AN1262

Simple Real-Time Kernels for M68HC05 Microcontrollers

**By Joanne Field
CSIC Applications
Motorola Ltd.
East Kilbride, Scotland**

INTRODUCTION

This application note demonstrates the operation of two different types of simple real-time kernels for the M68HC05 MCUs, namely, a priority-based kernel and a time-based kernel. Assembly source code is provided for each.

WHY USE A REAL-TIME KERNEL?

A kernel is similar to a simple operating system in that it offers very fast software development and gives flexibility that allows new modules to be added without interfering with those already in place. A real-time kernel is easy to debug and encourages the user to develop software in an organized fashion. Two simple real-time kernels are presented in this application note: a priority-based kernel and a time-based kernel.

The priority-based kernel provides a means of executing a number of user-defined tasks, where the order of execution of each task is determined by the priority level assigned by the user. This kernel is used for tasks that vary in their execution times or where interrupts may be common or lengthy.

The time-based kernel executes user-defined tasks at specific, regular time intervals. These tasks are written so that they run immediately and do not require code to determine the timing of their execution. Rather, the user determines the rate of execution. This kernel is ideal for many predicted duration routines with few or short duration interrupts.

Both these examples aim to demonstrate the ease with which software modules can be integrated into a kernel and executed to support different applications.

PRIORITY-BASED KERNEL

Specific features of the priority-based kernel are:

1. This implementation supports three priority levels, although more levels are possible. These will be referred to as Priority 1, 2, and 3, with Priority 1 having the highest ranking.
2. Each priority level is capable of controlling the execution of eight tasks via a task request register.
3. Task addresses are stored, by the user, in a task table located at the end of the program.
4. One bit in each of the priorities' task request registers corresponds to one task in the task table.
5. Within each of the priority levels, bit 0 of the task request register is assigned the highest priority and bit 7 is assigned the lowest priority.
6. A task can change priorities by being entered into more than one position in the task table, which means setting a different bit in one of the request registers.
7. When work is to “start” on a priority level, a copy of the task request register is made. The copy is referred to as the “shadow register.” The kernel operates on this copy. The original is then cleared, thus enabling it to be updated with new tasks that require execution.
8. Note that "start" means that the previous operation, carried out by the kernel, will have caused the shadow register to be declared empty, so that all the tasks in that priority at that time will have been completed and their corresponding bits cleared.
9. The Priority 1 shadow register is always updated/checked first.
10. The Priority 2 shadow register is updated/checked only after all the Priority 1 tasks set to execute at that time have been completed, so that the Priority 1 shadow register is empty. Only one Priority 2 task is executed at a time, before starting again on the Priority 1 task request register.
11. The Priority 3 shadow register is updated/checked only after all the Priority 1 tasks and Priority 2 tasks set to execute at that time have been completed, so that the Priority 1 and 2 shadow registers are empty. Only one Priority 3 task is executed at a time, before starting again on the Priority 1 task request register followed by Priority 2.
12. A task that is running can order another task to run by setting the appropriate bit in one of the task request registers.
13. The kernel is capable of supporting interrupts, such as EXT, SCI, TIMER, etc.
14. The kernel supports local and global variables, but the user must manage these carefully, especially when information is being passed between procedures.

NOTE

A task that is running can stop another task which is scheduled to run by clearing the appropriate bit in the correct task request register. However, this may not be advisable and is not supported in this implementation.

SOFTWARE OPERATION

For a task to run, it must be assigned a position in the task table. Each position in the table corresponds to a bit in one of the task request registers. The user's program sets the bit. Execution time has no constraints and any number of tasks may be scheduled to run at any one time.

Here is a basic description of how the software operates. Refer to the flowchart shown in Figure 3.

1. When a priority level is to be operated on, a copy is made of the corresponding task request register. This copy is called the shadow register. The original is then cleared so that it can be updated when new tasks require execution.
2. The kernel checks for bits set in the shadow registers. Any set bits which require execution correspond to particular tasks in the task table.
3. Priority 1 is checked first, starting from bit 0.
4. After all these tasks have been checked and executed, one Priority 2 task is executed.
5. If there are no Priority 2 tasks at this time, a Priority 3 task is executed. If there are no Priority 3 tasks at this time, the kernel updates and then checks the Priority 1 shadow register.
6. Every time a task has been executed, the bit in the shadow register, which corresponds to the task, is cleared.
7. When any one of the shadow registers is declared totally empty, it is updated again by copying the corresponding original task request register. In this way, any new tasks that require execution will be scheduled for execution.
8. After either a Priority 2 task or a Priority 3 task has been executed, the kernel checks the updated Priority 1 shadow register. If there are any Priority 1 tasks to be executed, all of them will be executed before any further Priority 2 or Priority 3 tasks are executed.
9. The whole process is then repeated.

An example of the software operation and steps carried out are shown in Figure 1.

1. Copy the Priority 1 task request register to a shadow register and clear the original. Inspect the Priority 1 shadow register, starting from bit 0 — execute task A, then task C, then task G.
2. Copy the Priority 2 task request register to a shadow register and clear the original.
3. Inspect the Priority 2 shadow register — execute task L.
4. Inspect the updated Priority 1 shadow register — no tasks to execute. Inspect the Priority 2 shadow register — no tasks to execute.
5. Copy the Priority 3 task request register to a shadow register and clear the original. Inspect the Priority 3 shadow register — execute task U.
6. Inspect the updated Priority 1 shadow register — no tasks to execute.
7. Inspect the updated Priority 2 shadow register — no tasks to execute.
8. Inspect the Priority 3 shadow register — execute task X.
9. Inspect the updated Priority 1 task request register.

