           :      .DSB 1 ; Currently running task number (1-n)
111      000A      bTaskNew        :      .DSB 1 ; Used by OS, Last task number added
112      000B      bTaskNext       :      .DSB 1 ; Next active task to switch in
113      000C      wAddress        :      .DSW 1 ; 16 bit address (MSB/LSB)
114      000E      bMaxTaskTime    :      .DSB 1 ; Max tick count for task monitor
115      000F      bTaskTime       :      .DSB 1 ; Remaining
116
117      0010      fStatus         :      .DSB 1 ; Status byte for kernal
118      0003      fUART           =      3     ; Bit 3: UART 1 in use
119      0002      fTimer3         =      2     ; Bit 2 : Timer 3 in use
120      0001      fTimer2         =      1     ; Bit 1 : Timer 2 in use
121      0000      fTimer1         =      0     ; Bit 0 : Timer 1 in use
122
123      0011      wPeriod         :      .DSW 1 ; Work areafor timer period
124      0013      wTimerCallBack  :      .DSW 1 ; Work area for call back
125      0015      wTimer1CallBack :      .DSW 1 ; Timer 1 callback address
126      0017      wTimer2CallBack :      .DSW 1 ; Timer 2 callback address
127      0019      wTimer3CallBack :      .DSW 1 ; Timer 3 callback address
128
129      001B      wIntCallBack     :      .DSW 1 ; Hardware interruptcallback
130
131      001D      bUARTSettings   :      .DSB 1 ; Current UART settings
132      0004      fDataBits       =      4     ; Data bits (0=7, 1=8)
133      0005      fStopBits      =      5     ; Stop Bits (0=1, 1=2)
134      0006      fParitySel      =      6     ; Parity Select (0=none, 1=On)

```

```

135      0007      fParityType      = 7              ; Parity Type (0=Odd, 1=Even)
136 001E      wUARTRxCallBack :      .DSW      1 ; UART RX Call back address
137
138 0020      bCOM1RxBuffer :      .DSB      cCOM1RxBufLen ; Setup buffer for RX
139 0030      pCOM1RxWrite :      .DSB      1 ; Write pointer to RX buffer
140 0031      pCOM1RxRead :      .DSB      1 ; Read pointer to RX buffer
141 0032      bCOM1TxBuffer :      .DSB      cCOM1TxBufLen ; Setup buffer for TX
142 0042      pCOM1TxWrite :      .DSB      1 ; Write pointer to TX buffer
143 0043      pCOM1TxRead :      .DSB      1 ; Read pointer to TX buffer
144 0044      bCOM1Status :      .DSB      1 ; Status byte for COM1
145      0007      fTxBusy      = 7 ; Tx FIFO Full
146      0006      fRxBusy      = 6 ; Rx FIFO Full
147
148      ;*****
149      ;
150      ;          T A S K   R E C O R D   S T R U C T U R E
151      ;
152      ; fTaskStatus : .DSB 1 ; Flags for each task
153      ; fActive      = 0 ; Bit 0 : State (0=Stopped, 1=Running)
154      ; bAddrLSB     : .DSB 1 ; Task Address LSB
155      ; bAddrMSB     : .DSB 1 ; Task Address MSB
156      ;
157      ;*****
158 0045      aTaskRecs :      .DSB      cRecSize * cMaxTasks ; Array of task records
159      0000      fActive      = 0 ; Active flag for task
160
161      ;***** I M P O R T   F U N C T I O N S *****
162      .EXTRN Initialize:ROM ; Inializes application code
163
164      ;***** E X P O R T   F U N C T I O N S *****
165      .PUBLIC osStart ; Starts the MTK running
166      .PUBLIC osAddTask ; Adds a task to the MTK task list
167      .PUBLIC osStartTask ; Starts a task running
168      .PUBLIC osStopTask ; Stops a task
169      .PUBLIC osSetMaxTaskTime ; Sets the maximum time a task can run before being killed
170      .PUBLIC osSignal ; Moves thread to top of queue
171      .PUBLIC osPUSH ; PUSHes a byte onto the OS stack from 'A'
172      .PUBLIC osPOP ; POPs a byte from the OS stack to 'A'
173      .PUBLIC osSetCommChannel ; Captures and initializes a communications resource
174      .PUBLIC osUARTGetChar ; Returns a byte from the UART RX queue
175      .PUBLIC osUARTSend ; Sends a character to the UART
176      .PUBLIC osSetTimer ; Captures and sets up a timer resource
177      .PUBLIC osKillTimer ; Stops and frees a timer resource
178      ;***** C O D E   B E G I N S *****
CODE SECTION
179      .FORM 'CODE SECTION'
180 0000      .SECT CODE, ROM, ABS=0x0000
181      ;
182 0000 DD6F 3 Start: LD SP, #cTopOfRAM ; Top of Stack for COP87L88EG
183      ;
184 0002      ClearRAM:
185 0002 DF00 3 LD S, #0 ; Point to RAM segment 0
186 0004 9F6F 2 LD B, #cTopOfRAM ; Setup pointer into Main RAM
187 0006      ClearRAMLoop1:
188 0006 9E00 2 LD [B], #000 ; Clear RAM location
189 0008 CE 3 DRSZ B ; Update counter, skip jump if done
190 0009 FC 3 JP ClearRAMLoop1 ; Continue until done
191 000A BC0000 3 LD 000 , #00 ; Clear last location
192      ;
193 000D DF01 3 LD S, #1 ; Point to RAM segment 1
194 000F 9F7F 2 LD B, #cTopOfSysRAM ; Setup pointer into OS RAM
195 0011      ClearRAMLoop2:
196 0011 9E00 2 LD [B], #000 ; Clear RAM location
197 0013 CE 3 DRSZ B ; Update counter, skip jump if done
198 0014 FC 3 JP ClearRAMLoop2 ; Continue until done
199 0015 BC0000 3 LD 000 , #00 ; Clear last location
200

```

```

201      ;
202 0018 BC087F 3 R      LD      pStack, #cTopOfSysRAM      ; Setup OS Data Stack
203      ;
204 001B 9F15 2 R      LD      B, #wTimer1CallBack      ; Initialize callbacks to nil
205 001D 9ACF 2      LD      [B+], #LOW(osUnassigned) ; to prevent the OS from crashing
206 001F 9A00 2      LD      [B+], #HIGH(osUnassigned) ; if the timer is active without
207 0021 9ACF 2      LD      [B+], #LOW(osUnassigned) ; the callback assigned.
208 0023 9A00 2      LD      [B+], #HIGH(osUnassigned) ;
209 0025 9ACF 2      LD      [B+], #LOW(osUnassigned) ;
210 0027 9A00 2      LD      [B+], #HIGH(osUnassigned) ;
211
212 0029 DF00 3      LD      S, #0      ; Point to main user RAM area
213
214 002B 9DB9 3      LD      A, RBUF      ; Receive and rough characters
215 002D BCBA00 3      LD      ENU, #B'00000000      ; Clear UART registers
216 0030 BCBB00 3      LD      ENUR, #B'00000000      ;
217 0033 BCBC00 3      LD      ENUI, #B'00000000      ;
218
219 0036 BCEE80 3      LD      CNTRL, #B'10000000      ; Setup timer 1A
220 0039 BCC685 3      LD      T2CNTRL, #B'10000101      ; Setup timer 2A,B
221 003C BCB685 3      LD      T3CNTRL, #B'10000101      ; Setup timer 3A,B
222 003F BCE801 3      LD      ICNTRL, #B'00000001      ; Setup timer 0, 1B
223 0042 BCEF11 3      LD      PSW, #B'00010001      ; Setup PSW, Timer 1A, GIE
224
225 0045 AC0000 > 4 X      JP      Initialize      ; Go setup tasks
226
227      ;*****
228      ;
229      ; Multitasking kernal routines
230      ;
231      ;*****
232      ;
233      ; Routine: osAddTask
234      ; Use: This routine will add a task entry point to the task list.
235      ;
236      ; Entry: Stack (SP) holds return address and data. Push LSB first
237      ; and then push MSB of task entry point callback routine.
238      ; Then call osAddTask
239      ; Returns: Accumulator returns assigned task number if successful,
240      ; zero if the task was not added.
241      ; Alters: 'A', 'B', and the SP
242      ;
243      ;*****
244 0048      osAddTask:
245 0048 9DFF 3      LD      A, S      ; Get current RAM segment
246 004A DF01 3      LD      S, #1      ; Switch to OS data segment
247 004C 9C00 3 R      X      A, aStore+0      ; Save for later to restore segment
248 004E 9D0A 3 R      LD      A, bTaskNew      ; Update New task counter
249 0050 8A 1      INC      A      ; Increment to next one
250 0051 930A 2      IFGT      A, #cMaxTasks      ; Check if larger than max task
251 0053 1C 3      JP      osAddTaskError      ; yes, Report error
252 0054 9C0A 3 R      X      A, bTaskNew      ; No, save it back
253      ;
254 0056 8C 3      POP      A      ; Get Return Address MSB
255 0057 9C0D 3 R      X      A, wAddress+1      ; Save in MSB of wAddress
256 0059 8C 3      POP      A      ; Get Return Address LSB
257 005A 9C0C 3 R      X      A, wAddress      ; Save in LSBoF wAddress
258      ;
259 005C 9D0A 3 R      LD      A, bTaskNew      ; Get New Task pointer
260 005E 8B 1      DEC      A      ; Correct for 0 reference
261 005F 30A4 5      JSR      osGetTaskAddress      ; Get address, pointed to by 'A'
262
263 0061 AA 2      LD      A, [B+]      ; Move past flags byte
264 0062 AA 2      LD      A, [B+]      ; Move past LSB of callback
265 0063 8C 3      POP      A      ; Get callback LSB
266 0064 A3 2      X      A, [B-]      ; Store MSB, increment B
267 0065 8C 3      POP      A      ; Get callback MSB

```

