COP8

Application Note 1099 Creating a Multitasking Kernel for the COP8 Microcontroller

Literature Number: SNOA010
Creating a Multitasking Kernel for the COP8 Microcontroller

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 predetermined 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 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:

\[ T_{\text{max}} = n \times T_{\text{cycle}} \times 4096 \]

where \( n \) is the value passed to 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-

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 (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:

\[ T_{\text{max}} = n \times T_{\text{cycle}} \times 4096 \]

where \( n \) is the value passed to 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-
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 with 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 subroutines 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 is created using `osSetTimer`, which also assigns a callback routine when the time period ends. 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.

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”.

```
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 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?
```

www.national.com
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bDigitCount</td>
<td>Provide a counter to keep track of which digit is in use</td>
</tr>
<tr>
<td>bDispBuffer</td>
<td>Provide a buffer for the characters</td>
</tr>
</tbody>
</table>

The timer will be assigned in the INITIALIZE routine.

```asm
call osSetTimer
IFEQ A, #0
JP GetTimerFailure
X A, hRefreshTimer
(Continue...) ; Continue to initialize other stuff...
```

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

```asm
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
LAI D ; 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:

ScanKeysLoop:
LD A, bScanCode ; Save back for next pass
LD bScanCode, #X'FF ; Scan code for no key pressed.
JP ScanKeysExit ; No keys pressed... return to OS
ScanKeyHit:

ScanKeysLoopsLoop:
LD 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

ScanKeyHit:

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
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.
<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>osStartTask</strong></td>
<td>This routine will start a task thread running.</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>Accumulator contains the task number to start.</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>Accumulator is zero if successful, non-zero if failed.</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>For a task to execute, this routine must be called. Otherwise, the task is skipped and never executed.</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osStopTask</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>This routine will stop a task thread from running.</td>
</tr>
<tr>
<td><strong>Entry:</strong></td>
<td>Accumulator contains the task number to stop.</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>Accumulator is zero if successful, non-zero if failed.</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>Use this routine to suspend a task from running.</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osStart</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>(None required)</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>(Should not return unless fatal system error)</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>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...</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osMain</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>(None required)</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>(Never unless really bad things have happened...)</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>(Used by OS only)</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osGetTaskAddress</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>Accumulator holds task-1</td>
</tr>
<tr>
<td><strong>Entry:</strong></td>
<td>B points to task entry address record</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>(Used by OS only)</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>(Used by OS only)</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osGetFirstTask</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>(None)</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>First running task number (1-n) in 'A' if successful, zero if it fails</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>(Used by OS only)</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osGetNextTask</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>(None)</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>next running task number (1-n) in 'A' if successful, zero if it fails</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>(Used by OS only)</td>
</tr>
<tr>
<td><strong>Routine:</strong></td>
<td>osUnassigned</td>
</tr>
<tr>
<td><strong>Use:</strong></td>
<td>(None)</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>(None)</td>
</tr>
<tr>
<td><strong>Comments:</strong></td>
<td>(Used by OS only)</td>
</tr>
</tbody>
</table>

**Routine:** osSetTimer

**Use:** This routine starts a timer with a call back routine.

**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 holds handle number for timer (1-n) if successful
Accumulator is zero (0) if resource unavailable

**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:** osKillTimer

**Use:** This routine stops a timer and frees its resource.

**Entry:** Accumulator contains the non-zero timer handle.

**Returns:** Accumulator is zero if successful, non-zero if failed.

**Comments:** When a task is done using a timer resource, it must release this resource by calling this routine.

**Routine:** osSignal

**Use:** This routine will move a task to the top of the queue.

**Entry:** Accumulator contains the task number to signal next.

**Returns:** Accumulator is zero if successful, non-zero if failed.

**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:** osSetUART

**Use:** This routine will capture a UART to a task and setup the baud rate, framing, data bits, and buffers.

**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 Tables 1, 2)
2. PUSH LSB of UART RX Call Back routine
3. PUSH MSB of UART RX Call Back routine

**Returns:** If successful, accumulator will return the handle to the UART (1-n).
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**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>Baud Rate — See BAUD Rate Table</td>
</tr>
<tr>
<td>4</td>
<td>Data Bits (0=7, 1=8)</td>
</tr>
<tr>
<td>5</td>
<td>Stop Bits (0=1, 1=2)</td>
</tr>
<tr>
<td>6</td>
<td>Parity (0=none, 1=On)</td>
</tr>
<tr>
<td>7</td>
<td>Parity Type (0=Odd, 1=Even)</td>
</tr>
</tbody>
</table>

**TABLE 2. Baud Rate Bits**

<table>
<thead>
<tr>
<th>Bits 3-0</th>
<th>Baud Rate (Bits per Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>110</td>
</tr>
<tr>
<td>0001</td>
<td>134.5</td>
</tr>
<tr>
<td>0010</td>
<td>150</td>
</tr>
<tr>
<td>0011</td>
<td>300</td>
</tr>
<tr>
<td>0100</td>
<td>600</td>
</tr>
<tr>
<td>0101</td>
<td>1200</td>
</tr>
<tr>
<td>0110</td>
<td>2400</td>
</tr>
<tr>
<td>0111</td>
<td>4800</td>
</tr>
<tr>
<td>1000</td>
<td>7200</td>
</tr>
</tbody>
</table>

**TABLE 3-0**

<table>
<thead>
<tr>
<th>Baud Rate (Bits per Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001 9600</td>
</tr>
<tr>
<td>1010 19200</td>
</tr>
<tr>
<td>1011 Reserved — Defaults to 9600</td>
</tr>
<tr>
<td>1100 Reserved — Defaults to 9600</td>
</tr>
<tr>
<td>1101 Reserved — Defaults to 9600</td>
</tr>
<tr>
<td>1110 Reserved — Defaults to 9600</td>
</tr>
<tr>
<td>1111 Reserved — Defaults to 9600</td>
</tr>
</tbody>
</table>

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**
Scheduler

FIGURE 2. Cooperative Multitasking Structure

<table>
<thead>
<tr>
<th>This Task</th>
<th>Next Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

FIGURE 3. COP8SGR Multitasking Kernel Architecture
FIGURE 4. Multiplexed 7 Segment Display Task Hookup

FIGURE 5. Keyboard Scanning Task Hookup
TITLE MTK, 'Multi Tasking Kernal'

FILENAME: MTK88815.ASM
Multi-Tasking Kernal for simple multithreaded applications
Copyright (C) 1996, 1997 National Semiconductor Corporation

CODE HISTORY

Comment Date Version
--
Code created by Rick Zarr 06/03/96 1.00
Conversion to 888 core (COP8SAC) 02/27/97 1.10
Additions for use with article 08/07/97 1.20
Conversion to COP8SGR 01/03/98 1.30
Additions for Supervised cooperative 01/07/98 1.40
Updates to fix bugs 01/29/98 1.41
Additions to UART functions 02/24/98 1.50

Notes:
A) Naming conventions:
The following naming methods are used throughout this source code.
This makes classifying the variables easier when looking through
the listing. It also helps prevent association errors since the
assembler cannot tell you if you've associated an incorrect variable
with an operation. For example, if you wrote the following:

LD A, xyz ; Get the data in variable xyz

and you made the mistake that xyz was in effect a constant, the
assembler would load the location in memory that was pointed to by
the constant. This means you might get something like this:

LD A, 3 ; Get the data in variable xyz

instead of the intended variable location. This naming convention
can help prevent this. Here is the same thing using the names. I
have defined two things, one is the variable name and size and the
other is its default value. They can be named almost the same thing
helping to keep track of what's what!

LD A, cXYZ ; Load the default value for xyz
X A, bXYZ ; Save it into the variable

Here's my definitions used in most of the source code I write.