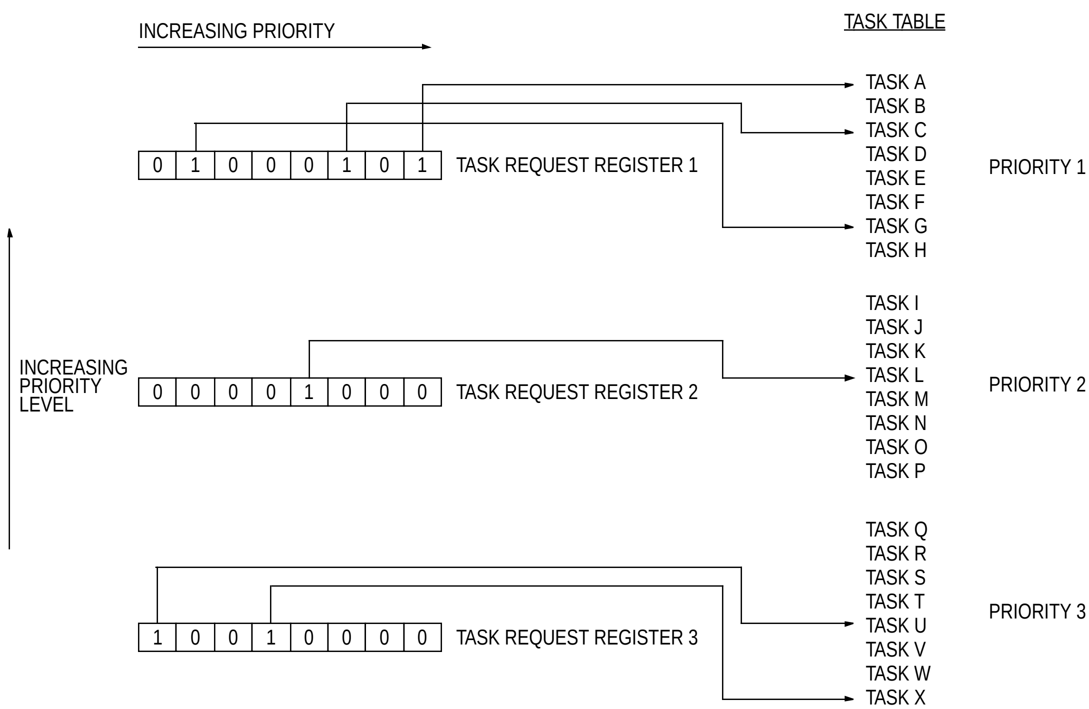


Figure 1. Software Operation Example

Figure 2 shows a change of selected tasks in Priority 1. This involves updating the corresponding bits in the task request register each time a task requires execution.

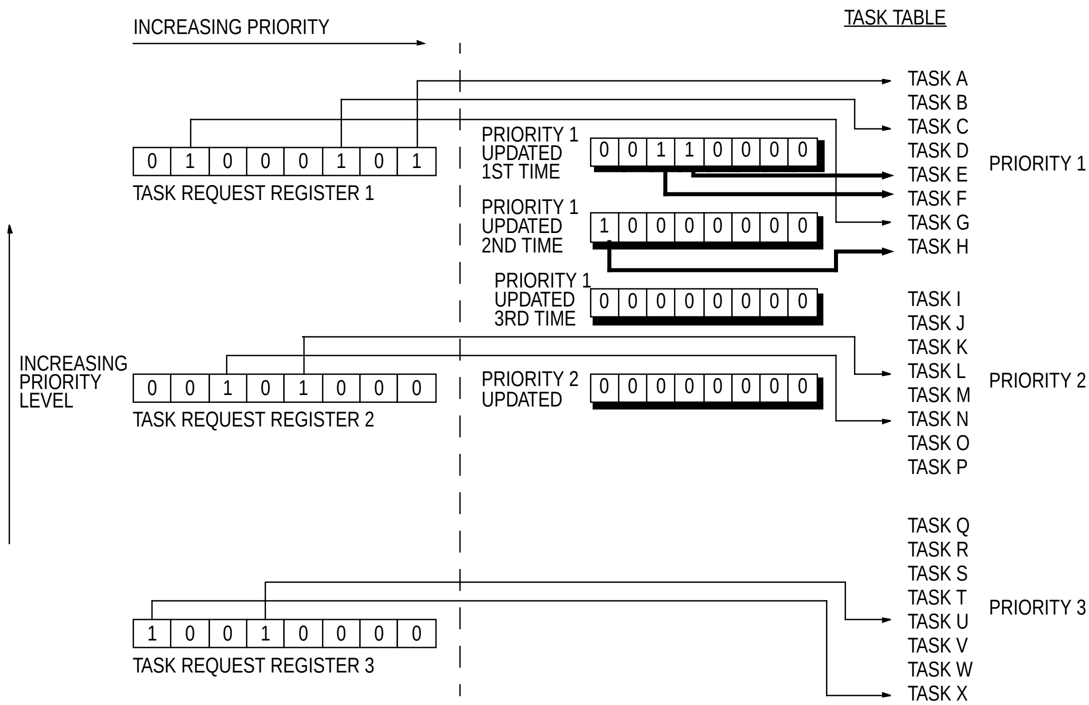


Figure 2. Updating Task Request Registers Example

The priority-based kernel performs these operations:

1. Copy the Priority 1 task request register to a shadow register and clear the original. Inspect the Priority 1 shadow register, starting from bit 0 — execute task A, then task C, then task G.
2. Copy the Priority 2 task request register to a shadow register and clear the original. Inspect the Priority 2 shadow register — execute task L.
3. Inspect the updated Priority 1 task request register (updated 1st time). For example, copy the Priority 1 task request register to a shadow register and clear the original. Inspect the Priority 1 shadow register — execute task E, then task F.
4. Inspect the Priority 2 shadow register again — execute task N.
5. Inspect the updated Priority 1 task request register (updated 2nd time). For example, copy the Priority 1 task request register to a shadow register and clear the original. Inspect the Priority 1 shadow register — execute task H.
6. Inspect the updated Priority 2 task request register (Priority 2 updated). For example, copy the Priority 2 task request register to a shadow register and clear the original. Inspect the Priority 2 shadow register — no tasks to execute.
7. Copy the Priority 3 task request register to a shadow register. Inspect the Priority 3 shadow register — execute task U.
8. Inspect the updated Priority 1 task request register (updated 3rd time). For example, copy the Priority 1 task request register to a shadow register and clear the original. Inspect the Priority 1 shadow register — no tasks left to execute.
9. Inspect the updated Priority 2 task request register — no tasks left to execute.
10. Inspect the Priority 3 shadow register again — execute task X.
11. Inspect the updated Priority 1 task request register.