```

268 0066 A6      1      X      A, [B]      ; Store LSB, increment B
269
270 0067 9D0C    3 R      LD      A, wAddress      ; Restore return address
271 0069 67      3      PUSH     A      ; Start with LSB
272 006A 9D0D    3 R      LD      A, wAddress+1      ;
273 006C 67      3      PUSH     A      ; Now add MSB
274 006D 9D0A    3 R      LD      A, bTaskNew      ; Get the current task number
275 006F 01      3      JP       osAddTaskEnd      ; Return handle of task
276 0070
277 0070 64      1      CLR      A      ; Report error
278 0071
279 0071 67      3      PUSH     A      ; Save return value
280 0072 9D00    3 R      LD      A, aStore+0      ; Get old segment
281 0074 9CFF    3      X        A, S      ; Restore old segment
282 0076 8C      3      POP      A      ; Get return value
283 0077 8E      5      RET
284
285
286 ;*****
287 ;
288 ; Routine: osStartTask
289 ; Use: This routine will start running a task thread.
290 ;
291 ; Entry: Accumulator contains the task number to start.
292 ; Returns: Accumulator is zero if successful, non-zero if failed.
293 ;
294 ;*****
295 0078
296 0078 67      3      PUSH     A      ; Save 'A' for now
297 0079 9DFF    3      LD      A, S      ; Get current segment
298 007B DF01    3      LD      S, #1      ; Switch to OS RAM Page
299 007D 9C00    3 R      X        A, aStore+0      ; Save old RAM segment
300 007F 8C      3      POP      A      ; Get task number off main stack
301 0080 BD0A83 4 R      IFGT     A, bTaskNew      ; Is the task number valid?
302 0083 06      3      JP       osStartTaskError      ; No, don't start
303
304 0084 8B      1      DEC      A      ; Use corrected pointer (p-1)
305 0085 30A4    5      JSR      osGetTaskAddress      ; Get a pointer into the task buffer
306 0087 78      1      SBIT     fActive, [B]      ; Set Start Task flag in MSB
307 0088 64      1      CLR      A      ; Clear 'A' to signal start
308 0089 02      3      JP       osStartTaskEnd      ; End routine
309 008A
310 008A 98FF    2      LD      A, #0xFF      ; Signal error (non-zero)
311 008C
312 008C 67      3      PUSH     A      ; Save return value on main stack
313 008D 9D00    3 R      LD      A, aStore+0      ; Get old RAM segment from store
314 008F 9CFF    3      X        A, S      ; Restore old RAM segment
315 0091 8C      3      POP      A      ; Get return value
316 0092 8E      5      RET      ; Return to calling task
317
318 ;*****
319 ;
320 ; Routine: osStopTask
321 ; Use: This routine will stop a task thread from running.
322 ;
323 ; Entry: Accumulator contains the task number to stop.
324 ; Returns: Accumulator is zero if successful, non-zero if failed.
325 ;
326 ;*****
327 0093
328 0093 DF01    3      LD      S, #1      ; Switch to OS RAM Page
329 0095 BD0A83 4 R      IFGT     A, bTaskNew      ; Is the task number valid?
330 0098 06      3      JP       osStopTaskError      ; No, don't start
331
332 0099 8B      1      DEC      A      ; Use corrected pointer (p-1)
333 009A 30A4    5      JSR      osGetTaskAddress      ; Get a pointer into the task buffer
334 009C 78      1      SBIT     fActive, [B]      ; Start Task

```

```

335 009D 64      1      CLR    A          ; Clear 'A' to signal start
336 009E 02      3      JP      osStopTaskEnd ; End routine
337 009F          ;      osStopTaskError:
338 009F 98FF    2      LD      A, #0xFF      ; Signal error (non-zero)
339 00A1          ;      osStopTaskEnd:
340 00A1 DF00    3      LD      S, #0          ; Switch back to user RAM area
341 00A3 8E      5      RET                    ; Return to calling task
342
343 ;*****
344 ;
345 ;      Get Task Address Subroutine
346 ;      (Used by OS only)
347 ;
348 ; osGetTaskAddress : A is task-1, B returns address to first entry
349 ;
350 ;*****
351 00A4          ;      osGetTaskAddress:
352 00A4 9CFE    3      X        A, B          ; Use 'B' to do math
353 00A6 9DFE    3      LD      A, B          ; Get task pointer back
354 00A8 BDFE84  4      ADD     A, B          ; Multiply x2 for pointer
355 00AB BDFE84  4      ADD     A, B          ; Multiply x3 for pointer
356 00AE 9445    2 R    ADD     A, #aTaskRecs ; Add base address of record buffer
357 00B0 9CFE    3      X        A, B          ; Use 'B' now as pointer
358 00B2 8E      5      RET                    ;
359
360 ;*****
361 ;
362 ;      Get First and Next running Task Subroutine
363 ;      (Used by OS only)
364 ;
365 ; osGetFirstTask : Returns first running task (1-n) in 'A' if successfull
366 ; osGetNextTask  : Returns next running task (1-n) in 'A' if successfull
367 ;
368 ;      If either call fails, 'A' is zero
369 ;
370 ;*****
371 00B3          ;      osGetFirstTask:
372 00B3 9800    2      LD      A, #0          ; First task record
373 00B5 67      3      PUSH    A          ; Save it on the stack
374 00B6 03      3      JP      osGetNextTaskLoop ; go see what's enabled
375 00B7          ;      osGetNextTask:
376 00B7 9D09    3 R    LD      A, bTask      ; Current Task+1 (Already incremented)
377 00B9 67      3      PUSH    A          ; Save it for later
378 00BA          ;      osGetNextTaskLoop:
379 00BA 30A4    5      JSR     osGetTaskAddress ; Go get the address of the task
380 00BC 70      1      IFBIT   fActive, [B] ; Active?
381 00BD 0C      3      JP      osGetNextTaskHit ; Yes, we got the next active one
382
383 00BE 8C      3      POP     A          ; Get the old count
384 00BF 8A      1      INC     A          ; Update it
385 00C0 BD0A83  4 R    IFGT    A, bTaskNew ; Past last one?
386 00C3 64      1      CLR     A          ; Yes, Start at beginnning
387 00C4 BD0982  4 R    IFEQ    A, bTask      ; Hit them all?
388 00C7 05      3      JP      osGetNextTaskError ; Yes, Nothing is running. Bail...
389 00C8 67      3      PUSH    A          ; Save it for later
390 00C9 F0      3      JP      osGetNextTaskLoop ; Try again
391
392 00CA          ;      osGetNextTaskHit:
393 00CA 8C      3      POP     A          ; Get the task count off the stack
394 00CB 8A      1      INC     A          ; Adjust task number to 1-n
395 00CC 01      3      JP      osGetNextTaskDone ;
396 00CD          ;      osGetNextTaskError:
397 00CD 64      1      CLR     A          ; Return invalid handle
398 00CE          ;      osGetNextTaskDone:
399 00CE 8E      5      RET                    ; Done
400
401 ;*****

```



```

402          ;
403          ;   Unassigned Callback Idle routine
404          ;   (Used by OS only)
405          ;
406          ;   This routine is used as a safty net incase a timer or other callback
407          ;   is unassigned and is executed. This will immediately return to the OS.
408          ;
409          ;*****
410 00CF          osUnassigned:          ; Do nothing and return
411 00CF 8E          5          RET          ;
412
INTERRUPT ROUTINE
413          .FORM   'INTERRUPT ROUTINE'
414          ;*****
415          ;
416          ;   I N T E R R U P T   H A N D L E R
417          ;
418          ;   NOTES
419          ;   -----
420          ;   This routine is the entry point for all vectors . The VIS instruction
421          ;   will find the correct vector for interrupt. It is the responsibility
422          ;   of each service routine to clear the cause of the interrupt. A
423          ;   template is used for each of the existing interrupts. The default
424          ;   device for this source is the COP8SA family. The interrupts for this
425          ;   device are all supported. If this MTK is migrated to another feature
426          ;   family device, simply replace the 'ServiceUnused' routine with the
427          ;   missing routines.
428          ;
429          ;*****
430 00FF          .ORG   0x00FF          ; Force to 0x00FF for interrupt
431 00FF          Interrupt:
432 00FF 67          3          PUSH    A          ; Save 'A'
433 0100 9DFE          3          LD      A, B          ; Save 'B'
434 0102 67          3          PUSH    A          ;
435 0103 9DEF          3          LD      A, PSW          ; Save 'PSW'
436 0105 67          3          PUSH    A          ;
437 0106 9DFF          3          LD      A, S          ; Save segment
438 0108 67          3          PUSH    A          ;
439 0109 B4          5          VIS          ; Find interrupt vector
440          ;*****
441 010A          ServiceUnused:
442 010A B5          1          RPND          ; Incase it's set, clear SW pending
443 010B 2200 > 3          JP      Restore          ; Just return
444          ;*****
445 010D          ServiceWakeup:
446
447
448          ; *** Check which 'L' Port pin caused the interrupt here ***
449
450 010D BCCA00 3          LD      WKPND, #0          ; Clear all pending flags
451 0110 2200 > 3          JP      Restore          ; Return
452          ;*****
453 0112          ServiceTimerT0:
454 0112 BDE86D 4          RBIT     TOPND, ICNTRL          ; Clear pending flag
455 0115 DF01          3          LD      S, #1          ; Switch to OS segment
456 0117 9D0F          3 R          LD      A, bTaskTime          ; Decrement task watcher counter
457 0119 8B          1          DEC     A          ;
458 011A 9300          2          IFGT   A, #0          ; Did current running task die?
459 011C 1F          3          JP      ServiceTimerT0Exit          ; No, still running
460          ;
461 011D 9D09          3 R          LD      A, bTask          ; Did the main message loop die?
462 011F 9201          2          IFEQ   A, #1          ;
463 0121 2000          3          JMP     Start          ; Yes, restart the microcontroller
464          ; No, Stop the thread and report it...
465 0123 8B          1          DEC     A          ; Adjust task number to get address
466 0124 30A4          5          JSR     osGetTaskAddress          ; Retrieve the task address.
467 0126 68          1          RBIT     fActive, [B]          ; Disable the task

```

```

468                                     ;
469 0127 9800 2          LD      A, #LOW(Restore)      ; Push return address on stack
470 0129 67 3          PUSH    A                      ;
471 012A 9802 2          LD      A, #HIGH(Restore)     ;
472 012C 67 3          PUSH    A                      ;
473 012D 9D46 3 R      LD      A, aTaskRecs+1        ; Push address of main task onto stack
474 012F 67 3          PUSH    A                      ;
475 0130 9D47 3 R      LD      A, aTaskRecs+2        ;
476 0132 67 3          PUSH    A                      ;
477 0133 9D09 3 R      LD      A, bTask              ; Get active task that failed
478 0135 9CFE 3        X        A, B                ; Save in 'B'
479 0137 980F 2          LD      A, #mTaskFailure     ; Setup message to task
480 0139 DF00 3          LD      S, #0                ; Switch to User RAM for call back
481 013B 8E 5          RET                          ; Call message loop.
482
483 013C                                     ServiceTimerT0Exit:
484 013C 9C0F 3 R      X        A, bTaskTime          ; Save new count back to RAM
485 013E 2200 > 3      JP        Restore              ; Return
486                                     ;*****
487 0140                                     ServiceTimerT1A:
488 0140 BDEF6D 4      RBIT     TPNDB, PSW            ; Clear Pending flag
489 0143                                     ServiceTimerT1:
490 0143 DF01 3          LD      S, #1                ; Switch to OS RAM page
491 0145 9800 2          LD      A, #LOW(Restore)     ; Push return address on stack
492 0147 67 3          PUSH    A                      ;
493 0148 9802 2          LD      A, #HIGH(Restore)    ;
494 014A 67 3          PUSH    A                      ;
495 014B 9D15 3 R      LD      A, wTimer1CallBack    ; Push callback address on stack
496 014D 67 3          PUSH    A                      ;
497 014E 9D16 3 R      LD      A, wTimer1CallBack+   ;
498 0150 67 3          PUSH    A                      ;
499 0151 DF00 3          LD      S, #0                ; Switch to User RAM for call back
500 0153 8E 5          RET                          ; Go service it, return to RESTORE
501
502                                     ;*****
503 0143                                     ServiceTimerT1:
504 0143 DF01 3          LD      S, #1                ; Switch to OS RAM page
505 0154                                     ServiceTimerT1B:
506 0154 BDE869 4      RBIT     T1PNDB, ICNTRL        ; Clear Pending flag
507 0157 EB 3          JP        ServiceTimerT1       ; Go service call-back
508                                     ;*****
509 0158                                     ServiceTimerT2A:
510 0158 BDC66B 4      RBIT     T2PNDA, T2CNTRL       ; Clear Pending flag
511 015B                                     ServiceTimerT2:
512 015B DF01 3          LD      S, #1                ; Switch to OS RAM page
513 015D 9800 2          LD      A, #LOW(Restore)     ; Push return address on stack
514 015F 67 3          PUSH    A                      ;
515 0160 9802 2          LD      A, #HIGH(Restore)    ;
516 0162 67 3          PUSH    A                      ;
517 0163 9D17 3 R      LD      A, wTimer2CallBack    ; Push callback address on stack
518 0165 67 3          PUSH    A                      ;
519 0166 9D18 3 R      LD      A, wTimer2CallBack+1  ;
520 0168 67 3          PUSH    A                      ;
521 0169 DF00 3          LD      S, #0                ; Switch to User RAM for call back
522 016B 8E 5          RET                          ; Go service it, return to RESTORE
523                                     ;*****
524 016C                                     ServiceTimerT2B:
525 016C BDC669 4      RBIT     T2PNDB, T2CNTRL       ; Clear Pending flag
526 016F EB 3          JP        ServiceTimerT2       ; Go service call-back
527                                     ;*****
528 0170                                     ServiceTimerT3A:
529 0170 BDB66B 4      RBIT     T3PNDA, T3CNTRL       ; Clear Pending flag
530 0173                                     ServiceTimerT3:
531 0173 DF01 3          LD      S, #1                ; Switch to OS RAM page
532 0175 9800 2          LD      A, #LOW(Restore)     ; Push return address on stack
533 0177 67 3          PUSH    A                      ;
534 0178 9802 2          LD      A, #HIGH(Restore)    ;

```