Feel free to adopt this methodology or create your own.

Variable prefix naming conventions:

a : ARRAY (Group of bytes. Example: aData)
b : BYTE VALUE (Single byte value. Example: bData)
w : WORD (Variable defined as DSW. Example: wTest)
f : FLAG BIT (Bit within byte. Example: fWorking)
c : CONSTANT (ROM defined value Example: cValue)
p : POINTER (Byte long pointer into RAM. Example: pBuffer)
pc : POINTER CONSTANT (Pointer into EEPROM. Example: pcMaxMinV)
r : REGISTER (CPU register R0-RA Example: rCounter)
io : I/O PIN (I/O pin on COP8 controller Example: iODa)
i : INPUT PIN (Input pin on COP8 controller Example: iSwitch)
o : OUTPUT PIN (Output pin on COP8 controller Example: oLED1)

..INCLD COP88SEG.INC ; Example 888 device for demo of MTK

CONSTANTS
Example cConstant = n; Setup up constants like this

006F cTopOfRAM = 0x6F; Top of RAM for COP87L88EG
007F cTopOfSysRAM = 0x7F; Top of OS RAM for COP87L88EG
0001 cSystemSegment = 1; Current system RAM segment
0003 cRecSize = 3; Size of task records in bytes
000A cMaxTasks = 10; Maximum number of tasks that can be loaded
0060 cBaudPrescale = 0x60; UART Prescaler value 6.5 (01100) for 10MHz
0010 cCOM1TxBufLen = 16; COM1 Transmitter FIFO length
0010 cCOM1RxBufLen = 16; COM1 Receiver FIFO Length

;*********************************************************************

; OPERATING SYSTEM MESSAGES
;*********************************************************************

0000 mNone = X’00; Idle message
0001 mTimer = X’01; Timer tick / service
0002 mUART = X’02; UART Service
000F mTaskFailure = X’0F; Task monitor service
0010 mUser = X’10; Beginning of user messages

;**********************************************************************

; VARIABLES
; Example:
; wWord : .DSW 1; Word format for 16 bit location
; bByte : .DSB 1; Byte format for 8 bit location

0000 aStore : .DSB 8; Local store for working with data
0008 pStack : .DSB 1; OS Data Stack Pointer

0009 bTask : .DSB 1; Currently running task number (1-n)
000A bTaskNew : .DSB 1; Used by OS, Last task number added
000B bTaskNext : .DSB 1; Next active task to switch in
000C wAddress : .DSW 1; 16 bit address (MSB/LSB)
000E bMaxTaskTime : .DSB 1; Max tick count for task monitor
000F bTaskTime : .DSB 1; Remaining

0010 fStatus : .DSB 1; Status byte for kernel
0003 fUART = 3; Bit 3: UART 1 in use
0002 fTimer3 = 2; Bit 2: Timer 3 in use
0001 fTimer2 = 1; Bit 1: Timer 2 in use
0000 fTimer1 = 0; Bit 0: Timer 1 in use

0011 wPeriod : .DSW 1; Work area for timer period
0013 wTimerCallBack : .DSW 1; Work area for call back
0015 wTimer1CallBack : .DSW 1; Timer 1 callback address
0017 wTimer2CallBack : .DSW 1; Timer 2 callback address
0019 wTimer3CallBack : .DSW 1; Timer 3 callback address
001B wIntCallBack : .DSW 1; Hardware interrupt callback

001D bUARTSettings : .DSB 1; Current UART settings
0004 fDataBits = 4; Data bits (0=7, 1=8)
0005 fStopBits = 5; Stop Bits (0=1, 1=2)
0006 fParitySel = 6; Parity Select (0=none, 1=On)
0007 fParityType = 7 ; Parity Type (0=Odd, 1=Even)
001E wUARTRxCallBack : .DSW 1 ; UART RX Call back address
0020 bCOM1RxBuffer : .DSB cCOM1RxBufLen ; Setup buffer for RX
0030 pCOM1RxWrite : .DSB 1 ; Write pointer to RX buffer
0031 pCOM1RxRead : .DSB 1 ; Read pointer to RX buffer
0032 bCOM1TxBuffer : .DSB cCOM1TxBufLen ; Setup buffer for TX
0042 pCOM1TxWrite : .DSB 1 ; Write pointer to TX buffer
0043 pCOM1TxRead : .DSB 1 ; Read pointer to TX buffer
0044 bCOM1Status : .DSB 1 ; Status byte for COM1
0007 fTxBusy = 7 ; Tx FIFO Full
0006 fRxBusy = 6 ; Rx FIFO Full

************************************************************************

; TASK RECORD STRUCTURE

; fTaskStatus : .DSB 1 ; Flags for each task
; fActive = 0 ; Bit 0 : State (0=Stopped, 1=Running)
; bAddrLSB : .DSB 1 ; Task Address LSB
; bAddrMSB : .DSB 1 ; Task Address MSB

************************************************************************

0045 aTaskRecs : .DSB cRecSize * cMaxTasks ; Array of task records
0000 fActive = 0 ; Active flag for task

;********************************************************************
; IMPORT FUNCTIONS ********
;************************************************************************

.EXTRN Initialize:ROM ; Inializes application code

;********************************************************************
; EXPORT FUNCTIONS ********
;************************************************************************

.PUBLIC osStart ; Starts the MTK running
.PUBLIC osAddTask ; Adds a task to the MTK task list
.PUBLIC osStartTask ; Starts a task running
.PUBLIC osStopTask ; Stops a task

.PUBLIC osSetMaxTaskTime ; Sets the maximum time a task can run before being killed
.PUBLIC osSignal ; Moves thread to top of queue
.PUBLIC osPUSH ; PUSHes a byte onto the OS stack from ‘A’
.PUBLIC osPOP ; POPs a byte from the OS stack to ‘A’

.PUBLIC osSetCommChannel ; Captures and initializes a communications resource
.PUBLIC osUARTGetChar ; Returns a byte from the UART RX queue
.PUBLIC osUARTSend ; Sends a character to the UART
.PUBLIC osSetTimer ; Captures and sets up a timer resource
.PUBLIC osKillTimer ; Stops and frees a timer resource

;******************************************** CODE BEGINS **********

.CODE SECTION

.FORM 'CODE SECTION'

0000 .SECT CODE, ROM, ABS=0x0000

; Start: LD SF, #cTopOfRAM ; Top of Stack for COP87L88EG
0000 .SECT CODE, ROM, ABS=0x0000

; ClearRAM:
0002 DD6F 3 Start: LD SF, #cTopOfRAM ; Top of Stack for COP87L88EG
0002 DF00 3 LD S, #0 ; Point to RAM segment 0
0004 9F6F 2 LD B, #cTopOfRAM ; Setup pointer into Main RAM
0006 ClearRAMLoop1:
0006 9E00 2 LD [B], #000 ; Clear RAM location
0008 CE 3 DRSZ B ; Update counter, skip jump if done
0009 FC 3 JP ClearRAMLoop1 ; Continue until done
000A BC0000 3 LD 0000 , #00 ; Clear last location
000D DD6F 3 Start: LD SF, #cTopOfRAM ; Top of Stack for COP87L88EG
000D DF00 3 LD S, #1 ; Point to RAM segment 1
000F 9F7F 2 LD B, #cTopOfSysRAM ; Setup pointer into OS RAM
0011 ClearRAMLoop2:
0011 9E00 2 LD [B], #000 ; Clear RAM location
0013 CE 3 DRSZ B ; Update counter, skip jump if done
0014 FC 3 JP ClearRAMLoop2 ; Continue until done
0015 BC0000 3 LD 0000 , #00 ; Clear last location

.