IMPLEMENTATION

Flowchart 1(Figure 3) explains how the software is designed to operate.

Listing 1 shows how the assembler code is used to implement the priority-based kernel. In this case, the MC68HC05C9 has been chosen as an example. However, the software is designed to support any M68HC(7)05 device with minimal changes to memory organization.

To integrate code into the kernel, the user must enter the address of the routine into the task table. Each 16-bit entry in the table points to a task. This implementation has 26 entries, but there can be as many as the user requires. When a task is to be executed, a corresponding entry in the task table is used as the destination address of a subroutine call. This means that each task must finish with an RTS command.

Unused entries in the task table must point to an RTS command for safety reasons.

The procedure WRITERAM, in Listing 1, controls which task is executed. The task table starts at an arbitrary value of $400 in the MC68HC05C9 user ROM.

The user controls the program flow using flags. These flags, internal to the task, control which subtask is carried out each time the task is executed.

Task D of Listing 1 shows an example of how code is integrated into the kernel.

The listing also includes an SCI interrupt service routine to demonstrate how the scheduler handles interrupts. This routine is an example of communication between the MCU's SCI and a dumb terminal. The MCU receives an ASCII character, which is sent by the dumb terminal through an RS232 cable. The MCU then translates the 8-bit binary character, representing the ASCII character, into two ASCII characters. These characters, which represent the original hexadecimal equivalent of the received character, then are transmitted back to the terminal.

The routine also shows how other tasks are scheduled to execute.

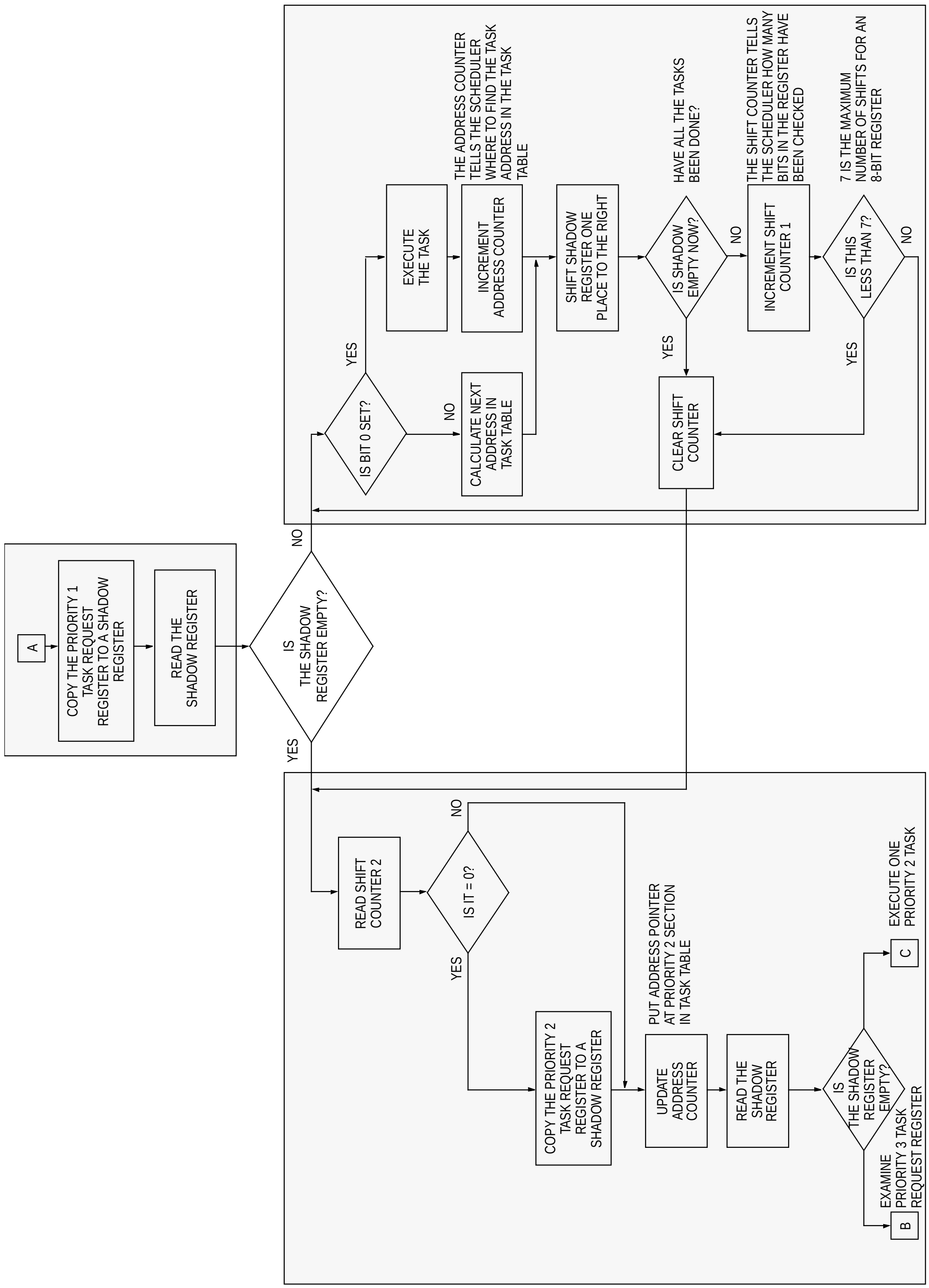


Figure 3. Flowchart 1 (Sheet 1 of 2)