```

533 017A 67      3      PUSH    A      ;
534 017B 9D19    3 R     LD      A, wTimer3CallBack ; Push callback address on stack
535 017D 67      3      PUSH    A      ;
536 017E 9D1A    3 R     LD      A, wTimer3CallBack+1 ;
537 0180 67      3      PUSH    A      ;
538 0181 DF00    3      LD      S, #0      ; Switch to User RAM for call back
539 0183 8E      5      RET      ; Go service it, return to RESTORE
540
;*****
541 0184      ServiceTimerT3B:
542 0184 BDB669    4      RBIT    T3PNDB, T3CNTRL ; Clear Pending flag
543 0187 EB      3      JP      ServiceTimerT3 ; Go service call-back
544
;*****
545 0188      ServiceUART_TX:
546 0188 DF01      3      LD      S, #1      ; Switch to OS RAM page
547 018A 9D43    3 R     LD      A, pCOM1TxRead ; Get the transmit buffer read pointer
548 018C 9CFE      3      X      A, B      ; Use 'B' as the pointer
549 018E AA      2      LD      A, [B+] ; Get the data
550 018F 9CB8      3      X      A, TBUF ; Move to UART Tx register
551 0191 BD446F    4 R     RBIT    fTxBusy, bCOM1Status ; Update status, free FIFO
552 0194 9DFE      3      LD      A, B      ; Get new pointer back
553 0196 9242      2 R     IFEQ    A, #bCOM1TxBuffer+cCOM1TxBufLen ; past buffer end?
554 0198 9832      2 R     LD      A, #bCOM1TxBuffer ; Reload start of buffer
555 019A BD4282    4 R     IFEQ    A, pCOM1TxWrite ; All data sent?
556 019D BDBC68    4      RBIT    ETI, ENUI ; Yes, Stop any new interrupts...
557 01A0 9C43      3 R     X      A, pCOM1TxRead ; Save new pointer value
558 01A2 2200 > 3      JP      Restore ; Return
559
;*****
560 01A4      ServiceUART_RX:
561 01A4 DF01      3      LD      S, #1      ; Switch to OS RAM page
562 01A6 9D30    3 R     LD      A, pCOM1RxWrite ; Get the write pointer
563 01A8 9CFE      3      X      A, B      ; Use 'B' as pointer
564 01AA 9DB9      3      LD      A, RBUF ; Get received character
565 01AC A2      2      X      A, [B+] ; Save into receive buffer
566 01AD 9DFE      3      LD      A, B      ; Update pointer
567 01AF 9230      2 R     IFEQ    A, #bCOM1RxBuffer+cCOM1RxBufLen; Past end of buffer
568 01B1 9820      2 R     LD      A, #bCOM1RxBuffer ; Reload start of buffer
569 01B3 9C30      3 R     X      A, pCOM1RxWrite ; Save pointer
570 01B5 9800      2      LD      A, #LOW(Restore) ; Push returnaddress on stack
571 01B7 67      3      PUSH    A      ;
572 01B8 9802      2      LD      A, #HIGH(Restore) ;
573 01BA 67      3      PUSH    A      ;
574 01BB 9D1E    3 R     LD      A, wUARTRxCallBack ; Push callback address on stack
575 01BD 67      3      PUSH    A      ;
576 01BE 9D1F    3R     LD      A, wUARTRxCallBack+1 ;
577 01C0 67      3      PUSH    A      ;
578 01C1 DF00      3      LD      S, #0      ; Switch to User RAM for call back
579 01C3 8E      5      RET      ; Launch the call back thread to service data
580
;*****
581
582 01C4      ServiceSoftware:
583 01C4 B5      1      RPND ; Clear SW interrupt (NMI) flag
584 01C5 8C      3      POP     A      ; Clear old return address
585 01C6 8C      3      POP     A      ;
586 01C7 DF00      3      LD      S, #0      ; Point to users RAM space
587 01C9 8E      5      RET      ; Return to Users osStart Call
588
;*****
589
590 01CA      ServiceMicrowire:
591 01CA BDE86B    4      RBIT    WPND, ICNTRL ; Clear microwire pending flag
592 01CD 2200 > 3      JP      Restore ; Return
593
;*****
594 01CF      ServiceExternal:
595 01CF BDEF6B    4      RBIT    IPND, PSW ; Clear the external interrupt pending
596 01D2 2200 > 3      JP      Restore ; Return
597
;*****
598
599 01E0      .ORG 0x01E0 ; Vector table begins at 0x01E0

```

```

600 01E0 010A      .ADDRW ServiceUnused ; Default vector (VIS with no interrupt)
601 01E2 010D      .ADDRW ServiceWakeup ; Wakeup vector
602 01E4 0184      .ADDRW ServiceTimerT3B ; Timer 3B vector
603 01E6 0170      .ADDRW ServiceTimerT3A ; Timer 3A vector
604 01E8 016C      .ADDRW ServiceTimerT2B ; Timer 2B vector
605 01EA 0158      .ADDRW ServiceTimerT2 ; Timer 2A vector
606 01EC 0188      .ADDRW ServiceUART_TX ; UART Transmit vector
607 01EE 01A4      .ADDRW ServiceUART_RX ; UART Receive vector
608 01F0 010A      .ADDRW ServiceUnused ;(Reserved for future use)
609 01F2 01CA      .ADDRW ServiceMicrowire ; Microwire vector
610 01F4 0154      .ADDRW ServiceTimerT1B ; Timer1B vector
611 01F6 0140      .ADDRW ServiceTimerT1A ; Timer 1A vector
612 01F8 0112      .ADDRW ServiceTimerT0 ; Timer 0 vector
613 01FA 01CF      .ADDRW ServiceExternal ; External pin vector
614 01FC010A      .ADDRW ServiceUnused ; (Reserved for future use)
61501FE 01C4      .ADDRW ServiceSoftware ; NMI - Software interrupt
616 ;*****
617 0200      Restore:
618 ;
619 0200 8C      3      POP      A ; Get old segment
620 0201 9CFF      3      X      A, S ; Restore segment register
621 0203 8C      3      POP      A ; Restore 'PSW' Carry flags
622 0204 BDEF6E      4      RBIT      C, PSW ; Assume no carry
623 0207 6040      2      IFBIT      C, A ; Was there a carry?
624 0209 BDEF7E      4      SBIT      C, PSW ; Yes, set it up again
625 020C BDEF6F      4      RBIT      HC, PSW ; Assume no half carry
626 020F 6080      2      IFBIT      HC, A ; Was there a half carry?
627 0211 BDEF7F      4      SBIT      HC, PSW ; Yes, set it up again
628 0214 8C      3      POP      A ; Restore 'B'
629 0215 9CFE      3      X      A, B ;
630 0217 8C      3      POP      A ; Restore 'A'
631 0218 8F      5      RETI ; Return from interrupt
632
633 ;*****
634
635 ;*****
636 ;
637 ; Routine: osPUSH
638 ; Use: This routine pushes the 'A' register onto the OS data stack
639 ; Entry: Accumulator contains data byte to push
640 ; Returns: Accumulator contains original data (Like real PUSH)
641 ; Comments: Used for user temporary storage
642 ;
643 ;*****
644 0219      osPUSH:
645 0219 67      3      PUSH      A ; Save 'A' for page switch
646 021A 9DFF      3      LD      A, S ; Get old segment
647 021C DF01      3      LD      S, #1 ; Switch to the OS Segment
648 021E 9C00      3 R      X      A, aStore+0 ; Save
649 0220 9DFE      3      LD      A, B ; Save 'B' for now
650 0222 9C01      3 R      X      A, aStore+1
651 0224 9D08      3 R      LD      A, pStack ; Get the stack pointer
652 0226 9CFE      3      X      A, B ; Stick it into 'B' for pointer
653 0228 8C      3      POP      A ; Get the data from the main stack
654 0229 67      3      PUSH      A ; Put it back on the main stack for later
655 022A A3      2      X      A, [B-] ; Push data, Adjust pointer
656 022B 9DFE      3      LD      A, B ; Save new stack pointer
657 022D 9C08      3 R      X      A, pStack ; in pointer register
658 022F 9D01      3 R      LD      A, aStore+1 ; Restore 'B'
659 0231 9CFE      3      X      A, B ;
660 0233 9D00      3 R      LD      A, aStore+0 ; Get old RAM segment
661 0235 9CFF      3      X      A, S ; Restore the old segment
662 0237 8C      3      POP      A ; return original 'A' value (like real PUSH)
663 0238 8E      5      RET ; Go do some more stuff...
664
665 ;*****
666 ;

```