www.national.com
201 0018 BC08F 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 BCEB65 3 LD T3CNTRL, #B'10000101 ; Setup timer 1A
222 003F BCEF81 3 LD T1CNTRL, #B'00000000 ; Setup timer 0, 1B
223 0042 BCE011 3 LD PSW, #B'00010001 ; Setup PSW, Timer 1A, GIE
224
225 0045 AC0000 > 4 X JP Initialize ; Go setup tasks
226
227 ;*********************************************************************
228 ; Multitasking kernel routines
229 ;*********************************************************************
230
231 ; Routine: osAddTask
232 ; Use: This routine will add a task entry point to the task list.
233 ;
234 ; Entry: Stack (SP) holds return address and data. Push LSB first
235 ; and then push MSB of task entry point callback routine.
236 ;
237 ; Returns: Accumulator returns assigned task number if successful,
238 ; zero if the task was not added.
239 ; Alters: 'A', 'B', and the SP
240 ;
241 ;*********************************************************************
242 0048 osAddTask:
243 0048 9DFF 3 LD A, S ; Get current RAM segment
244 004A DF01 3 LD S, #1 ; Switch to OS data segment
245 004C 9C00 3 R X A, aStore+0 ; Save for later to restore segment
246 004E 9D0A 3 R LD A, bTaskNew ; Update New task counter
247 0050 8A 1 INC A ; Increment to next one
248 0051 930A 2 IFGT A, #cMaxTasks ; Check if larger than max task
249 0053 1C 3 JP osAddTaskError ; yes, Report error
250 0054 9C0A 3 R X A, bTaskNew ; No, save it back
251
252 ;
253 0056 8C 3 POP A ; Get Return Address MSB
254 0057 9C0D 3 R X A, wAddress+1 ; Save in MSB of wAddress
255 0059 8C 3 POP A ; Get Return Address LSB
256 005A 9C0C 3 R X A, wAddress ; Save in LSB of wAddress
257
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

www.national.com 14
; Store LSB, increment B
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 9DA0 3 R LD A, bTaskNew ; Get the current task number
275 006F 01 3 JP osAddTaskEnd ; Return handle of task

276 osAddTaskError:
277 0070 64 1 CLR A ; Report error
278 0071 osAddTaskEnd:
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 ; Return to calling routine
284
285 ;***************************************************************
286 ; Routine: osStartTask
287 ; Use: This routine will start running a task thread.
288 ; Entry: Accumulator contains the task number to start.
289 ; Returns: Accumulator is zero if successful, non-zero if failed.
290 ;***************************************************************
291 osStartTask:
292 0078 67 3 PUSH A ; Save 'A' for now
293 0079 9DFF 3 LD A, S ; Get current segment
294 007B DF01 3 LD S, #1 ; Switch to OS RAM Page
295 007D 9C00 3 R X A, aStore+0 ; Save old RAM segment
296 007F 8C 3 POP A ; Get task number off main stack
297 0080 BD0A83 4 R IFGT A, bTaskNew ; Is the task number valid?
298 0083 06 3 JP osStartTaskEnd ; No, don't start
299 ;
300 0084 8B 1 DEC A ; Use corrected pointer (p-1)
301 0085 30A4 5 JSR osGetTaskAddress ; Get a pointer into the task buffer
302 0087 78 1 SBIT fActive, [B] ; Set Start Task flag in MSB
303 0088 64 1 CLR A ; Clear 'A' to signal start
304 0089 02 3 JP osStartTaskEnd ; End routine
305 osStartTaskError:
306 008A 9BFF 2 LD A, #0xFF ; Signal error (non-zero)
307 osStartTaskEnd:
308 008C 67 3 PUSH A ; Save return value on main stack
309 008D 9D00 3 R LD A, aStore+0 ; Get old RAM segment from store
310 008F 9CFF 3 X A, S ; Restore old RAM segment
311 0091 8C 3 POP A ; Get return value
312 0092 8E 5 RET ; Return to calling task
313
314 ;***************************************************************
315 ; Routine: osStopTask
316 ; Use: This routine will stop a task thread from running.
317 ; Entry: Accumulator contains the task number to stop.
318 ; Returns: Accumulator is zero if successful, non-zero if failed.
319 ;***************************************************************
320 osStopTask:
321 0093 DF01 3 LD S, #1 ; Switch to OS RAM Page
322 0095 BD0A83 4 R IFGT A, bTaskNew ; Is the task number valid?
323 0098 06 3 JP osStopTaskEnd ; No, don’t start
324 ;
325 0099 8B 1 DEC A ; Use corrected pointer (p-1)
326 009A 30A4 5 JSR osGetTaskAddress ; Get a pointer into the task buffer
327 009C 78 1 SBIT fActive, [B] ; Start Task

15 www.national.com
CLR A ; Clear 'A' to signal start
JP osStopTaskEnd ; End routine
LD A, #0xFF ; Signal error (non-zero)
LD S, #0 ; Switch back to user RAM area
RET ; Return to calling task

;*********************************************************************
; Get Task Address Subroutine
; (Used by OS only)
; osGetTaskAddress : A is task-1, B returns address to first entry
;*********************************************************************
LD A, #0 ; First task record
PUSH A ; Save it on the stack
JP osGetNextTaskLoop ; go see what’s enabled
LD A, bTask ; Current Task+1 (Already incremented)
PUSH A ; Save it for later
oGetTaskAddress:
LD A, B ; Use 'B' to do math
LD A, B ; Get task pointer back
ADD A, B ; Multiply x2 for pointer
ADD A, B ; Multiply x3 for pointer
ADD A, #aTaskRecs ; Add base address of record buffer
X A, B ; Use 'B' now as pointer
RET ;

;*********************************************************************
; Get First and Next running Task Subroutine
; (Used by OS only)
; osGetFirstTask : Returns first running task (1-n) in 'A' if successfull
; osGetNextTask : Returns next running task (1-n) in 'A' if successfull
; If either call fails, 'A' is zero
;*********************************************************************
LD A, #0 ; First task record
PUSH A ; Save it on the stack
JP osGetNextTaskLoop ; go see what’s enabled
LD A, bTask ; Current Task+1 (Already incremented)
PUSH A ; Save it for later
osGetTaskAddress:
LD A, bTask ; Go get the address of the task
jis osGetTaskAddressHit ; Yes, we got the next active one
LD A, bTask ; Hit them all?
LD A, bTask ; Past last one?
LD A, bTask ; Nothing is running. Bail...
PUSH A ; Save it for later
JP osGetNextTaskLoop ; Try again

;*********************************************************************
;*********************************************************************
Unassigned Callback Idle routine

This routine is used as a safety net in case a timer or other callback is unassigned and is executed. This will immediately return to the OS.

;*********************************************************************

; INTERRUPT ROUTINE
;*********************************************************************

; INTERRUPT HANDLER
; NOTES
; -----
; This routine is the entry point for all vectors. The VIS instruction will find the correct vector for interrupt. It is the responsibility of each service routine to clear the cause of the interrupt. A template is used for each of the existing interrupts. The default device for this source is the COP8A family. The interrupts for this device are all supported. If this MTK is migrated to another feature family device, simply replace the ‘ServiceUnused’ routine with the missing routines.

;*********************************************************************

; Force to 0xFF for interrupt
.ORG 0xFF

; Interrupt:
PUSH A ; Save 'A'
LD A, B ; Save 'B'
PUSH A ;
LD A, PSW ; Save 'PSW'
PUSH A ;
LD A, S ; Save segment
PUSH A ;
VIS ; Find interrupt vector

;*********************************************************************

; ServiceUnused:
RPND ; Incase it's set, clear SW pending
JP Restore ; Just return

;*********************************************************************

; ServiceWakeup:

; *** Check which 'L' Port pin caused the interrupt here ***

;*********************************************************************

; Clear all pending flags
LD WKPND, #0
JP Restore ; Return

;*********************************************************************

; Clear pending flag
LD S, #1 ; Switch to OS segment

; Decrement task watcher counter
IFGT A, #0 ; Did current running task die?