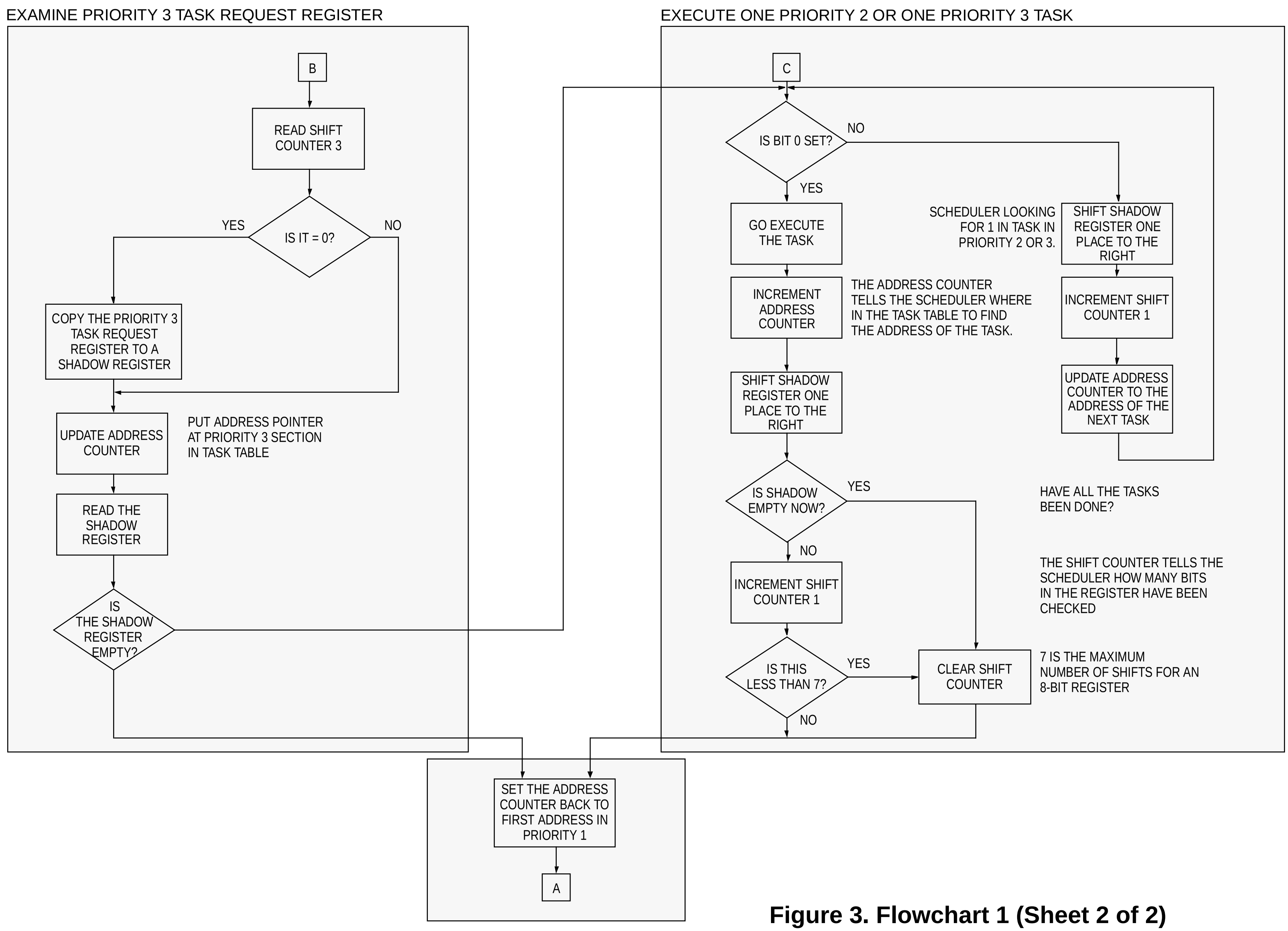


Figure 3. Flowchart 1 (Sheet 2 of 2)

TIME-BASED KERNEL

Specific features of the time-based kernel are:

1. This kernel uses the MCU system clock and different counters to allocate time slots for each task to be executed.
2. The timing of execution of these tasks is controlled by the generation of timer interrupts inside the MCU. These interrupts are generated in different ways, depending on the timer that is used.
3. Two kinds of timers are supported in this implementation: the programmable timer and the core timer. The timer used depends on which MCU is being used in the application. Some HC05s have only one of the two timers. The MC68HC05L4, used in this example, has both timers, so the timer required to control the kernel has to be selected by the user before assembling the program.
4. Both timers have a continually incrementing counter which acts as a clock for the kernel. The programmable timer has a free-running counter and the core timer has a timer counter register.
5. When a timer interrupt occurs, a flag is generated by the timer. The programmable timer generates an output compare flag and the core timer generates a core timer overflow flag. A service routine, pointed to by the interrupt vector, is then executed. The flags are tested within the interrupt routine to verify the interrupt source, since the interrupt vector is shared.
6. User-generated interrupt service routines should be kept as short as possible to ensure that maximum time is allowed for each task to execute. Strict testing must be made for the worst case timing of each.
7. A time slice counter determines the minimum time between tasks by counting the timer interrupts.
8. The time slice counter is available as a timer for tasks to use, for example, for delay or debounce routines.
9. A task counter determines exactly which task is to be executed. Each time the time slice counter decides that a task is to be performed, the task counter increments. The kernel then tests which bit in the task counter is clear, and, depending on which bit is clear, a corresponding task is executed.
10. The number of tasks has no limit. The user can have the number required since this is only dependent on the number of bits in the task counter.
11. Tasks that take longer than one time slot to execute can be split into subtasks. For example, this is useful in an EEPROM programming routine where a time delay is required between the procedures. This routine could be divided into:
    - byte erase
    - byte program
    - program verify
12. Flags, internal to the task, are used to control which subtask is to be carried out each time the task is executed.
13. The kernel supports local and global variables, but the user must manage these carefully, especially when information is being passed between procedures.