```

667          ; Routine: osPOP
668          ; Use:      This routine pops data from the OS data stack into
669          ;           the 'A' register.
670          ; Entry:    (none)
671          ; Returns:  Accumulator contains popped data from OS data stack
672          ; Comments: Used for user temporary storage
673          ;
674          ;*****
675 0239      osPOP:
676 0239 9DFF 3      LD      A, S          ; Get current RAM segment
677 023B DF01 3      LD      S, #1        ; Switch to the OS Segment
678 023D 9C00 3 R    X      A, aStore+0    ; Save for later
679 023F 9DFE 3      LD      A, B          ; Save 'B'
680 0241 9C01 3 R    X      A, aStore+1    ; in store+1
681 0243 9D08 3 R    LD      A, pStack     ; Get the current OS Data Stack Pointer
682 0245 927F 2      IFEQ    A, #cTopOfSysRAM ; At the top?
683 0247 0E 3      JP      osPOPError     ; Yes, bad news... User screwed up.
684 0248 8A 1      INC      A            ; Adjust Stack pointer
685 0249 9C08 3 R    X      A, pStack     ; Save it back
686 024B 9D08 3 R    LD      A, pStack     ; Reloadpointer
687 024D 9CFE 3      X      A, B          ; Use 'B' as pointer into RAM
688 024F AE 1      LD      A, [B]         ; Get the data from the stack
689 0250 67 3      PUSH    A            ; Save it for return value
690 0251 9D01 3 R    LD      A, aStore+1    ; Restore 'B'
691 0253 9CFE 3      X      A, B          ; to original value
692 0255 02 3      JP      osPOPExit      ; Done, return
693 0256          osPOPError:
694 0256 64 1      CLR      A            ; Put '0' on stack for return value
695 0257 67 3      PUSH    A            ; Save return value
696 0258          osPOPExit:
697 0258 9D00 3 R    LD      A, aStore+0    ; Restore user RAM segment
698 025A 9CFF 3      X      A, S          ;
699 025C 8C 3      POP      A            ; Return value from OS Stack
700 025D 8E 5      RET                     ; Go do some more stuff...
701
702
703          ;*****
704          ;
705          ; Routine: osSetMaxTaskTime
706          ; Use:      This routine enables the task watcher and sets the time limit.
707          ;
708          ; Entry:    Accumulator holds the time constant, if zero function disabled.
709          ;
710          ; Returns:  Accumulator returns assigned task number if successful,
711          ;           zero if the task was not added.
712          ; Alters:   'A'
713          ;
714          ;*****
715 025E      osSetMaxTaskTime:
716 025E DF01 3      LD      S, #1        ; Switch to OS segment
717 0260 9200 2      IFEQ    A, #0        ; Turn function off?
718 0262 0A 3      JP      osSMTT_Off     ; Yes, turn it off, return.
719 0263 9C0E 3 R    X      A, bMaxTaskTime ; Save max task time
720 0265 9D0E 3 R    LD      A, bMaxTaskTime ; Get it back
721 0267 9C0F 3 R    X      A, bTaskTime   ; Transfer to register for counter
722 0269 BDE87C 4    SBIT    T0EN, ICNTRL   ; Turn on the timer interrupt
723 026C 09 3      JP      osSMTT_Exit    ; Done, return
724 026D          osSMTT_Off:
725 026D BDE86C 4    RBIT    T0EN, ICNTRL   ; Disable the T0 interrupt
726 0270 BC0F00 3 R    LD      bTaskTime, #0 ; Clear registers and counters
727 0273 BC0E00 3 R    LD      bMaxTaskTime, #0 ;
728 0276          osSMTT_Exit:
729 0276 DF00 3      LD      S, #0        ; Restore application segment
730 0278 8E 5      RET                     ; Return
731
732          ;*****
733          ;

```

```

734 ; Routine: osSetTimer
735 ; Use: This routine Start a timer with a call back routine. If the
736 ; call back is set to zero, Task 1 (Main task) will get the
737 ; call with the accumulator set to cmTimer, and 'B' with the
738 ; timer handle.
739 ;
740 ; Entry: Stack has all data pushed on in this order:
741 ; 1) Push LSB of Timer value
742 ; 2) Push MSB of Timer value
743 ; 3) Push LSB of call back routine address (0 if main task)
744 ; 4) Push MSB of call back routine address (0 if main task)
745 ; Returns: Accumulator holds handle number for timer (1-n) if successful
746 ; Accumulator is zero (0) if resource unavailable
747 ;
748 ;*****
749 0279 osSetTimer:
750 0279 DF01 3 LD S, #1 ; Switch Data RAM segment to OS
751 027B 8C 3 POP A ; Get MSB of calling routine
752 027C 9C0D 3 R X A, wAddress+1 ; Save for now
753 027E 8C 3 POP A ; Get LSB of calling routine
754 027F 9C0C 3 R X A, wAddress ; Save for now
755 0281 8C 3 POP A ; Get MSB of Call Back routine
756 0282 9C14 3 R X A, wTimerCallBack+1 ; Save for now
757 0284 8C 3 POP A ; Get LSB of Call Back routine
758 0285 9C13 3 R X A, wTimerCallBack ; Save for now
759 0287 8C 3 POP A ; Get MSB of time period
760 0288 9C12 3 R X A, wPeriod+1 ; Save for now
761 028A 8C 3 POP A ; Get LSB of time period
762 028B 9C11 3 R X A, wPeriod ; Save for now
763 028D 9D0C 3 R LD A, wAddress ; Restore the return address
764 028F 67 3 PUSH A ; Save LSB first
765 0290 9D0D 3 R LD A, wAddress+1 ;
766 0292 67 3 PUSH A ; Save MSB second
767 ;
768 0293 64 1 CLR A ; Clear 'A'
769 0294 9F10 2 R LD B, #fStatus ; Find a free timer
770 0296 osSetTimer1:
771 0296 70 1 IFBIT fTimer1, [B] ; Timer 1 free?
772 0297 06 3 JP osSetTimer2 ; No, try timer 2
773 0298 78 1 SBIT fTimer1, [B] ; Capture timer
774 0299 9F15 2 R LD B, #wTimer1CallBack ; yes, load and start timer 1
775 029B 9801 2 LD A, #1 ;
776 029D 10 3 JP osSetTimerStart ;
777 029E osSetTimer2:
778 029E 71 1 IFBIT fTimer2, [B] ; Timer 2 free?
779 029F 06 3 JP osSetTimer3 ; No, try timer 3
780 02A0 79 1 SBIT fTimer2, [B] ; Capture timer
781 02A1 9F17 2 R LD B, #wTimer2CallBack ; yes, load and start timer 2
782 02A3 9802 2 LD A, #2 ;
783 02A5 08 3 JP osSetTimerStart ;
784 02A6 osSetTimer3:
785 02A6 72 1 IFBIT fTimer3, [B] ; Timer 2 free?
786 02A7 22FB > 3 JP osSetTimerExit ; No timers available, sorry...
787 02A9 7A 1 SBIT fTimer3, [B] ; Capture timer
788 02AA 9F17 2 R LD B, #wTimer2CallBack ; yes, load and start timer 2
789 02AC 9802 2 LD A, #2 ;
790 ;
791 02AE osSetTimerStart:
792 02AE 67 3 PUSH A ; Save timer handle
793 02AF A0 1 RC ; Check if zero
794 02B0 9D13 3 R LD A, wTimerCallBack ; Get return address
795 02B2 BD1480 4 R ADC A, wTimerCallBack+1 ; See if all zeros
796 02B5 88 1 IFC ; If there was a carry, good address
797 02B6 0A 3 JP osSetTimerAddress ;
798 02B7 9900 2 IFNE A, #0 ; If not zero, also good address
799 02B9 07 3 JP osSetTimerAddress ;
800 ;

```

```

801 02BA 9D46    3 R      LD      A, aTaskRecs+1 ; Point to TASK 1 LSB (Default callback)
802 02BC A2      2        X      A, [B+]      ; Save LSB in call back record
803 02BD 9D47    3 R      LD      A, aTaskRecs+2 ; Now get MSB
804 02BF A6      1        X      A, [B]      ; Save MSB in call back record
805 02C0 06      3        JP      osSetTimerPeriod ; Now go fire it up!
806 02C1
807 02C1 9D13    3 R      LD      A, wTimerCallBack ; Get LSB of call back routine
808 02C3 A2      2        X      A, [B+]      ; Save in LSB of call back record
809 02C4 9D14    3 R      LD      A, wTimerCallBack+1 ; Get MSB of call back routine
810 02C6 A6      1        X      A, [B]      ; Save in MSB of call back record
811 02C7
812 02C7 8C      3        POP     A          ; Get timer handle back
813 02C8 67      3        PUSH    A          ; Save for later
814
815 02C9 9FEA     2        LD      B, #TMR1LO ; Point to timer 1 as default
816 02CB9202     2        IFEQ    A, #2      ; Timer 2?
817 02CD 9FC0     2        LD      B, #TMR2LO ; Yes, use timer 2
818 02CF 9203     2        IFEQ    A, #3      ; Timer 3?
819 02D1 9FB0     2        LD      B, #TMR3LO ; Yes, use timer 3
820
821 02D3 9D11    3 R      LD      A, wPeriod ; Get LSB of period
822 02D5 A2      2        X      A, [B+]      ; Save it
823 02D6 9D12    3 R      LD      A, wPeriod+1 ; Get MSB of period
824 02D8 A2      2        X      A, [B+]      ; Save it
825 02D9 9D11    3 R      LD      A, wPeriod ; Get LSB of period
826 02DB A2      2        X      A, [B+]      ; Save it
827 02DC 9D12    3 R      LD      A, wPeriod+1 ; Get MSB of period
828 02DE A2      2        X      A, [B+]      ; Save it
829
830 02DF 8C      3        POP     A          ; Check if timer 1 (FIX FOR MEM MAP)
831 02E0 67      3        PUSH    A          ; memory is scrambled for timer 1
832 02E1 9201     2        IFEQ    A, #1      ; Timer 1?
833 02E3 9FE6     2        LD      B, #T1RBLO ; Yes, fix location
834
835 02E5 9D11    3 R      LD      A, wPeriod ; Get LSB of period
836 02E7 A2      2        X      A, [B+]      ; Save it
837 02E8 9D12    3 R      LD      A, wPeriod+1 ; Get MSB of period
838 02EA A6      1        X      A, [B]      ; Save it
839
840 02EB 8C      3        POP     A          ; Get timer handle back
841 02EC 9201     2        IFEQ    A, #1      ; Start timer 1?
842 02EE BDEE7C   4        SBIT    T1CO, CNTRL ; Yes, start it up
843 02F1 9202     2        IFEQ    A, #2      ; Start timer 2?
844 02F3 BDC67C   4        SBIT    T2CO, T2CNTRL ; Yes, start it up
845 02F6 9203     2        IFEQ    A, #3      ; Start timer 3?
846 02F8 BDB67C   4        SBIT    T3CO, T3CNTRL ; Yes, start it up
847
848 02FB
849 02FB DF00     3        LD      S, #0      ; Switch data RAM segment back to user
850 02FD 8E      5        RET      ; Return
851
852 ;*****
853 ;
854 ; Routine: osKillTimer
855 ; Use: This routine stop a timer and free it.
856 ;
857 ; Entry: Accumulator contains the non-zero timer handle
858 ; Returns: Accumulator is zero if successful, non-zero if failed.
859 ;
860 ;*****
861 02FE
862 02FE DF01     3        LD      S, #1      ; Switch data segment to OS
863 0300 9201     2        IFEQ    A, #1      ; Kill Timer 1?
864 0302 09      3        JP      osKillTimer1 ; Yes, stop it.
865 0303 9202     2        IFEQ    A, #2      ; Kill Timer 2?
866 0305 0D      3        JP      osKillTimer2 ; Yes, stop it.
867 0306 9203     2        IFEQ    A, #3      ; Kill Timer 3?

```