INC fActive, [B] ; Disable the task

;*********************************************************************

; Clear pending flag
LD S, #1 ; Switch to OS segment

; Decrement task watcher counter
IFGT A, #0 ; Did current running task die?
INC 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 ServiceTimerTOExit:
484 013C 9C0F 3 R  LD A, bTaskTime ; Save new count back to RAM
485 013E 2200 > 3 JP Restore ; Return
486 ;*********************************************************************
487 0140 ServiceTimerT1A:
488 0140 BDEF6D 4 RBIT TPND, 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 014D 9D15 3 R  LD A, wTimer1CallBack ; Push callback address on stack
496 014F 67 3  PUSH A ;
497 0150 9D16 3 R  LD A, wTimer1CallBack+1 ;
498 0152 67 3  PUSH A ;
499 0154 DF00 3 LD S, #0 ; Switch to User RAM for call back
500 0156 8E 5 RET ; Go service it, return to RESTORE
501
502 ;*********************************************************************
503 0158 ServiceTimerT2A:
504 0158 BDC66B 4 RBIT TPND, T2CNTRL ; Clear Pending flag
505 015B ServiceTimerT2:
506 ;*********************************************************************
507 0158 ServiceTimerT3A:
508 0158 BDC66B 4 RBIT TPND, T3CNTRL ; Clear Pending flag
509 015B ServiceTimerT3:
510 ;*********************************************************************
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+1] ; Get the data
550 018F 9CB8  3  X A, TBUF ; Move to UART Tx register
551 0191 BD446F  4 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 9CB9  3  X A, B ; Use ‘B’ as pointer
564 01AA 9802  2  R LD A, RBUF ; Get received character
565 01AC A2  2  X A, [B+1] ; Save into receive buffer
566 01AD 9DFE  3  LD A, B ; Update pointer
567 01AE 9230  2  R IFEQ A, #COM1RxBuffer+cCOM1RxBufLen; Past end of buffer
568 01B1 9820  2  R LD A, #COM1RxBuffer ; 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 BDE6F6  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

www.national.com
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 ; Timer1A vector
612 01F8 0112 .ADDRW ServiceTimerT0 ; Timer 0 vector
613 01FA 0154 .ADDRW ServiceExternal ; External pin vector
614 01FC 010A .ADDRW ServiceUnused ; (Reserved for future use)
615 01FE 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 ; Routine: osPUSH
635 ; Use: This routine pushes the 'A' register onto the OS data stack
636 ; Entry: Accumulator contains data byte to push
637 ; Returns: Accumulator contains original data (Like real PUSH)
638 ; Comments: Used for user temporary storage
639 ;*********************************************************************
640 0219 osPUSH:
641 0219 67 3 PUSH A ; Save 'A' for page switch
642 021A 9DFF 3 LD A, S ; Get old segment
643 021C DF01 3 LD S, #1 ; Switch to the OS Segment
644 021E 9C00 3 R X A, aStore+0 ; Save
645 0220 9DFF 3 LD A, B ; Save 'B' for now
646 0222 9C01 3 R X A, aStore+1 ;
647 0224 9D08 3 R LD A, pStack ; Get the stack pointer
648 0226 9CFE 3 X A, B ; Stick it into 'B' for pointer
649 0228 8C 3 POP A ; Get the data from the main stack
650 0229 67 3 PUSH A ; Put it back on the main stack for later
651 022A A3 2 X A, [R-] ; Push data, Adjust pointer
652 022B 9DFF 3 LD A, B ; Save new stack pointer
653 022D 9C08 3 R X A, pStack ; in pointer register
654 022F 9D01 3 R LD A, aStore+1 ; Restore 'B'
655 0231 9C0F 3 X A, B ;
656 0233 9D00 3 R LD A, aStore+0 ; Get old RAM segment
657 0235 9CFF 3 X A, S ; Restore the old segment
658 0237 8C 3 POP A ; return original 'A' value (like real PUSH)
659 0238 8E 5 RET ; Go do some more stuff...
660 ;*********************************************************************
661 ;*********************************************************************
662 www.national.com 20
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 9DFF 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 9C0E 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 BDE87C 4  SBIT T0EN, ICNTRL ; Turn on the timer interrupt
68A 0253 9C0F 3  X A, B ; to original value
68B 0255 02 3  JP osPOPExit ; Done, return
68C 0256 osPOPError:
68D 0256 64 1  CLR A ; Put '0' on stack for return value
68E 0257 67 3  PUSH A ; Save return value
68F 0258 osPOPExit:
709 0258 DF00 3  LD S, #0 ; Restore application segment
70A 025A 8E 5  RET ; Go do some more stuff...
70B
70C
70D ;*********************************************************************
70E ; Routine: osSetMaxTaskTime
70F ; Use: This routine enables the task watcher and sets the time limit.
710 ; Entry: Accumulator holds the time constant, if zero function disabled.
711 ; Returns: Accumulatorreturns assigned task number if successful,
712 ; zero if the task was not added.
713 ; Alters: 'A'
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 9D08 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  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 ;*********************************************************************
; Routine: osSetTimer
; Use: This routine start a timer with a call back routine. If the
; call back is set to zero, Task 1 (Main task) will get the
; call with the accumulator set to cmTimer, and 'B' with the
; timer handle.
;
; 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 holds handle number for timer (1-n) if successful
; Accumulator is zero (0) if resource unavailable
;
;*********************************************************************

0279 osSetTimer:
0279 DF01 3 LD S, #1 ; Switch Data RAM segment to OS
027B 8C 3 POP A ; Get MSB of calling routine
027C 9C0D 3 R X A, wAddress+1 ; Save for now
027E 8C 3 POP A ; Get LSB of calling routine
027F 9C0C 3 R X A, wAddress ; Save for now
0281 8C 3 POP A ; Get MSB of Call Back routine
0282 9C14 3 R X A, wTimerCallBack+1 ; Save for now
0284 8C 3 POP A ; Get LSB of Call Back routine
0285 9C13 3 R X A, wTimerCallBack ; Save for now
0287 8C 3 POP A ; Get MSB of time period
0288 9C12 3 R X A, wPeriod+1 ; Save for now
028A 8C 3 POP A ; Get MSB of time period
028B 9C11 3 R X A, wPeriod ; Save for now
028D 9D0C 3 R LD A, wAddress ; Restore the return address
028F 67 3 PUSH A ; Save LSB first
0292 9D0D 3 R LD A, wAddress+1 ;
0292 67 3 PUSH A ; Save MSB second
;
0293 64 1 CLR A ; Clear 'A'
0294 9F10 2 R LD B, #fStatus ; Find a free timer
0296 70 1 IFBIT fTimer1, [B] ; Timer 1 free?
0297 06 3 JP osSetTimer2 ; No, try timer 2
0298 78 1 SBIT fTimer1, [B] ; Capture timer
0299 9F15 2 R LD A, wTimer1CallBack ; yes, load and start timer 1
029B 9801 2 LD A, #1 ;
029D 10 3 JP osSetTimerStart ;
029E osSetTimer2:
029E 71 1 IFBIT fTimer2, [B] ; Timer 2 free?
029F 06 3 JP osSetTimer3 ; No, try timer 3
02A0 79 1 SBIT fTimer2, [B] ; Capture timer
02A1 9F17 2 R LD B, wTimer2CallBack ; yes, load and start timer 2
02A3 9802 2 LD A, #2 ;
02A5 08 3 JP osSetTimerStart ;
02A6 osSetTimer3:
02A6 72 1 IFBIT fTimer3, [B] ; Timer 2 free?
02A7 22FB > 3 JP osSetTimerExit ; No timers available, sorry...