SOFTWARE OPERATION

Timer Interrupt Generation

In the case of the programmable timer, the output compare interrupt is generated when the free-running counter counts up to equal a pre-determined value of the output compare. This pre-determined value is called the output compare period and is declared at the start of the program, so that the value in the output compare can be updated easily. Setting the output compare period in this way allows for easy adaptation to suit the timing of the application.

When using the core timer, the interrupt is generated each time the core timer counter register rolls over from $FF to $00. Thus, the core timer overflow interrupt is generated every 512 microseconds (when using a 4-MHz clock). Unlike the programmable timer, its value cannot be changed.

Task Execution

A time slice period is set at a pre-defined value at the start of the program, again to allow easy adaptation of the routine. The time slice counter will increment each time an interrupt is generated until it reaches the value of the time slice period. When this occurs, the task counter is incremented and, therefore, a task is executed. At this point, the time slice counter is reset, ready to count the next time slice period.

Each of the tasks should take, or be split into subtasks that take, less than one time slice period to execute. The kernel provides a task flag for different task rates, so that tasks should be running at binary power multiples of the time slice period. The fastest task runs at a period of twice the time slice period, the next fastest runs at a period of four times the time slice period, the next task eight times the time slice period and so on. These tasks are referred to as tasks A, B, C, etc. Thus, task H will run at a period of 256 times the time slice period.

Each bit of the task counter corresponds to a task. Each time the task counter is incremented, the task counter byte is tested for the presence of a zero, starting with the least significant bit. When a zero bit is found, the routine aborts the check and the corresponding task is executed. Note that no task is executed when the task counter is all ones ($FF if one byte). This signifies that a background task or idle loop will be the only activity run for this task period.

There can be as many tasks as there are number of bits in the task counter, and this counter can be as many bytes as the application requires.

It is possible to have several small tasks, rather then one big task, executing inside one time slot. When entering the time slot, the kernel detects which task to execute by inspecting flags controlled by the user routines.

It is also possible to only use some of the time slots available. The unused slots could allow more time for background tasks.

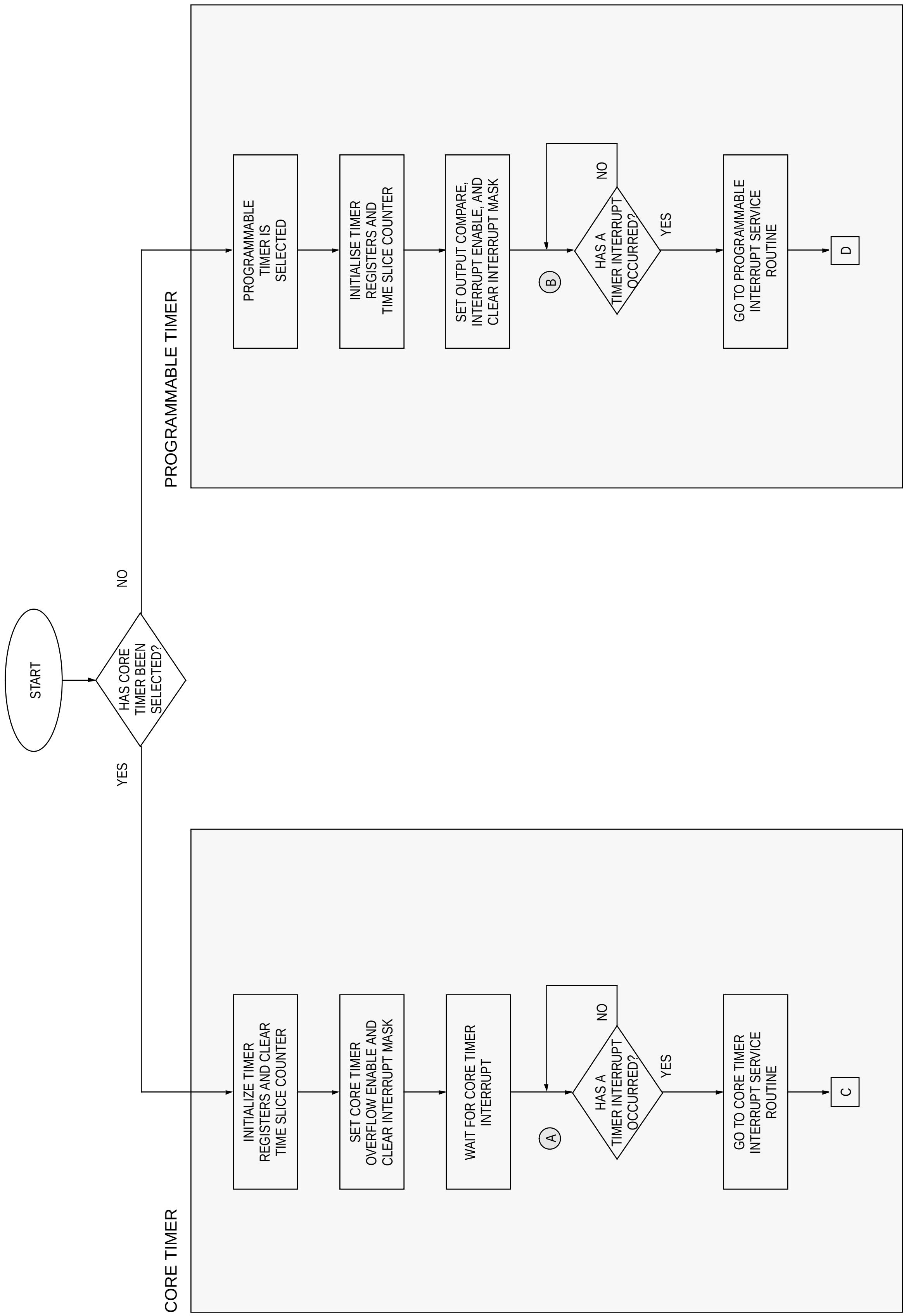


Figure 4. Flowchart 2 (Sheet 1 of 3)