```

868 0308 11      3      JP      osKillTimer3      ; Yes, stop it.
869 0309 98FF    2      LD      A, #0xFF          ; Failure, incorrect handle.
870 030B 15      3      JP      osKillTimerExit    ; Return
871 030C                                osKillTimer1:
872 030C BD1068  4 R    RBIT    fTimer1, fStatus    ; Free timer
873 030F BDEE6C  4      RBIT    T1C0, CNTRL         ; Stop the timer
874 0312 0D      3      JP      osKillTimerOK       ;
875 0313                                osKillTimer2:
876 0313 BD1069  4 R    RBIT    fTimer2, fStatus    ; Free timer
877 0316 BDC66C  4      RBIT    T2CO, T2CNTRL       ; Stop the timer
878 0319 06      3      JP      osKillTimerOK       ;
879 031A                                osKillTimer3:
880 031A BD106A  4 R    RBIT    fTimer3, fStatus    ; Free timer
881 031D BDB66C  4      RBIT    T3CO, T3CNTRL       ; Stop the timer
882 0320                                osKillTimerOK:
883 0320 64      1      CLR      A                  ; Report timer dead...
884 0321                                osKillTimerExit:
885 0321 DF00    3      LD      S, #0                ; Switch data segment back to user
886 0323 8E      5      RET                        ; Return
887
888 ;*****
889 ;
890 ; Routine: osSignal
891 ; Use:      This routine will move a task to the top of the queue.
892 ;
893 ; Entry:    Accumulator contains the task number to signal next.
894 ; Returns:  Accumulator is zero if successful, non-zero if failed.
895 ;
896 ;*****
897 0324                                osSignal:
898 0324 DF01    3      LD      S, #1                ; Switch to OS RAM Page
899 0326 BD0A83  4 R    IFGT     A, bTaskNew          ; Is the task number valid?
900 0329 0A      3      JP      osSignalError        ; No, don't start
901 ;
902 032A 8B      1      DEC      A                  ; Use corrected pointer (p-1)
903 032B 30A4    5      JSR      osGetTaskAddress    ; Get a pointer into the task buffer
904 032D 70      1      IFBIT    fActive, [B]        ; Check if task running
905 032E 01      3      JP      osSignalOK          ; Yes, go move to high priority
906 032F 04      3      JP      osSignalError        ; No, Go error out
907 ;
908 0330                                osSignalOK:
909 0330 9C0B    3 R    X        A, bTaskNext        ; Next task goes to top!
910 0332 64      1      CLR      A                  ; Clear 'A' to signal OK
911 0333 02      3      JP      osSignalEnd          ; End routine
912 0334                                osSignalError:
913 0334 98FF    2      LD      A, #0xFF            ; Signal error (non-zero)
914 0336                                osSignalEnd:
915 0336 DF00    3      LD      S, #0                ; Switch back to user RAM
916 0338 8E      5      RET                        ; Return to calling task
917
918 ;*****
919 ;
920 ; Routine: osSetCommChannel
921 ; Use:      This routine will capture a UART to a task
922 ;           and setup the baud rate, framing, data bits, and buffers.
923 ;
924 ; Entry:    'A' holds requested COM channel (1=UART 1, 2=UART 2, etc.)
925 ;           Data is pushed on the stack as follows:
926 ;           * PUSH baud rate (see tables below), data bits, and parity
927 ;           * PUSH LSB of UART RX Call Back routine
928 ;           * PUSH MSB of UART RX Call Back routine
929 ; Returns:  If successful, returns handle to UART (1-n) in 'A'
930 ;           If failure, return zero (0) in 'A'
931 ;
932 ; Table 1: 76543210 (Setup byte for UART - pushed on stack first)
933 ;           |||'---- Baud rate:
934 ;           |||'---- Data bits (0=7, 1=8)

```



```

935          ;          ||'----- Stop bits (0=1, 1=2)
936          ;          |'----- Parity (0=none, 1=On)
937          ;          '----- Parity Type(0=Odd, 1=Even)
938          ;
939          ;          Table 2: Baud Rate (bits 0-3)
940          ;
941          ;          0000 = 110   BPS
942          ;          0001 = 134.5 BPS
943          ;          0010 = 150   BPS
944          ;          0011 = 300   BPS
945          ;          0100 = 600   BPS
946          ;          0101 = 1200  BPS
947          ;          0110 = 2400  BPS
948          ;          0111 = 4800  BPS
949          ;          1000 = 7200  BPS
950          ;          1001 = 9600  BPS
951          ;          1010 = 19200 BPS
952          ;          1011 = (Reserved*)
953          ;          1100 = (Reserved*)
954          ;          1101 = (Reserved*)
955          ;          1110 = (Reserved*)
956          ;          1111 = (Reserved*)
957          ;
958          ;          Note * Defaults to 9600 BPS
959          ;
960          ;*****
961 0339          osSetCommChannel:
962 0339 9DFF      3          LD      A, S          ; Save old segment
963 033B DF01      3          LD      S, #1         ; Switch Data RAM segment to OS
964 033D 9C00      3 R        X      A, aStore+0 ; Save for later to restore (in OS Segment)
965
966 033F 8C        3          POP     A          ; Get MSB of calling routine
967 0340 9C0D      3 R        X      A, wAddress+1 ; Save for now
968 0342 8C        3          POP     A          ; Get LSB of calling routine
969 0343 9C0C      3 R        X      A, wAddress ; Save for now
970 0345 8C        3          POP     A          ; Get MSB of Call Back routine
971 0346 9C1F      3 R        X      A, wUARTRxCallBack+1 ; Save for now
972 0348 8C        3          POP     A          ; Get LSB of Call Back routine
973 0349 9C1E      3 R        X      A, wUARTRxCallBack ; Save for now
974 034B 8C        3          POP     A          ; Get MSB of time period
975 034C 9C1D      3 R        X      A, bUARTSettings ; Save for now
976          ;
977 034E 9D0C      3 R        LD      A, wAddress ; Restore the return address
978 0350 67        3          PUSH    A          ; Save LSB first
979 0351 9D0D      3 R        LD      A, wAddress+1 ;
980 0353 67        3          PUSH    A          ; Save MSB second
981          ;
982 0354 BD1073    4 R        IFBIT   fUART, fStatus ; is UART already in use?
983 0357 23D1      > 3        JP      osSetUARTFail ; Yes, bail out
984          ;
985 0359 BD107B    4 R        SBIT     fUART, fStatus ; Take control of UART
986 035C 9D1D      3 R        LD      A, bUARTSettings ; Setup UART
987 035E 950F      2          AND     A, #0x0F ; Clear unused bits for now
988 0360 A0        1          RC      ; Multiply by 2
989 0361 A8        1          RLC     A          ;
990 0362 9476      2          ADD     A, #LOW(osBaudTable) ; Get table offset
991 0364 A4        3          LAID    ; Get first byte
992 0365 9CBD      3          X      A, BAUD ; Put into baud rate register
993          ;
994 0367 9D1D      3 R        LD      A, bUARTSettings ; Setup UART
995 0369 950F      2          AND     A, #0x0F ; Clear unused bits for now
996 036B A0        1          RC      ; Multiply by 2
997 036C A8        1          RLC     A          ;
998 036D 9477      2          ADD     A, #LOW(osBaudTable)+1 ; Get table offset+1
999 036F A4        3          LAID    ; Get first byte
1000 0370 9760     2          OR      A, #cBaudPrescale ; Add prescaler value
1001 0372 9CBE     3          X      A, PSR ; Put into Prescaler register

```

```

1002 0374 2396 > 3 JP osSetUART2
1003 0376 osBaudTable:
1004 0376 6903 .DW 873 ; 110 BPS
1005 0378 CA02 .DW 714 ; 134.5 BPS
1006 037A 8002 .DW 640 ; 150 BPS
1007 037C 3F01 .DW 319 ; 300 BPS
1008 037E 9F00 .DW 159 ; 600 BPS
1009 0380 4F00 .DW 79 ; 1200 BPS
1010 0382 2700 .DW 39 ; 2400 BPS
1011 0384 1300 .DW 19 ; 4800 BPS
1012 0386 0E00 .DW 14 ; 7200 BPS
1013 0388 0900 .DW 9 ; 9600 BPS
1014 038A 0400 .DW 4 ; 19200 BPS
1015 038C 0900 .DW 9 ; 9600 BPS (Just in case)
1016 038E 0900 .DW 9 ; 9600 BPS (Ditto)
1017 0390 0900 .DW 9 ; 9600 BPS (Ditto)
1018 0392 0900 .DW 9 ; 9600 BPS (Ditto)
1019 0394 0900 .DW 9 ; 9600 BPS (Ditto)
1020 ;
1021 0396 osSetUART2:
1022 0396 BCBA08 3 LD ENUI, #B'00001000 ; No parity, 7 data bits
1023 0399BCBB00 3 LD ENUR, #B'00000000 ; Clear attention mode
1024 039CBCBC20 3 LD ENUI, #B'00100000 ; 1 stop bit, TDX pin enabled,
1025 ; ; Async mode, Int Clks
1026 039F BDD16B 4 RBIT 3, PORTLC ; RX pin as input
1027 03A2 BDD17A 4 SBIT 2, PORTLC ; TX Pin as output
1028 03A5 9D1D 3 R LD A, bUARTSettings ; Get the settings again
1029 03A7 6010 2 IFBIT fDataBits, A ; 7 or 8 data bits?
1030 03A9 BDBA6B 4 RBIT CHLO, ENUI ; 8, Set to 8 data bits
1031 03AC 6020 2 IFBIT fStopBits, A ; 1 or 2 stop bits
1032 03AE BDBC7F 4 SBIT STP2, ENUI ; 2, Set to 2 stop bits
1033 03B1 6040 2 IFBIT fParitySel, A ; Parity?
1034 03B3 BDBA7F 4 SBIT PEN, ENUI ; Yes, turn it on
1035 03B6 6080 2 IFBIT fParityType,A ; Odd/Even?
1036 03B8 BDBA7D 4 SBIT PSEL0, ENUI ; Even, set it up.
1037 ;
1038 03BB BC4232 3 R LD pCOM1TxWrite, #bCOM1TxBuffer ; Setup Tx Write pointer
1039 03BE BC4332 3 R LD pCOM1TxRead, #bCOM1TxBuffer ; Setup Tx Read pointer
1040 03C1 BC3020 3 R LD pCOM1RxWrite, #bCOM1RxBuffer ; Setup Rx write pointer
1041 03C4 BC3120 3 R LD pCOM1RxRead, #bCOM1RxBuffer ; Setup Rx Read pointer
1042 ;
1043 03C7 9D00 3 R LD A, aStore+0 ; Restore old RAM segment
1044 03C9 9CFF 3 X A, S ;
1045 03CB 9801 2 LD A, #1 ; Only one UART so return handle=1
1046 03CD BDBC79 4 SBIT ERI, ENUI ; Turn on the receiver interrupt
1047 03D0 05 3 JP osSetUARTExit ; Done
1048 03D1 osSetUARTFail:
1049 03D1 9D00 3 R LD A, aStore+0 ; Restore old RAM segment
1050 03D3 9CFF 3 X A, S ;
1051 03D5 64 1 CLR A ; Returnbad handle value
1052 03D6 osSetUARTExit:
1053 03D6 8E 5 RET ; Return
1054 ;
1055 ;*****
1056 ;
1057 ; Routine: osUARTGetChar
1058 ; Use: This routine gets one character from the receive queue
1059 ;
1060 ; Entry: 'B' holds comm channel handle
1061 ; Returns: 'A' hold received character
1062 ; 'B' is zero if fail, otherwise handle remains.
1063 ; 'X' is altered.
1064 ;
1065 ;*****
1066 03D7 osUARTGetChar:
1067 03D7 9DFF 3 LD A, S ; Save old segment
1068 03D9 DF01 3 LD S, #1 ; Switch Data RAM segment to OS

```