02A9 7A 1 SBIT fTimer3, [B] ; Capture timer
02AA 9F17 2 R LD B, wTimer3CallBack ; yes, load and start timer 2
02AC 9802 2 LD A, #2 ;
02AE osSetTimerStart:
02AE 67 3 FUSH A ; Save timer handle
02AF A5 1 RC ; Check if zero
02B0 9D13 3 R LD A, wTimerCallBack ; Get return address
02B2 BD1480 4 R ADC A, wTimerCallBack+1 ; See if all zeros
02B5 88 1 IPC ; If there was a carry, good address
02B6 0A 3 JP osSetTimerAddress ;
02B7 9900 2 IFNE A, #0 ; If not zero, also good address
02B9 07 3 JP osSetTimerAddress ;
800

www.national.com 22
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 osSetTimerAddress:
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 osSetTimerPeriod:
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 02CB 9202 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 9F66 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 T1C0, CNTRL ; Yes, start it up
843 02F1 9202 2 IFEQ A, #2 ; Start timer 2?
844 02F3 BDC67C 4 SBIT T2C0, T2CNTRL ; Yes, start it up
845 02F6 9203 2 IFEQ A, #3 ; Start timer 3?
846 02FA BDB67C 4 SBIT T3C0, T3CNTRL ; Yes, start it up
847 osSetTimerExit:
848 02FB osKillTimer:
849 02FB DF00 3 LD S, #0 ; Switch data RAM segment back to user
850 02FD 8E 5 RET ; Return
851
852 ;*********************************************************************
853 ; Routine: osKillTimer
854 ; Use: This routine stop a timer and free it.
855 ; Entry: Accumulator contains the non-zero timer handle
856 ; Returns: Accumulator is zero if successful, non-zero if failed.
857 ;*********************************************************************
858 02FE osKillTimer:
859 02FE DF01 3 LD S, #1 ; Switch data segment to OS
860 02FF 9201 2 IFEQ A, #1 ; Kill Timer 1?
861 0300 9201 2 IFEQ A, #1 ; Kill Timer 1?
862 0302 09 3 JP osKillTimer1 ; Yes, stop it.
863 0303 9202 2 IFEQ A, #2 ; Kill Timer 2?
864 0305 0D 3 JP osKillTimer2 ; Yes, stop it.
865 0306 9203 2 IFEQ A, #3 ; Kill Timer 3?
Yes, stop it.
Failure, incorrect handle.
Return
osKillTimer1:
Free timer
Stop the timer
osKillTimerOK
Return
Free timer
Stop the timer
osKillTimerOK
osKillTimer3:
Free timer
Stop the timer
osKillTimerOK
osKillTimerExit:
Switch data segment back to user
Return
Routine: osSignal
Use: This routine will move a task to the top of the queue.
Entry: Accumulator contains the task number to signal next.
Returns: Accumulator is zero if successful, non-zero if failed.
hello
Table 1: 76543210 (Setup byte for UART - pushed on stack first)
| | | | | | Baud rate: || Data bits (0=7, 1=8)
Table 2: Baud Rate (bits 0–3)

<table>
<thead>
<tr>
<th>Bits 0-3</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>110 BPS</td>
</tr>
<tr>
<td>0001</td>
<td>134.5 BPS</td>
</tr>
<tr>
<td>0010</td>
<td>150 BPS</td>
</tr>
<tr>
<td>0011</td>
<td>300 BPS</td>
</tr>
<tr>
<td>0100</td>
<td>600 BPS</td>
</tr>
<tr>
<td>0101</td>
<td>1200 BPS</td>
</tr>
<tr>
<td>0110</td>
<td>2400 BPS</td>
</tr>
<tr>
<td>0111</td>
<td>4800 BPS</td>
</tr>
<tr>
<td>1000</td>
<td>7200 BPS</td>
</tr>
<tr>
<td>1001</td>
<td>9600 BPS</td>
</tr>
<tr>
<td>1010</td>
<td>19200 BPS</td>
</tr>
<tr>
<td>1111</td>
<td>(Reserved*)</td>
</tr>
</tbody>
</table>

Note * Defaults to 9600 BPS
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 3F01   .DW   319   ;   300   BPS
1007 037C 8002   .DW   640   ;   150   BPS
1008 0380 9F00   .DW   159   ;   600   BPS
1009 0382 4F00   .DW   79   ;   1200   BPS
1010 0384 2700   .DW   39   ;   2400   BPS
1011 0386 1300   .DW   19   ;   4800   BPS
1012 0388 0E00   .DW   14   ;   7200   BPS
1013 038A 9F00   .DW   9   ;   9600   BPS
1014 038C 9F00   .DW   9   ;   9600   BPS
1015 038E 9F00   .DW   9   ;   9600   BPS
1016 0390 3F01   .DW   319   ;   300   BPS
1017 0392 8002   .DW   640   ;   150   BPS
1018 0394 9F00   .DW   9   ;   9600   BPS
1019 0396 9F00   .DW   9   ;   9600   BPS
1020 0398 BDD16B 4 RBIT 3, PORTLC ; RX pin as input
1021 039B BDD17A 4 SBIT 2, PORTLC ; TX Pin as output
1022 039E BDD18C 4 RBIT CHLO, ENU ; Set to 8 data bits
1023 03C5 9D00 3 R LD A, #b'00000000' ; Clear attention mode
1024 03C7 9DFF 3 LD A, #b'00001000' ; 1 stop bit, TXD pin enabled, Async mode, Int Clks
1025 03C9 DF01 3 LD S, #1 ; Switch Data RAM segment to OS
1026 03CC BDD199 4 RBIT CHLO, ENU ; Set to 8 data bits
1027 03CF BDD19A 4 SBIT STP2, ENUI ; 2, Set to 2 stop bits
1028 03D0 BDD19B 4 SBIT PEN, ENU ; Yes, turn it on
1029 03D3 BDD19C 4 SBIT PSEL0, ENU ; Even, set it up.
1030 03D4 9D00 3 R LD A, #b'00000000' ; 1 stop bit, TXD pin enabled, Async mode, Int Clks
1031 03D7 9D00 3 R LD A, #b'00000000' ; 1 stop bit, TXD pin enabled, Async mode, Int Clks
1032 03DA BDD19D 4 SBIT STP2, ENUI ; 2, Set to 2 stop bits
1033 03DB BDD19E 4 SBIT PEN, ENU ; Yes, turn it on
1034 03DC BDD19F 4 SBIT PSEL0, ENU ; Even, set it up.
1035 03DE BDD200 4 SBIT PSSEL0, ENU ; Even, set it up.
1036 03DF BDD201 4 SBIT PSSEL0, ENU ; Even, set it up.
1037 03E0 BDD202 4 SBIT PSSEL0, ENU ; Even, set it up.
1038 03E1 BDD203 4 SBIT PSSEL0, ENU ; Even, set it up.
1039 03E2 BDD204 4 SBIT PSSEL0, ENU ; Even, set it up.
1040 03E3 BDD205 4 SBIT PSSEL0, ENU ; Even, set it up.
1041 03E4 BDD206 4 SBIT PSSEL0, ENU ; Even, set it up.
1042 03E5 BDD207 4 SBIT PSSEL0, ENU ; Even, set it up.
1043 03E6 BDD208 4 SBIT PSSEL0, ENU ; Even, set it up.
1044 03E7 BDD209 4 SBIT PSSEL0, ENU ; Even, set it up.
1045 03E8 BDD210 4 SBIT PSSEL0, ENU ; Even, set it up.
1046 03E9 BDD211 4 SBIT PSSEL0, ENU ; Even, set it up.
1047 03EA BDD212 4 SBIT PSSEL0, ENU ; Even, set it up.
1048 03EB BDD213 4 SBIT PSSEL0, ENU ; Even, set it up.
1049 03EC BDD214 4 SBIT PSSEL0, ENU ; Even, set it up.
1050 03ED BDD215 4 SBIT PSSEL0, ENU ; Even, set it up.