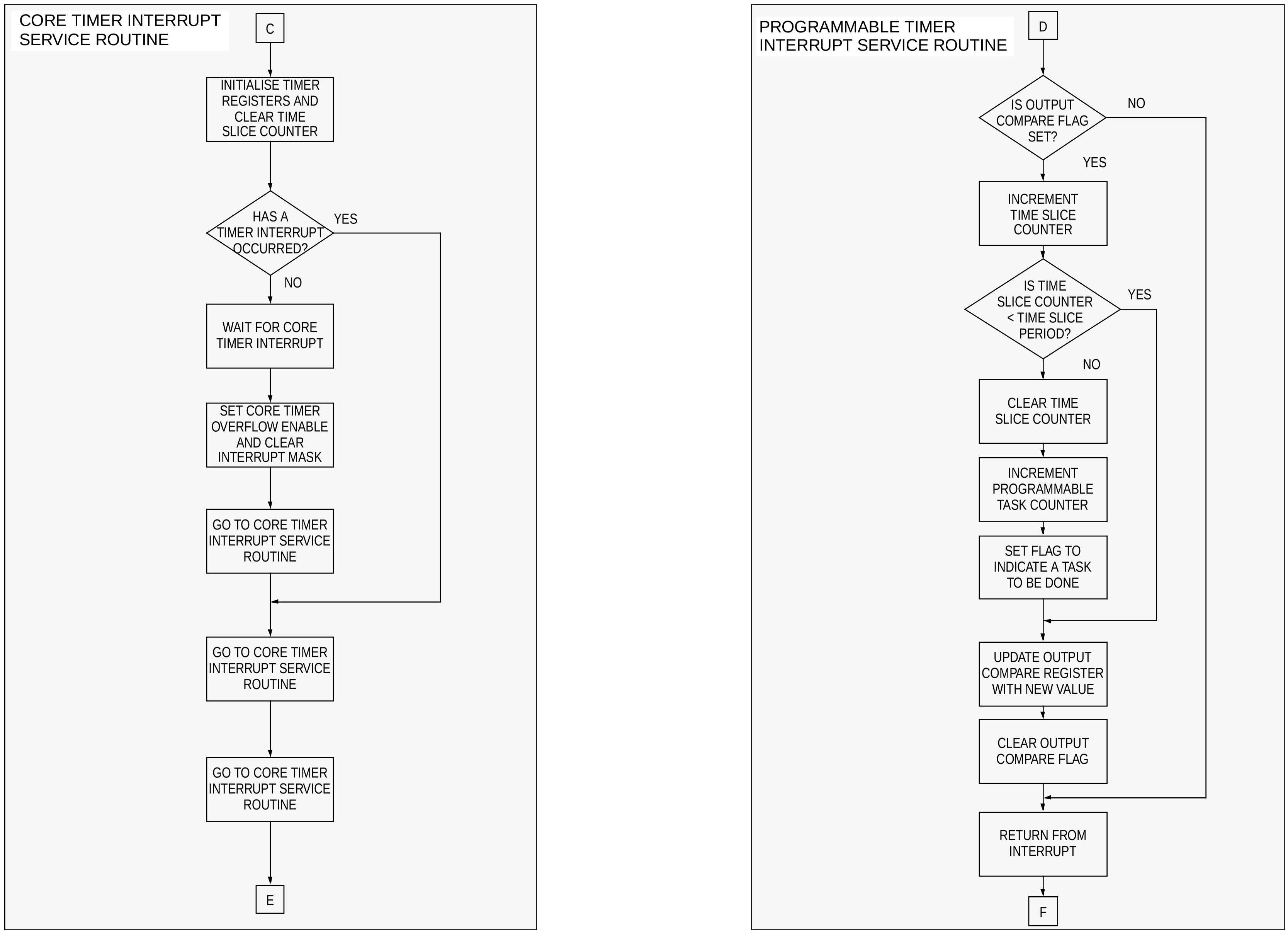
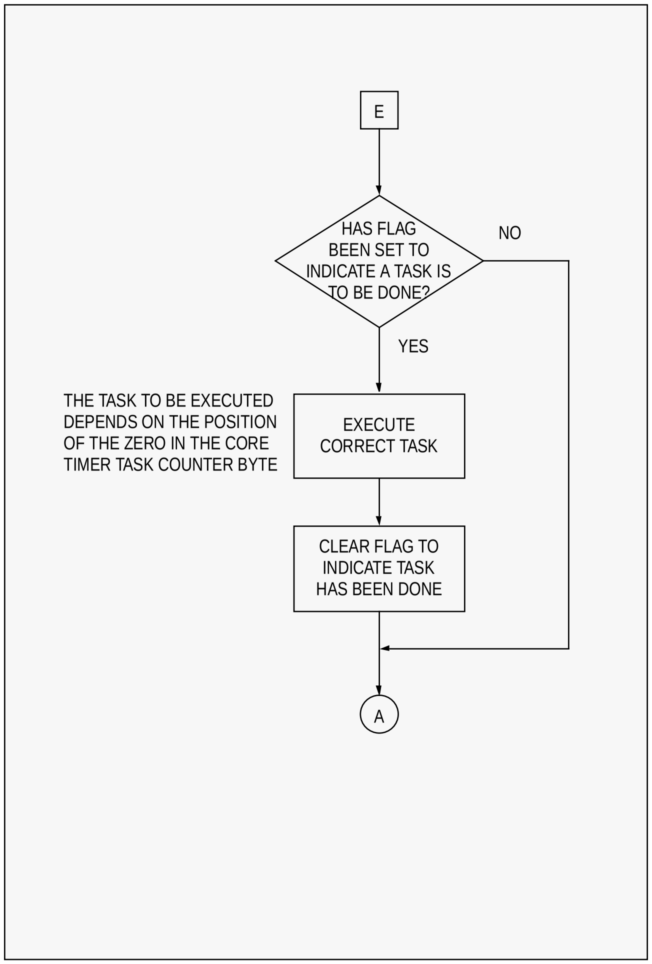