```

1069 03DB 9C00    3 R      X      A, aStore+0 ; Save for later to restore (in OS Segment)
1070
1071 03DD 9DFE    3        LD      A, B                ; Get comm channel handle
1072 03DF 9201    2        IFEQ    A, #1                ; Check handle to see if valid UART
1073 03E1 01      3        JP      osUARTRxCh1          ; UART on channel 1
1074 03E2 18      3        JP      osUARTRxFail         ; Not a valid UART
1075 03E3
1076 03E3 9D31    3 R      LD      A, pCOM1RxRead        ; Get the read pointer
1077 03E5 BD3082  4 R      IFEQ    A, pCOM1RxWrite       ; Any data to receive?
1078 03E8 12      3        JP      osUARTRxFail         ; No, bail...
1079 03E9 9CFC    3        X        A, X                ; Use 'X' as pointer into buffer
1080 03EB BA      3        LD      A, [X+]              ; Get the new character
1081 03EC 67      3        PUSH    A                    ; Save on stack for now
1082 03ED 9DFC    3        LD      A, X                ; Get new pointer value
1083 03EF 9230    2 R      IFEQ    A, #bCOM1RxBuffer+cCOM1RxBufLen ; Past end of buffer
1084 03F1 9820    2 R      LD      A, #bCOM1RxBuffer     ; Reload start of buffer
1085 03F3 9C31    3 R      X        A, pCOM1RxRead        ; Save pointer
1086 03F5 9D00    3 R      LD      A, aStore+0          ; Restore old RAM segment
1087 03F7 9CFF    3        X        A, S                ;
1088 03F9 8C      3        POP     A                    ; Get data back
1089 03FA 05      3        JP      osUARTRxExit         ; Done
1090 03FB
1091 03FB 9D00    3 R      LD      A, aStore+0          ; Restore old RAM segment
1092 03FD 9CFF    3        X        A, S                ;
1093 03FF 5F      1        LD      B, #0                ; Clear handle
1094 0400
1095 0400 8E      5        RET                          ; Done...
1096
1097 ;*****
1098 ;
1099 ; Routine: osUARTSend
1100 ; Use: This routine queues a byte of data for transmission by the UART.
1101 ;
1102 ; Entry: 'A' holds data byte, 'B' holds comm channel handle
1103 ; Returns: If successful, returns zero 'A'
1104 ; If failure, return non-zero in 'A'
1105 ;*****
1106 0401
1107 0401 67      3        PUSH    A                    ; Save data
1108 0402 9DFE    3        LD      A, B                ; Get comm channel handle
1109 0404 9201    2        IFEQ    A, #1                ; Check handle to see if valid UART
1110 0406 02      3        JP      osUARTSendCh1         ; UART on channel 1
1111 0407 2438    > 3      JP      osUARTSendError        ; Not a valid UART
1112 0409
1113 0409 9DFF    3        LD      A, S                ; Get the current segment register
1114 040B DF01    3        LD      S, #1                ; Switch to OS segment
1115 040D 9C00    3 R      X        A, aStore+0          ; Save for later on OS page
1116 040F BD4477  4 R      IFBIT    fTxBusy, bCOM1Status ; Is transmitter busy?
1117 0412 1D      3        JP      osUARTSendBusy        ; Yes, bail out
1118 0413 9D42    3 R      LD      A, pCOM1TxWrite       ; Get the write pointer to COM1
1119 0415 9CFE    3        X        A, B                ; Use 'B' to write the data
1120 0417 8C      3        POP     A                    ; Get the data byte off the stack
1121 0418 A2      2        X        A, [B+]              ; Write into buffer
1122 0419 9DFE    3        LD      A, B                ; Get pointer back
1123 041B 9242    2 R      IFEQ    A, #bCOM1TxBuffer+cCOM1TxBufLen ; Past end of buffer?
1124 041D 9832    2 R      LD      A, #bCOM1TxBuffer     ; Reload at start of buffer
1125 041F BD4382  4 R      IFEQ    A, pCOM1TxRead        ; Have we overrun the FIFO?
1126 0422 BD447F  4 R      SBIT     fTxBusy, bCOM1Status ; Yes, use later for test.
1127 0425 9C42    3 R      X        A, pCOM1TxWrite     ; Save the new COM1 write pointer
1128 0427 BDBC78  4 SBIT    ETI, ENUI ; Enable transmitter interrupt (start transmitter)
1129 042A 9D00    3 R      LD      A, aStore+0          ; Get old segment pointer back
1130 042C 9CFF    3        X        A, S                ; Return old segment
1131 042E 64      1        CLR     A                    ; Return with zero (success)
1132 042F 0B      3        JP      osUARTSendExit        ; Done, return
1133 0430
1134 0430 8C      3        POP     A                    ; Adjust Stack
1135 0431 9D00    3 R      LD      A, aStore+0          ; Get old segment pointer back

```

```

1136 0433 9CFF      3          X      A, S          ; Return old segment
1137 0435 98FE      2          LD      A, #X'FE      ; Return non-zero value (Busy)
1138 0437 03         3          JP      osUARTSendExit ; Done, return
1139 0438            osUARTSendError:
1140 0438 8C         3          POP      A          ; Adjust stack pointer
1141 0439 98FF      2          LD      A, #X'FF      ; Return non-zero value (Error)
1142 043B            osUARTSendExit:
1143 043B 8E         5          RET              ; Return
1144
1145            ;*****
1146            ;
1147            ; Routine: osStart
1148            ; Use:      This routine will start the MTK Operating System (OS)
1149            ;
1150            ; Entry:    (None required)
1151            ; Returns:  Accumulator is zero if successful, non-zero if failed.
1152            ;
1153            ;*****
1154 043C            osStart:
1155 043C DF01      3          LD      S, #1          ; Switch to OS RAM page
1156 043E 9D0A     3 R        LD      A, bTaskNew      ; Check if any task loaded.
1157 0440 9200      2          IFEQ     A, #0          ; Any tasks loaded?
1158 0442 2466     > 3        JP      osStartError    ; No, Return with error
1159            ;
1160 0444 30B3      5          JSR      osGetFirstTask ; Get the first enabled task
1161 0446 9200      2          IFEQ     A, #0          ; Anything to run?
1162 0448 1D        3          JP      osStartError    ; No, Return with error
1163 0449 9C09     3 R        X        A, bTask        ; Save in currently running task
1164 044B 30B7      5          JSR      osGetNextTask ; Get the next task number
1165 044D 9200      2          IFEQ     A, #0          ; Anything to run next?
1166 044F 16        3          JP      osStartError    ; No, Return with error
1167 0450 9C0B     3 R        X        A, bTaskNext   ; Save in next task variable
1168            ;
1169 0452 986B      2          LD      A, #LOW(osMain) ; Setup return Address
1170 0454 67        3          PUSH     A          ; First LSB
1171 0455 9804      2          LD      A, #HIGH(osMain) ;
1172 0457 67        3          PUSH     A          ; Next MSB
1173            ;
1174 0458 9D09     3 R        LD      A, bTask        ; Get first task that's enabled
1175 045A 8B        1          DEC      A          ; Yes, get the address
1176 045B 30A4      5          JSR      osGetTaskAddress ; Use to get the next task
1177 045D AA        2          LD      A, [B+]        ; Move past flags, Point to LSB
1178 045E AA        2          LD      A, [B+]        ; Get LSB, Point to MSB
1179 045F 67        3          PUSH     A          ; Put LSB on stack first (FILO)
1180 0460 AE        1          LD      A, [B]        ; Get MSB, Point at LSB
1181 0461 67        3          PUSH     A          ; Put on stack
1182
1183 0462 BDEF78     4          SBIT     GIE, PSW      ;
1184 0465 02        3          JP      osStartEnd     ; Go begin
1185 0466            osStartError:
1186 0466 98FF      2          LD      A, #0xFF        ; Make non-zero
1187 0468            osStartEnd:
1188 0468 DF00      3          LD      S, #0          ; Switch back to user RAM
1189 046A 8E        5          RET              ; Start first task (Cool!)
1190
1191            ;*****
1192            ;
1193            ; Routine: osMain
1194            ; Use:      This routine is the main loop of the kernal. All dispatching
1195            ;           is done from here. Any error handling is also done here.
1196            ;
1197            ; Entry:    (None required)
1198            ; Returns:  (Never)
1199            ;
1200            ;*****
1201 046B            osMain:
1202

```

```

1203                ; ***** DO ERROR CHECKING HERE *****
1204
1205 046B DF01      3      LD      S, #1          ; Switch to OS data segment
1206 046D 9D0B     3 R    LD      A, bTaskNext    ; Get the pointer to the next task
1207 046F 9200     2      IFEQ     A, #0          ; Something very bad has happened!
1208 0471 2495     > 3    JP      osMainExit      ; Let the user decide how to fix it.
1209 0473 986B     2      LD      A, #LOW(osMain) ; Put return address on stack
1210 0475 67       3      PUSH    A              ; First LSB
1211 0476 9804     2      LD      A, #HIGH(osMain) ;
1212 0478 67       3      PUSH    A              ; Next MSB
1213                ;
1214 0479 9D0B     3 R    LD      A, bTaskNext    ; Update current task
1215 047B 9C09     3 R    X        A, bTask      ;
1216 047D 30B7     5      JSR     osGetNextTask    ; Get the next one
1217 047F 9C0B     3 R    X        A, bTaskNext
1218
1219 0481 9D09     3 R    LD      A, bTask      ; Start next task thread
1220 0483 8B       1      DEC     A              ; Adjust task number for table look-up
1221 0484 30A4     5      JSR     osGetTaskAddress ; Look up task address
1222 0486 70       1      IFBIT   fActive, [B]    ; Check to make sure it's still active
1223 0487 08       3      JP      osMainSwitch    ; Yes, still active - go switch
1224 0488 30B7     5      JSR     osGetNextTask    ; No, find one that is OK to start
1225 048A 9C0B     3 R    X        A, bTaskNext    ; Save it
1226 048C 8C       3      POP     A              ; Adjust stack
1227 048D 8C       3      POP     A              ;
1228 048E 246B     > 3    JP      osMain          ; Go test it again
1229 0490                osMainSwitch:
1230 0490 AA       2      LD      A, [B+]          ; Load 'A' with Flags, point to LSB
1231 0491 AA       2      LD      A, [B+]          ; Load 'A' with LSB, point to MSB
1232 0492 67       3      PUSH    A              ; Save LSB
1233 0493 AE       1      LD      A, [B]          ; Load 'A' with MSB
1234 0494 67       3      PUSH    A              ; Save MSB
1235 0495                osMainExit:
1236 0495 DF00     3      LD      S, #0          ; Switch to user base RAM data segment
1237 0497 8E       5      RET
1238                ; .ENDSECT                ; Section ends here
1239 0498                .END      Start
.sect_CODE_1 . . . 0000 Abs Null ROM