1051 03EE BDD216 4 SBIT PSSEL0, ENU ; Even, set it up.
1052 03EF BDD217 4 SBIT PSSEL0, ENU ; Even, set it up.
1053 03F0 BDD218 4 SBIT PSSEL0, ENU ; Even, set it up.
1054 03F1 BDD219 4 SBIT PSSEL0, ENU ; Even, set it up.
1055 03F2 BDD220 4 SBIT PSSEL0, ENU ; Even, set it up.
1056 03F3 BDD221 4 SBIT PSSEL0, ENU ; Even, set it up.
1057 03F4 BDD222 4 SBIT PSSEL0, ENU ; Even, set it up.
1058 03F5 BDD223 4 SBIT PSSEL0, ENU ; Even, set it up.
1059 03F6 BDD224 4 SBIT PSSEL0, ENU ; Even, set it up.
1060 03F7 BDD225 4 SBIT PSSEL0, ENU ; Even, set it up.
1061 03F8 BDD226 4 SBIT PSSEL0, ENU ; Even, set it up.
1062 03F9 BDD227 4 SBIT PSSEL0, ENU ; Even, set it up.
1063 03FA BDD228 4 SBIT PSSEL0, ENU ; Even, set it up.
1064 03FB BDD229 4 SBIT PSSEL0, ENU ; Even, set it up.
1065 03FC BDD230 4 SBIT PSSEL0, ENU ; Even, set it up.
1066 03FD BDD231 4 SBIT PSSEL0, ENU ; Even, set it up.
1067 03FE BDD232 4 SBIT PSSEL0, ENU ; Even, set it up.
1068 03FF BDD233 4 SBIT PSSEL0, ENU ; Even, set it up.

;*********************************************************************
; Routine: osUARTGetChar
; Use: This routine gets one character from the receive queue
; Entry: 'B' holds comm channel handle
; Returns: 'A' hold received character
; 'B' is zero if fail, otherwise handle remains.
; 'X' is altered.
;*********************************************************************
03D7 osUARTGetChar:
03D7 9DFF 3 LD A, #b'00000000' ; Save old segment
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 osUARTRxCh1:
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 03EB BA 3 LD A, \[X+\] ; Get the new character
1079 03EC 67 3 PUSH A ; Save on stack for now
1080 03ED 9DFC 3 LD A, X ; Get new pointer value
1081 03EF 9230 2 R IFEQ A, #bCOM1RxBuffer+cCOM1RxBufLen; Past end of buffer
1082 03F1 9820 2 R LD A, #bCOM1RxBuffer ; Reload start of buffer
1083 03F3 9D00 3 R LD A, aStore+0 ; Restore old RAM segment
1084 03F5 9CFF 3 X A, S ;
1085 03F9 8C 3 POP A ; Get data back
1086 03FA 05 3 JP osUARTRxExit ; Done
1087 ;*********************************************************************
1088 ; Routine: osUARTSend
1089 ; Use: This routine queues a byte of data for transmission by the UART.
1090 ; Entry: 'A' holds data byte, 'B' holds comm channel handle
1091 ; Returns: If successful, returns zero 'A'
1092 ; If failure, return non-zero in 'A'
1093 ;*********************************************************************
1094 0401 osUARTSend:
1095 0401 67 3 PUSH A ; Save data
1096 1097 0402 9DFE 3 LD A, B ; Get comm channel handle
1098 0404 9201 2 IFEQ A, #1 ; Check handle to see if valid UART
1099 0406 02 3 JP osUARTSendCh1 ; UART on channel 1
1100 0407 2438 3 JP osUARTSendError ; Not a valid UART
1101 0409 osUARTSendCh1:
1102 0409 9DFF 3 LD A, S ; Get the current segment register
1103 040B DF01 3 LD S, #1 ; Switch to OS segment
1104 040D 00 3 R X A, aStore+0 ; Save for later on OS page
1105 040F BD477 4 R IFBIT fTxBusy, bCOM1Status ; Is transmitter busy?
1106 0412 1D 3 JP osUARTSendBusy ; Yes, bail out
1107 0413 9D42 3 R LD A, pCOM1TxWrite ; Get the write pointer to COM1
1108 0415 9CFE 3 X A, B ; Use 'B' to write the data
1109 0417 BC 3 POP A ; Get the byte data off the stack
1110 041A 92 2 X A, [B+] ; Write into buffer
1111 041B 9DFE 3 LD A, B ; Get pointer back
1112 041C BD472 4 R LD A, #bCOM1TxBuffer+cCOM1TxBufLen; Past end of buffer?
1113 041D 9832 4 R LD A, #bCOM1TxBuffer ; Reload at start of buffer
1114 041F BD438 4 R IFEQ A, pCOM1TxRead ; Have we overrun the FIFO?
1115 0422 BD447F 4 R SBIT fTxBusy, bCOM1Status ; Yes, use later for test.
1116 0425 9C42 3 R X A, pCOM1TxWrite ; Save the new COM1 write pointer
1117 0427 BD8BC7 4 SBIT ETI, ENUI ; Enable transmitter interrupt (start transmitter)
1118 0429 9D00 3 R LD A, aStore+0 ; Get old segment pointer back
1119 042C 9DFF 3 X A, S ; Return old segment
1120 042E 64 1 CLR A ; Return with zero (success)
1121 042F 0B 3 JP osUARTSendExit ; Done, return
1122 0430 osUARTSendExit:
1123 0430 8C 3 POP A ; Adjust Stack
1124 0433 9D00 3 R LD A, aStore+0 ; Get old segment pointer back
1136 0433 9CFF 3 X A, S ; Return old segment
1137 0435 9FE8 2 LD A, #X'FE' ; Return non-zero value (Busy)
1138 0437 03 3 JP osUARTSendExit ; Done, return
1139 0438 08 3 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 08E 5 osUARTSendExit:
1143 043B 8E 5 RET ; Return

1144 ;*********************************************************************
1145 ; Routine: osStart
1146 ; Use: This routine will start the MTK Operating System (OS)
1147 ; Entry: (None required)
1148 ; Returns: Accumulator is zero if successful, non-zero if failed.
1149 ;*********************************************************************
1150 043C 043C DF01 3 LD S, #1 ; Switch to OS RAM page
1151 043E 043E 9D0A 3 R LD A, bTaskNew ; Check if any tasked loaded.
1152 0440 0440 9200 2 IFEQ A, #0 ; Anything to run?
1153 0442 0442 1D 3 JP osStartError ; No, Return with error
1154 ;
1155 0444 0444 9C09 3 R X A, bTask ; Save in currently running task
1156 0446 0446 30B7 5 JSR osGetNextTask ; Get the next task number
1157 044D 044D 9C02 2 IFEQ A, #0 ; Anything to run next?
1158 044F 044F 16 3 JP osStartError ; No, Return with error
1159 0450 0450 9C0B 3 R X A, bTaskNext ; Save in next task variable
1160 ;
1161 0452 0452 9BFE 2 LD A, #LOW(osMain) ; Setup return Address
1162 0454 0454 67 3 PUSH A ; First LSB
1163 0455 0455 9B04 2 LD A, #HIGH(osMain) ;
1164 0457 0457 67 3 PUSH A ; Next LSB
1165 ;
1166 0458 0458 9D09 3 R LD A, bTask ; Get first task that’s enabled
1167 045A 045A 8B 1 DEC A ; Yes, get the address
1168 045B 045B 30A4 5 JSR osGetTaskAddress ; Use to get the next task
1169 045D 045D AA 2 LD A, [B+] ; Move past flags, Point to LSB
1170 045E 045E AA 2 LD A, [B+] ; Get LSB, Point to MSB
1171 045F 045F 67 3 PUSH A ; Put LSB on stack first (FILO)
1172 0460 0460 AE 1 LD A, [B] ; Get MSB, Point at LSB
1173 0461 0461 67 3 PUSH A ; Put on stack

1174 046B 046B 046B BDEF78 4 SBIT GIE, P5W ;
1175 0466 0466 02 3 JP osStartEnd ; Go begin
1176 0466 0466 98FF 2 LD A, #0xFF ; Make non-zero
1177 0468 0468 DF00 3 LD S, #0 ; Switch back to user RAM
1178 046A 046A 8E 5 RET ; Start first task (Cool!)