Figure 4. Flowchart 2 (Sheet 2 of 3)

CORE TIMER



PROGRAMMABLE TIMER

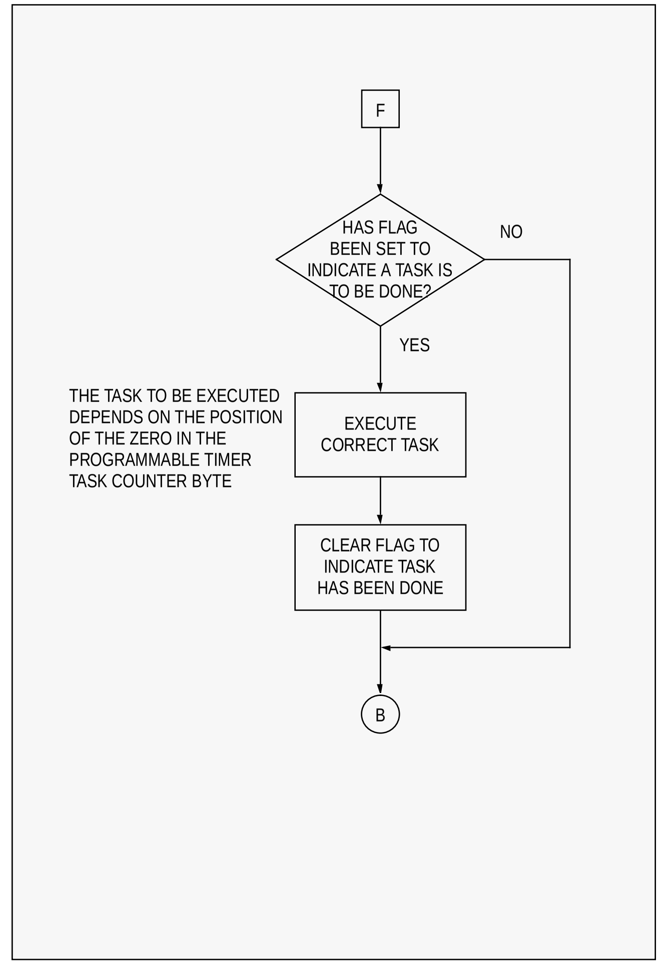


Figure 4. Flowchart 2 (Sheet 3 of 3)

Example 1 assumes the programmable timer is being used and a 5 ms time slice period is required, the most frequent task executing every 10 ms. The 5 ms time slice period is obtained by multiplying the internal system clock (2 µs) by an output compare period set at 250, multiplied by a time slice period set at 10. This gives an interrupt every 500 µs and a task executed every 5 ms (500 µs x 10).

The sequence of the task execution using the programmable timer in this way is shown in Figure 5.

The execution repetition period of each task = 5 x 2n, where n = position number of the letter in the alphabet, for example, task B's execution repetition period = 5 x 2(2) = 20 ms.