ADRSLT . . . . . 00CC Abs Byte
-4
ANYCOP . . . . . 0000 Abs Null
-64
ATTN . . . . . 0002 Abs Null
-4
aStore . . . . . 0000 Rel Byte SEG
-107 247 280 299 313 648 650 658 660 678 680 690 697 964 1043 1049
1069 1086 1091 1115 1129 1135
aTaskRecs . . . . 0045 Rel Byte SEG
-158 356 473 475 801 803
BAUD . . . . . 00BD Abs Byte
-4 992
BUSY . . . . . 0002 Abs Null
-4
bCOM1RxBuffer . . 0020 Rel Byte SEG
-138 567 568 1040 1041 1083 1084
bCOM1Status . . . 0044 Rel Byte SEG
-144 551 1116 1126
bCOM1TxBuffer . . 0032 Rel Byte SEG
-141 553 554 1038 1039 1123 1124
bMaxTaskTime . . . 000E Rel Byte SEG
-114 719 720 727
bTask . . . . . 0009 Rel Byte SEG
-110 376 387 461 477 1163 1174 1215 1219
bTaskNew . . . . 000A Rel Byte SEG
-111 248 252 259 274 301 329 385 899 1156
bTaskNext . . . . 000B Rel Byte SEG

```

-112	909	1167	1206	1214	1217	1225
bTaskTime	. . . .	000F	Rel	Byte	SEG	
-115	456	484	721	726		
bUARTSettings	. . . .	001D	Rel	Byte	SEG	
-131	975	986	994	1028		
C	. . . . .	0006	Abs	Null		
-4	622	623	624			
CHL1	. . . . .	0004	Abs	Null		
-4						
CHLO	. . . . .	0003	Abs	Null		
-4	1030					
CKO	. . . . .	0007	Abs	Null		
-4						
CKX	. . . . .	0001	Abs	Null		
-4						
CMP10E	. . . . .	0003	Abs	Null		
-4						
CMP1EN	. . . . .	0001	Abs	Null		
-4						
CMP1INN	. . . . .	0001	Abs	Null		
-4						
CMP1INP	. . . . .	0002	Abs	Null		
-4						
CMP1OUT	. . . . .	0003	Abs	Null		
-4						
CMP1RD	. . . . .	0002	Abs	Null		
-4						
CMP20E	. . . . .	0006	Abs	Null		
-4						
CMP2EN	. . . . .	0004	Abs	Null		
-4						
CMP2INN	. . . . .	0004	Abs	Null		
-4						
CMP2INP	. . . . .	0005	Abs	Null		
-4						
CMP2OUT	. . . . .	0006	Abs	Null		
-4						
CMP2RD	. . . . .	0005	Abs	Null		
-4						
CMPSL	. . . . .	00B7	Abs	Byte		
-4						
CNTRL	. . . . .	00EE	Abs	Byte		
-4	219	842	873			
COP820	. . . . .	0002	Abs	Null		
-64						
COP820CJ	. . . . .	0005	Abs	Null		
-64						
COP840	. . . . .	0003	Abs	Null		
-64						
COP840CJ	. . . . .	0006	Abs	Null		
-64						
COP8620	. . . . .	0007	Abs	Null		
-64						
COP8640	. . . . .	0008	Abs	Null		
-64						
COP8720	. . . . .	0009	Abs	Null		
-64						
COP8780	. . . . .	000A	Abs	Null		
-64						
COP880	. . . . .	0004	Abs	Null		
-64						
COP888BC	. . . . .	001B	Abs	Null		
-64						
COP888CF	. . . . .	0014	Abs	Null		
-64						
COP888CG	. . . . .	0015	Abs	Null		
-64						

```

COP888CL . . . . . 0016 Abs Null
-64
COP888CS . . . . . 0017 Abs Null
-64
COP888EB . . . . . 001C Abs Null
-64
COP888EG . . . . . 0018 Abs Null
-64      4
COP888EK . . . . . 0019 Abs Null
-64
COP888EW . . . . . 001D Abs Null
-64
COP888FH . . . . . 001E Abs Null
-64
COP888GD . . . . . 001F Abs Null
-64
COP888GG . . . . . 0020 Abs Null
-64
COP888GW . . . . . 0021 Abs Null
-64
COP888HG . . . . . 0022 Abs Null
-64
COP888KG . . . . . 0023 Abs Null
-64
COP8ACC . . . . . 001A Abs Null
-64
COP8SAA . . . . . 0024 Abs Null
-64
COP8SAB . . . . . 0025 Abs Null
-64
COP8SAC . . . . . 0026 Abs Null
-64
COP912C . . . . . 0001 Abs Null
-64
COP943 . . . . . 000B Abs Null
-64
COPCHIP . . . . . 0018 Abs Null
-4
ClearRAM . . . . . 0002 Abs Null ROM
-184
ClearRAMLoop1 . . 0006 Abs Null ROM
-187      190
ClearRAMLoop2 . . 0011 Abs Null ROM
-195      198
cBaudPrescale . . 0060 Abs Null
-80      1000
cCOM1RxBufLen . . 0010 Abs Null
-82      138      567      1083
cCOM1TxBufLen . . 0010 Abs Null
-81      141      553      1123
cMaxTasks . . . . . 000A Abs Null
-78      158      250
cRecSize . . . . . 0003 Abs Null
-77      158
cSystemSegment . . 0001 Abs Null
-76
cTopOfRAM . . . . . 006F Abs Null
-74      182      186
cTopOfSysRAM . . . 007F Abs Null
-75      194      202      682
DOE . . . . . 0007 Abs Null
-4
ENAD . . . . . 00CB Abs Byte
-4
ENI . . . . . 0001 Abs Null
-4
ENTI . . . . . 0004 Abs Null

```

-4									
ENU	. . . . .	00BA Abs Byte							
-4	215	1022 1030	1034	1036					
ENUI	. . . . .	00BC Abs Byte							
-4	217	556 1024	1032	1046	1128				
ENUR	. . . . .	00BB Abs Byte							
-4	216	1023							
ERI	. . . . .	0001 Abs Null							
-4	1046								
ERR	. . . . .	0002 Abs Null							
-4									
ETDX	. . . . .	0005 Abs Null							
-4									
ETI	. . . . .	0000 Abs Null							
-4	556	1128							
FE	. . . . .	0006 Abs Null							
-4									
fActive	. . . . .	0000 Abs Null							
-159	306	334 380	467	904	1222				
fDataBits	. . . . .	0004 Abs Null							
-132	1029								
fParitySel	. . . . .	0006 Abs Null							
-134	1033								
fParityType	. . . . .	0007 Abs Null							
-135	1035								
fRxBusy	. . . . .	0006 Abs Null							
-146									
fStatus	. . . . .	0010 Rel Byte SEG							
-117	769	872 876	880	982	985				
fStopBits	. . . . .	0005 Abs Null							
-133	1031								
fTimer1	. . . . .	0000 Abs Null							
-121	771	773 872							
fTimer2	. . . . .	0001 Abs Null							
-120	778	780 876							
fTimer3	. . . . .	0002 Abs Null							
-119	785	787 880							
fTxBusy	. . . . .	0007 Abs Null							
-145	551	1116 1126							
fUART	. . . . .	0003 Abs Null							
-118	982	985							
GIE	. . . . .	0000 Abs Null							
-4	1183								
HC	. . . . .	0007 Abs Null							
-4	625	626 627							
ICNTRL	. . . . .	00E8 Abs Byte							
-4	222	454 504	591	722	725				
IEDG	. . . . .	0002 Abs Null							
-4									
INT	. . . . .	0000 Abs Null							
-4									
INTR	. . . . .	0000 Abs Null							
-4									
IPND	. . . . .	0003 Abs Null							
-4	595								
Initialize	. . . . .	**** Rel Null ROM Ext							
162	225								
Interrupt	. . . . .	00FF Abs Null ROM							
-431									
LPEN	. . . . .	0006 Abs Null							
-4									
MSEL	. . . . .	0003 Abs Null							
-4									
mNone	. . . . .	0000 Abs Null							
-90									
mTaskFailure	. . . . .	000F Abs Null							
-93	479								



```

mTimer . . . . . 0001 Abs Null
-91
mUART . . . . . 0002 Abs Null
-92
mUser . . . . . 0010 Abs Null
-95
osAddTask . . . . 0048 Abs Null ROM Pub
166 -244
osAddTaskEnd . . . 0071 Abs Null ROM
275 -278
osAddTaskError . . 0070 Abs Null ROM
251 -276
osBaudTable . . . 0376 Abs Null ROM
990 998 -1003
osGetFirstTask . . 00B3 Abs Null ROM
-371 1160
osGetNextTask . . 00B7 Abs Null ROM
-375 1164 1216 1224
osGetNextTaskDone 00CE Abs Null ROM
395 -398
osGetNextTaskError 00CD Abs Null ROM
388 -396
osGetNextTaskHit . 00CA Abs Null ROM
381 -392
osGetNextTaskLoop 00BA Abs Null ROM
374 -378 390
osGetTaskAddress . 00A4 Abs Null ROM
261 305 333 -351 379 466 903 1176 1221
osKillTimer . . . 02FE Abs Null ROM Pub
177 -861
osKillTimer1 . . . 030C Abs Null ROM
864 -871
osKillTimer2 . . . 0313 Abs Null ROM
866 -875
osKillTimer3 . . . 031A Abs Null ROM
868 -879
osKillTimerExit . 0321 Abs Null ROM
870 -884
osKillTimerOK . . 0320 Abs Null ROM
874 878 -882
osMain . . . . . 046B Abs Null ROM
1169 1171 -1201 1209 1211 1228
osMainExit . . . . 0495 Abs Null ROM
1208 -1235
osMainSwitch . . . 0490 Abs Null ROM
1223 -1229
osPOP . . . . . 0239 Abs Null ROM Pub
172 -675
osPOPErrror . . . . 0256 Abs Null ROM
683 -693
osPOPExit . . . . . 0258 Abs Null ROM
692 -696
osPUSH . . . . . 0219 Abs Null ROM Pub
171 -644
osSMTT_Exit . . . . 0276 Abs Null ROM
723 -728
osSMTT_Off . . . . 026D Abs Null ROM
718 -724
osSetCommChannel . 0339 Abs Null ROM Pub
173 -961
osSetMaxTaskTime . 025E Abs Null ROM Pub
169 -715
osSetTimer . . . . 0279 Abs Null ROM Pub
176 -749
osSetTimer1 . . . . 0296 Abs Null ROM
-770
osSetTimer2 . . . . 029E Abs Null ROM

```

772	-777								
osSetTimer3	. . .	02A6	Abs	Null	ROM				
779	-784								
osSetTimerAddress		02C1	Abs	Null	ROM				
797	799	-806							
osSetTimerExit	. .	02FB	Abs	Null	ROM				
786	-848								
osSetTimerPeriod	. .	02C7	Abs	Null	ROM				
805	-811								
osSetTimerStart	. .	02AE	Abs	Null	ROM				
776	783	-791							
osSetUART2	. . . .	0396	Abs	Null	ROM				
1002	-1021								
osSetUARTEExit	. .	03D6	Abs	Null	ROM				
1047	-1052								
osSetUARTFail	. .	03D1	Abs	Null	ROM				
983	-1048								
osSignal	. . . . .	0324	Abs	Null	ROM			Pub	
170	-897								
osSignalEnd	. . .	0336	Abs	Null	ROM				
911	-914								
osSignalError	. .	0334	Abs	Null	ROM				
900	906	-912							
osSignalOK	. . . .	0330	Abs	Null	ROM				
905	-908								
osStart	. . . . .	043C	Abs	Null	ROM			Pub	
165	-1154								
osStartEnd	. . . .	0468	Abs	Null	ROM				
1184	-1187								
osStartError	. . .	0466	Abs	Null	ROM				
1158	1162	1166	-1185						
osStartTask	. . .	0078	Abs	Null	ROM			Pub	
167	-295								
osStartTaskEnd	. .	008C	Abs	Null	ROM				
308	-311								
osStartTaskError	. .	008A	Abs	Null	ROM				
302	-309								
osStopTask	. . . .	0093	Abs	Null	ROM			Pub	
168	-327								
osStopTaskEnd	. .	00A1	Abs	Null	ROM				
336	-339								
osStopTaskError	. .	009F	Abs	Null	ROM				
330	-337								
osUARTGetChar	. .	03D7	Abs	Null	ROM			Pub	
174	-1066								
osUARTRxCh1	. . .	03E3	Abs	Null	ROM				
1073	-1075								
osUARTRxExit	. . .	0400	Abs	Null	ROM				
1089	-1094								
osUARTRxFail	. . .	03FB	Abs	Null	ROM				
1074	1078	-1090							
osUARTSend	. . . .	0401	Abs	Null	ROM			Pub	
175	-1106								
osUARTSendBusy	. .	0430	Abs	Null	ROM				
1117	-1133								
osUARTSendCh1	. .	0409	Abs	Null	ROM				
1110	-1112								
osUARTSendError	. .	0438	Abs	Null	ROM				
1111	-1139								
osUARTSendExit	. .	043B	Abs	Null	ROM				
1132	1138	-1142							
osUnassigned	. . .	00CF	Abs	Null	ROM				
205	206	207	208	209	210	-410			
PE	. . . . .	0005	Abs	Null					
-4									
PEN	. . . . .	0007	Abs	Null					
-4	1034								