1179 ;*********************************************************************
1180 ; Routine: osMain
1181 ; Use: This routine is the main loop of the kernel. All dispatching
1182 ; is done from here. Any error handling is also done here.
1183 ; Entry: (None required)
1184 ; Returns: (Never)
1185 ;*********************************************************************
1186 046B 046B 046B 046B 046B 046B 046B 046B
; *********** DO ERROR CHECKING HERE ***********
046B DF01 3        LD   S, #1 ; Switch to OS data segment
046D 9D0B 3        R    LD   A, bTaskNext ; Get the pointer to the next task
046F 9200 2        IFPEQ A, #0 ; Something very bad has happened!
0471 2495 > 3      JP    osMainExit ; Let the user decide how to fix it.
0473 980B 2        LD   A, #LOW(osMain) ; Put return address on stack
0475 67 3          PUSH  A ; First LSB
0477 9804 2        LD   A, #HIGH(osMain) ;
0479 67 3          PUSH  A ; Next MSB
047B 9C0B 3        R    X    A, bTaskNext ; Update current task
047D 30B7 5        JSR   osGetNextTask ; Get the next one
047F 9C0B 3        R    X    A, bTaskNext
0481 9D09 3        R    LD   A, bTask ; Start next task thread
0483 AA 2          LD    A, [B+] ; Adjust task number for table look-up
0485 AA 2          LD    A, [B+] ; Load 'A' with Flags, point to LSB
0487 67 3          PUSH  A ; Save LSB
0489 6A 2          LD    A, [B]  ; Load 'A' with MSB
048B 67 3          PUSH  A ; Save MSB
048D 9C0B 3        R    X    A, bTaskNext ; Save it
048F 8C 3          POP   A ; Adjust stack
0491 8C 3          POP   A ;
0493 246B > 3      JP    osMain ; Go test it again
0495 AA 2          LD    A, bTask ; osMainSwitch:
0497 AA 2          LD    A, [B+] ; Load 'A' with Flags, point to LSB
0499 AA 2          LD    A, [B+] ; Load 'A' with LSB, point to MSB
049B 67 3          PUSH  A ; Save LSB
049D 6A 2          LD    A, [B]  ; Load 'A' with MSB
049F 67 3          PUSH  A ; Save MSB
04A1 9D09 3        R    LD   A, bTaskNext ; osMainExit:
04A3 DF00 3        LD    S, #0 ; Switch to user base RAM data segment
04A5 8E 5          RET   ;
04A7 9C0B 3        R    X    A, bTaskNext
04A9 8C 3          POP   A ;
04AB 8C 3          POP   A ;
04AD 246B > 3      JP    osMain ; .ENDSECT ; Section ends here
04AF 9C0B 3        R    X    A, bTaskNext
04B1 9D09 3        R    LD   A, bTask ; .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
-4 982
BUSY            . . 0002 Abs Null
-4
DCOM1Status     . . 0020 Rel Byte SEG
-138 567 568 1040 1041 1083 1084
DCOM1Status     . . 0044 Rel Byte SEG
-144 551 1116 1126
DCOM1Status     . . 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
-112 248 252 259 274 301 329 385 899 1156
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bTaskTime</td>
<td>000F Rel Byte SEG</td>
<td></td>
</tr>
<tr>
<td>bUARTSettings</td>
<td>001D Rel Byte SEG</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0006 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CHL1</td>
<td>0004 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CHLO</td>
<td>0003 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CKO</td>
<td>0007 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CKX</td>
<td>0001 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP10E</td>
<td>0003 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP1EN</td>
<td>0001 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP1INN</td>
<td>0001 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP1INF</td>
<td>0002 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP1OUT</td>
<td>0003 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP1RD</td>
<td>0002 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP20E</td>
<td>0006 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP2EN</td>
<td>0004 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP2INN</td>
<td>0004 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP2INF</td>
<td>0005 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP2OUT</td>
<td>0006 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMP2RD</td>
<td>0005 Abs Null</td>
<td></td>
</tr>
<tr>
<td>CMPSL</td>
<td>0087 Abs Byte</td>
<td></td>
</tr>
<tr>
<td>CNTRL</td>
<td>00DE Abs Byte</td>
<td></td>
</tr>
<tr>
<td>COP820</td>
<td>0002 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP820CJ</td>
<td>0005 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP840</td>
<td>0003 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP840CJ</td>
<td>0006 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP8620</td>
<td>0007 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP8640</td>
<td>0008 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP8720</td>
<td>0009 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP8780</td>
<td>000A Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP880</td>
<td>0004 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP888BC</td>
<td>0018 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP888CF</td>
<td>0014 Abs Null</td>
<td></td>
</tr>
<tr>
<td>COP888CG</td>
<td>0015 Abs Null</td>
<td></td>
</tr>
</tbody>
</table>

www.national.com
-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
ERR ........ 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
-121 776 773 872
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
MSKEL .... 0003 Abs Null
-4
mNone ...... 0000 Abs Null
-90
mTaskFailure ... 000F Abs Null
-93 479

www.national.com 32
<table>
<thead>
<tr>
<th>Function</th>
<th>Address</th>
<th>Description</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTimer</td>
<td>0001</td>
<td>Abs Null</td>
<td></td>
</tr>
<tr>
<td>mUART</td>
<td>0002</td>
<td>Abs Null</td>
<td></td>
</tr>
<tr>
<td>mUser</td>
<td>0010</td>
<td>Abs Null</td>
<td></td>
</tr>
<tr>
<td>osAddTask</td>
<td>0048</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osAddTaskEnd</td>
<td>0071</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osAddTaskError</td>
<td>0070</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osBaudTable</td>
<td>0376</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osGetFirstTask</td>
<td>00B3</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osGetNextTask</td>
<td>00B7</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osGetNextTaskDone</td>
<td>00CE</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osGetTaskAddress</td>
<td>00A4</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osKillTimer</td>
<td>02FE</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osKillTimer1</td>
<td>030C</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osKillTimer2</td>
<td>0313</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osKillTimer3</td>
<td>031A</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osKillTimerExit</td>
<td>0321</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osKillTimerOK</td>
<td>0320</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osMain</td>
<td>046B</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osMainExit</td>
<td>0495</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osMainSwitch</td>
<td>0490</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osPOP</td>
<td>0239</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osPOPError</td>
<td>0256</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osPOPExit</td>
<td>0258</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osPUSH</td>
<td>0219</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osSMTT_Write</td>
<td>0219</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osSMTT_Write</td>
<td>0276</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osSetCommChannel</td>
<td>0339</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osSetMaxTaskTime</td>
<td>025E</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osSetTimer</td>
<td>0279</td>
<td>Abs Null ROM</td>
<td>Pub</td>
</tr>
<tr>
<td>osSetTimer1</td>
<td>0296</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>osSetTimer2</td>
<td>029E</td>
<td>Abs Null ROM</td>
<td></td>
</tr>
<tr>
<td>772</td>
<td>-777</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetTimer3</td>
<td>02A6 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>779</td>
<td>-784</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetTimerAddress</td>
<td>02C1 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>797</td>
<td>799</td>
<td>-806</td>
<td></td>
</tr>
<tr>
<td>osSetTimerExit</td>
<td>02FB Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>786</td>
<td>-848</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetTimerPeriod</td>
<td>02C7 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>805</td>
<td>-811</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetTimerStart</td>
<td>02AE Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>776</td>
<td>783</td>
<td>-791</td>
<td></td>
</tr>
<tr>
<td>osSetUART2</td>
<td>0396 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>902</td>
<td>-1021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetUARTExit</td>
<td>03D6 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1047</td>
<td>-1052</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSetUARTFail</td>
<td>03D1 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1103</td>
<td>-1048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSignal</td>
<td>0324 Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>-897</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSignalEnd</td>
<td>0336 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>911</td>
<td>-914</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osSignalError</td>
<td>0334 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>906</td>
<td>-912</td>
<td></td>
</tr>
<tr>
<td>osSignalOK</td>
<td>0330 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>905</td>
<td>-908</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStart</td>
<td>043C Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>-1154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStartEnd</td>
<td>0468 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1184</td>
<td>-1187</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStartError</td>