```

PORTCC . . . . . 00D9 Abs Byte
-4
PORTCD . . . . . 00D8 Abs Byte
-4
PORTCP . . . . . 00DA Abs Byte
-4
PORTD . . . . . 00DC Abs Byte
-4
PORTGC . . . . . 00D5 Abs Byte
-4
PORTGD . . . . . 00D4 Abs Byte
-4
PORTGP . . . . . 00D6 Abs Byte
-4
PORTI . . . . . 00D7 Abs Byte
-4
PORTLC . . . . . 00D1 Abs Byte
-4 1026 1027
PORTLD . . . . . 00D0 Abs Byte
-4
PORTLP . . . . . 00D2 Abs Byte
-4
PSEL0 . . . . . 0005 Abs Null
-4 1036
PSEL1 . . . . . 0006 Abs Null
-4
PSR . . . . . 00BE Abs Byte
-4 1001
PSW . . . . . 00EF Abs Byte
-4 223 435 488 595 622 624 625 627 1183
pCOM1RxRead . . . 0031 Rel Byte SEG
-140 1041 1076 1085
pCOM1RxWrite . . . 0030 Rel Byte SEG
-139 562 569 1040 1077
pCOM1TxRead . . . 0043 Rel Byte SEG
-143 547 557 1039 1125
pCOM1TxWrite . . . 0042 Rel Byte SEG
-142 555 1038 1118 1127
pStack . . . . . 0008 Rel Byte SEG
-108 202 651 657 681 685 686
RBFL . . . . . 0001 Abs Null
-4
RBIT9 . . . . . 0003 Abs Null
-4
RBUF . . . . . 00B9 Abs Byte
-4 214 564
RCVG . . . . . 0000 Abs Null
-4
RDX . . . . . 0003 Abs Null
-4
Restore . . . . . 0200 Abs Null ROM
443 451 469 471 485 491 493 511 513 530 532 558 570 572 592 596
-617
S . . . . . 00FF Abs Byte
-4 185 193 212 245 246 281 297 298 314 328 340 437 455 480 490
499 510 519 529 538 546 561 578 586 620 646 647 661 676 677 698
716 729 750 849 862 885 898 915 962 963 1044 1050 1067 1068 1087 1092
1113 1114 1130 1136 1155 1188 1205 1236
S0 . . . . . 0000 Abs Null
-4
S1 . . . . . 0001 Abs Null
-4
SI . . . . . 0006 Abs Null
-4
SIO . . . . . 00E9 Abs Byte
-4
SIOR . . . . . 00E9 Abs Byte

```

```

-4
SK . . . . . 0005 Abs Null
-4
SO . . . . . 0004 Abs Null
-4
SSEL . . . . . 0004 Abs Null
-4
STP2 . . . . . 0007 Abs Null
-4 1032
STP78 . . . . . 0006 Abs Null
-4
ServiceExternal . 01CF Abs Null ROM
-594 613
ServiceMicrowire . 01CA Abs Null ROM
-590 609
ServiceSoftware . 01C4 Abs Null ROM
-582 615
ServiceTimerT0 . . 0112 Abs Null ROM
-453 612
ServiceTimerT0Exit 013C Abs Null ROM
459 -483
ServiceTimerT1 . . 0143 Abs Null ROM
-489 505
ServiceTimerT1A . 0140 Abs Null ROM
-487 611
ServiceTimerT1B . 0154 Abs Null ROM
-503 610
ServiceTimerT2 . . 015B Abs Null ROM
-509 524
ServiceTimerT2A . 0158 Abs Null ROM
-507 605
ServiceTimerT2B . 016C Abs Null ROM
-522 604
ServiceTimerT3 . . 0173 Abs Null ROM
-528 543
ServiceTimerT3A . 0170 Abs Null ROM
-526 603
ServiceTimerT3B . 0184 Abs Null ROM
-541 602
ServiceUART_RX . . 01A4 Abs Null ROM
-560 607
ServiceUART_TX . . 0188 Abs Null ROM
-545 606
ServiceUnused . . 010A Abs Null ROM
-441 600 608 614
ServiceWakeup . . 010D Abs Null ROM
-445 601
Start . . . . . 0000 Abs Null ROM
-182 463 1239
TOEN . . . . . 0004 Abs Null
-4 722 725
TOPND . . . . . 0005 Abs Null
-4 454
T1A . . . . . 0003 Abs Null
-4
T1B . . . . . 0002 Abs Null
-4
T1C0 . . . . . 0004 Abs Null
-4 4 842 873
T1C1 . . . . . 0005 Abs Null
-4 4
T1C2 . . . . . 0006 Abs Null
-4 4
T1C3 . . . . . 0007 Abs Null
-4 4
T1ENB . . . . . 0000 Abs Null
-4

```

T1PNDB	. . . . .	0001	Abs	Null		
-4	504					
T1RAHI	. . . . .	00ED	Abs	Byte		
-4						
T1RALO	. . . . .	00EC	Abs	Byte		
-4						
T1RBHI	. . . . .	00E7	Abs	Byte		
-4						
T1RBLO	. . . . .	00E6	Abs	Byte		
-4	833					
T2A	. . . . .	0004	Abs	Null		
-4						
T2B	. . . . .	0005	Abs	Null		
-4						
T2C1	. . . . .	0005	Abs	Null		
-4						
T2C2	. . . . .	0006	Abs	Null		
-4						
T2C3	. . . . .	0007	Abs	Null		
-4						
T2CNTRL	. . . . .	00C6	Abs	Byte		
-4	220	508	523		844	877
T2CO	. . . . .	0004	Abs	Null		
-4	844	877				
T2ENA	. . . . .	0002	Abs	Null		
-4						
T2ENB	. . . . .	0000	Abs	Null		
-4						
T2PNDA	. . . . .	0003	Abs	Null		
-4	508					
T2PNDB	. . . . .	0001	Abs	Null		
-4	523					
T2RAHI	. . . . .	00C3	Abs	Byte		
-4						
T2RALO	. . . . .	00C2	Abs	Byte		
-4						
T2RBHI	. . . . .	00C5	Abs	Byte		
-4						
T2RBLO	. . . . .	00C4	Abs	Byte		
-4						
T3A	. . . . .	0006	Abs	Null		
-4						
T3B	. . . . .	0007	Abs	Null		
-4						
T3C1	. . . . .	0005	Abs	Null		
-4						
T3C2	. . . . .	0006	Abs	Null		
-4						
T3C3	. . . . .	0007	Abs	Null		
-4						
T3CNTRL	. . . . .	00B6	Abs	Byte		
-4	221	527	542		846	881
T3CO	. . . . .	0004	Abs	Null		
-4	846	881				
T3ENA	. . . . .	0002	Abs	Null		
-4						
T3ENB	. . . . .	0000	Abs	Null		
-4						
T3PNDA	. . . . .	0003	Abs	Null		
-4	527					
T3PNDB	. . . . .	0001	Abs	Null		
-4	542					
T3RAHI	. . . . .	00B3	Abs	Byte		
-4						
T3RALO	. . . . .	00B2	Abs	Byte		
-4						
T3RBHI	. . . . .	00B5	Abs	Byte		

```

-4
T3RBLO . . . . . 00B4 Abs Byte
-4
TBMT . . . . . 0000 Abs Null
-4
TBUF . . . . . 00B8 Abs Byte
-4      550
TC1 . . . . . 0007 Abs Null
-4
TC2 . . . . . 0006 Abs Null
-4
TC3 . . . . . 0005 Abs Null
-4
TDX . . . . . 0002 Abs Null
-4
TMR1HI . . . . . 00EB Abs Byte
-4
TMR1LO . . . . . 00EA Abs Byte
-4      815
TMR2HI . . . . . 00C1 Abs Byte
-4
TMR2LO . . . . . 00C0 Abs Byte
-4      817
TMR3HI . . . . . 00B1 Abs Byte
-4
TMR3LO . . . . . 00B0 Abs Byte
-4      819
TPND . . . . . 0005 Abs Null
-4      488
TRUN . . . . . 0004 Abs Null
-4
WDOUT . . . . . 0001 Abs Null
-4
WDSVR . . . . . 00C7 Abs Byte
-4
WEN . . . . . 0002 Abs Null
-4
WKEDG . . . . . 00C8 Abs Byte
-4
WKEN . . . . . 00C9 Abs Byte
-4
WKPND . . . . . 00CA Abs Byte
-4      450
WPND . . . . . 0003 Abs Null
-4      591
wAddress . . . . . 000C Rel Word SEG
-113 255 257 270 272 752 754 763 765 967 969 977 979
wIntCallBack . . . 001B Rel Word SEG
-129
wPeriod . . . . . 0011 Rel Word SEG
-123 760 762 821 823 825 827 835 837
wTimer1CallBack . 0015 Rel Word SEG
-125 204 495 497 774
wTimer2CallBack . 0017 Rel Word SEG
-126 515 517 781 788
wTimer3CallBack . 0019 Rel Word SEG
-127 534 536
wTimerCallBack . . 0013 Rel Word SEG
-124 756 758 794 795 807 809
wUARTRxCallBack . 001E Rel Word SEG
-136 574 576 971 973
XBIT9 . . . . . 0005 Abs Null
-4
XMTG . . . . . 0001 Abs Null
-4
XRCLK . . . . . 0003 Abs Null
-4


```

```
XTCLK . . . . . 0002 Abs Null
-4
**** Errors:      0, Warnings:      0
Checksum:      0xF108
Byte Count:    0x045D (1117)
Input File:    mtk88815.asm
Output File:   mtk88815.obj
Memory Model:  Large
Chip:          888EG
```

# LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

 <p><b>National Semiconductor Corporation</b> Americas Tel: 1-800-272-9959 Fax: 1-800-737-7018 Email: support@nsc.com</p> <p>www.national.com</p>	<p><b>National Semiconductor Europe</b> Fax: +49 (0) 1 80-530 85 86 Email: europe.support@nsc.com Deutsch Tel: +49 (0) 1 80-530 85 85 English Tel: +49 (0) 1 80-532 78 32 Français Tel: +49 (0) 1 80-532 93 58 Italiano Tel: +49 (0) 1 80-534 16 80</p>	<p><b>National Semiconductor Asia Pacific Customer Response Group</b> Tel: 65-2544466 Fax: 65-2504466 Email: sea.support@nsc.com</p>	<p><b>National Semiconductor Japan Ltd.</b> Tel: 81-3-5639-7560 Fax: 81-3-5639-7507</p>
--	---	--	---