<td>0466 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1588</td>
<td>1162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStartTask</td>
<td>0078 Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>-295</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStartTaskEnd</td>
<td>008C Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>308</td>
<td>-311</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStartTaskError</td>
<td>008A Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>-309</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStopTask</td>
<td>0093 Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>-327</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStopTaskEnd</td>
<td>00A1 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>336</td>
<td>-339</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osStopTaskError</td>
<td>009F Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>-337</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTGetChar</td>
<td>03D7 Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>-1066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTRxCh1</td>
<td>03E3 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1073</td>
<td>-1075</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTRxExit</td>
<td>0400 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1089</td>
<td>-1094</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTRxFail</td>
<td>03F6 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1074</td>
<td>1078</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTSend</td>
<td>0401 Abs Null ROM Pub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>-1106</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTSendBusy</td>
<td>0430 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1112</td>
<td>-1133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTSendCh1</td>
<td>0409 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1115</td>
<td>-1112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTSendError</td>
<td>0438 Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>-1139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUARTSendExit</td>
<td>043B Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1132</td>
<td>1138</td>
<td></td>
<td></td>
</tr>
<tr>
<td>osUnassigned</td>
<td>00CF Abs Null ROM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>206</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>PE</td>
<td>0005 Abs Null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEN</td>
<td>0007 Abs Null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td>1034</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PORTC C...... 00D9 Abs Byte
-4
PORTC D...... 00D8 Abs Byte
-4
PORTC P...... 00DA Abs Byte
-4
PORTD ...... 00DC Abs Byte
-4
PORTG C...... 00D5 Abs Byte
-4
PORTG D...... 00D4 Abs Byte
-4
PORTG P...... 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
KFL ... ....... 0001 Abs Null
-4
KBIT9 .. ....... 0003 Abs Null
-4
KBUF ... ....... 0089 Abs Byte
-4 214 564
KCVG ... ....... 0000 Abs Null
-4
KDX ... ....... 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
716 720 750 849 862 885 898 915 962 963 1044 1050 1067 1068 1087 1092
1113 1114 1130 1136 1155 1188 1205 1236
S0 ...... ....... 0000 Abs Null
-4
S1 ...... ....... 0001 Abs Null
-4
S10 ... ....... 0006 Abs Null
-4
S10 ... ....... 00E9 Abs Byte
-4
SIOR ... ... ... 00E9 Abs Byte
<table>
<thead>
<tr>
<th>Service</th>
<th>Offset</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceExternal</td>
<td>01CF</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceMicrowire</td>
<td>01CA</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceSoftware</td>
<td>01C4</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT0</td>
<td>0112</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT0Exit</td>
<td>013C</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT1A</td>
<td>0140</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT1B</td>
<td>0154</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT2</td>
<td>0158</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT2A</td>
<td>016C</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT2B</td>
<td>0170</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT3</td>
<td>0174</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT3A</td>
<td>0180</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceTimerT3B</td>
<td>0184</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceUART_RX</td>
<td>01A4</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceUART_TX</td>
<td>0188</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceUnused</td>
<td>010A</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>ServiceWakeup</td>
<td>01D</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>Start</td>
<td>0000</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T0EN</td>
<td>0004</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T0PND</td>
<td>0005</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1A</td>
<td>0003</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1B</td>
<td>0002</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1C0</td>
<td>0004</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1C1</td>
<td>0005</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1C2</td>
<td>0006</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1C3</td>
<td>0007</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>T1ENB</td>
<td>0000</td>
<td>Abs</td>
<td>Null ROM</td>
</tr>
<tr>
<td>Code</td>
<td>Type</td>
<td>Value 1</td>
<td>Value 2</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>T1PND8</td>
<td></td>
<td>0001</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>504</td>
</tr>
<tr>
<td>T1RAH1</td>
<td></td>
<td>00ED</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T1RALO</td>
<td></td>
<td>00EC</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T1RBIH</td>
<td></td>
<td>00E7</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T1RLIO</td>
<td></td>
<td>00E6</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>833</td>
</tr>
<tr>
<td>T2A</td>
<td></td>
<td>0004</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2B</td>
<td></td>
<td>0005</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2C1</td>
<td></td>
<td>0005</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2C2</td>
<td></td>
<td>0006</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2C3</td>
<td></td>
<td>0007</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2CNTRL</td>
<td></td>
<td>00C6</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>220</td>
</tr>
<tr>
<td>T2CO</td>
<td></td>
<td>0004</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>844</td>
</tr>
<tr>
<td>T2ENA</td>
<td></td>
<td>0002</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2ENB</td>
<td></td>
<td>0000</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2PND8</td>
<td></td>
<td>0001</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>508</td>
</tr>
<tr>
<td>T2PND9</td>
<td></td>
<td>0002</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2RAH1</td>
<td></td>
<td>00C3</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2RBH1</td>
<td></td>
<td>00C5</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T2RLO</td>
<td></td>
<td>00C4</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3A</td>
<td></td>
<td>0006</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3B</td>
<td></td>
<td>0007</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3C1</td>
<td></td>
<td>0005</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3C2</td>
<td></td>
<td>0006</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3C3</td>
<td></td>
<td>0007</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3CNTRL</td>
<td></td>
<td>00B6</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>221</td>
</tr>
<tr>
<td>T3CO</td>
<td></td>
<td>0004</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>846</td>
</tr>
<tr>
<td>T3ENA</td>
<td></td>
<td>0002</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3ENB</td>
<td></td>
<td>0000</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3PND8</td>
<td></td>
<td>0003</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>527</td>
</tr>
<tr>
<td>T3PND9</td>
<td></td>
<td>0004</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td>542</td>
</tr>
<tr>
<td>T3RAH1</td>
<td></td>
<td>00B3</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3RALO</td>
<td></td>
<td>00B2</td>
<td>Abs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>T3RBH1</td>
<td></td>
<td>00B5</td>
<td>Abs</td>
</tr>
</tbody>
</table>
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.

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and they agree and acknowledge that they are solely responsible for all legal and regulatory requirements concerning such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and they agree and acknowledge that they are solely responsible for all legal and regulatory requirements concerning such use. Buyers must fully indemnify, defend, and hold TI and its representatives harmless from any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use. TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failures to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Communications and Telecom</td>
</tr>
<tr>
<td>Amplifiers</td>
<td>Computers and Peripherals</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Consumer Electronics</td>
</tr>
<tr>
<td>DLP® Products</td>
<td>Energy and Lighting</td>
</tr>
<tr>
<td>DSP</td>
<td>Industrial</td>
</tr>
<tr>
<td>Clocks and Timers</td>
<td>Medical</td>
</tr>
<tr>
<td>Interface</td>
<td>Security</td>
</tr>
<tr>
<td>Logic</td>
<td>Space, Avionics and Defense</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Transportation and Automotive</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Video and Imaging</td>
</tr>
<tr>
<td>RFID</td>
<td></td>
</tr>
<tr>
<td>OMAP Mobile Processors</td>
<td></td>
</tr>
<tr>
<td>Wireless Connectivity</td>
<td></td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2011, Texas Instruments Incorporated