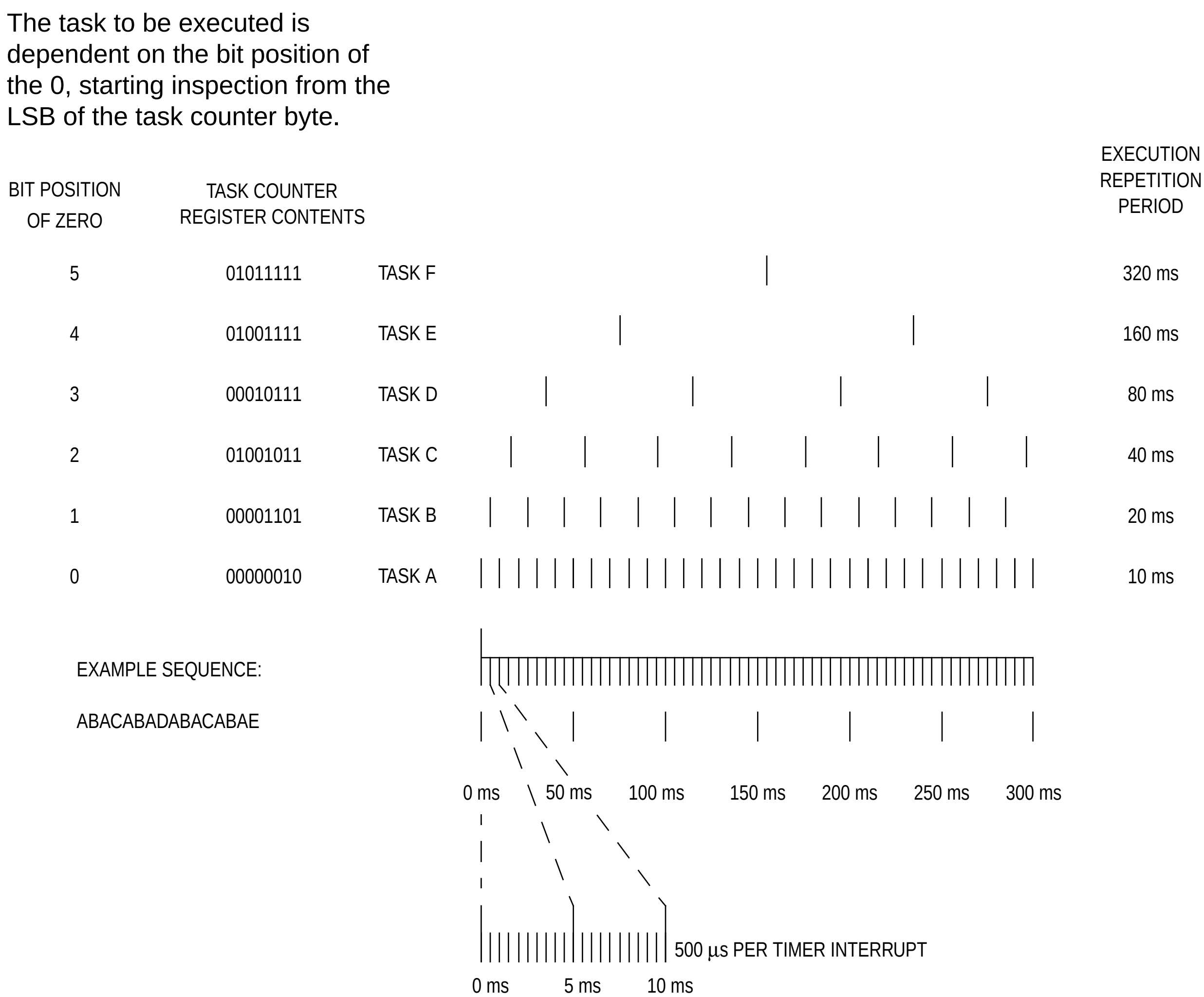


Figure 5. Example 1 — Sequence of Task Execution for Programmable Timer

Example 2 assumes the core timer is being used and that a 5.1 ms time slice period is required, the most frequent task executing every 10.2 ms. The 5.1 ms time slice period is obtained by multiplying the internal system clock (2 µs), multiplied by 255, which is the number the core timer counter register counts up to before rolling over to $00, multiplied by a time slice period of 10. This gives an interrupt every 510 µs and a task executed every 5.1 ms (510 µs x 10).

The sequence of task execution using the core timer in this way is shown in Figure 6.

The execution repetition period of each task = 5.1 x 2n, where n = position number of the letter in the alphabet, for example, task B's execution repetition period = 5.1 x 2(2) = 20.4 ms.

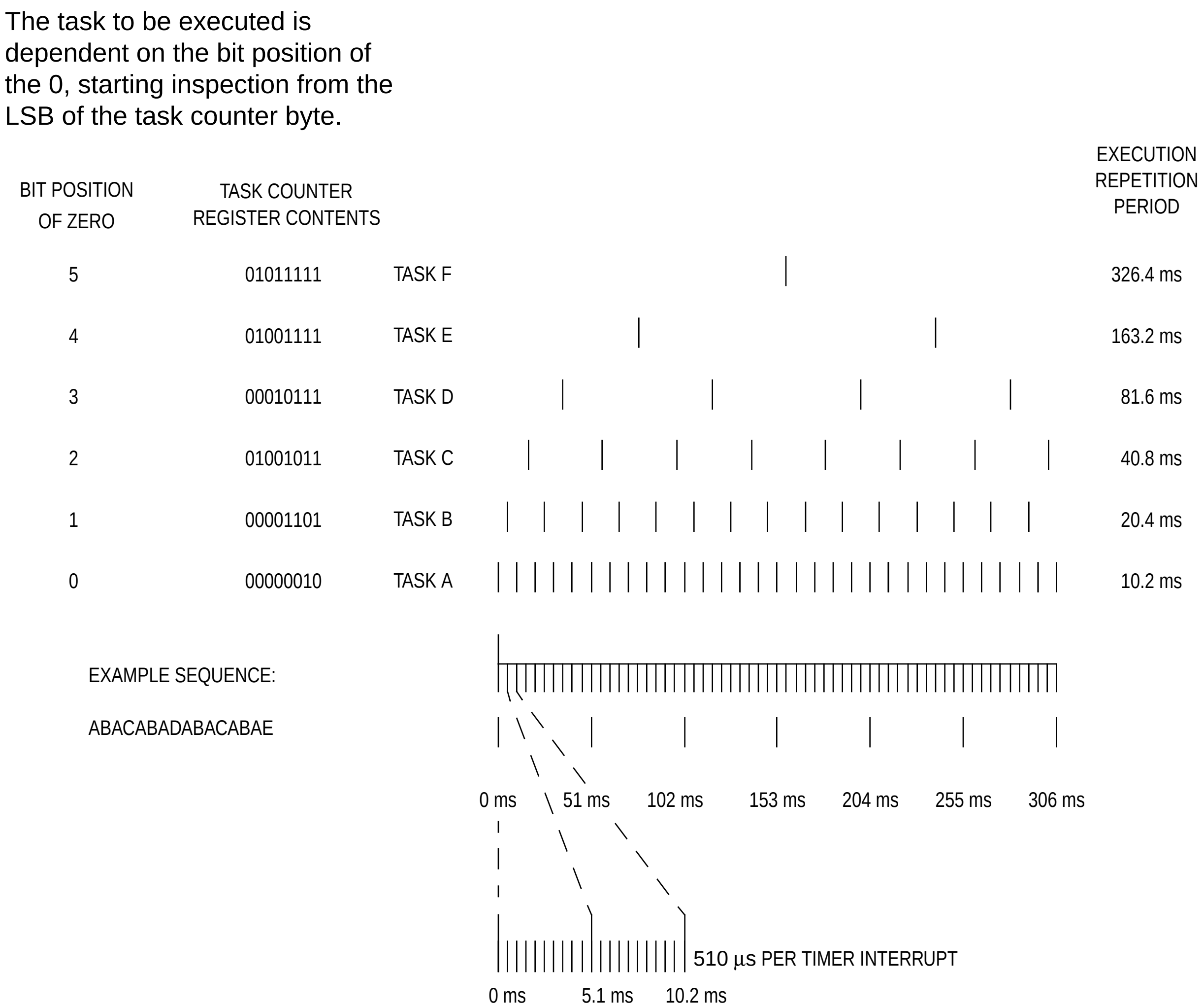


Figure 6. Example 2 — Sequence of Task Execution for Core Timer

IMPLEMENTATION

Flowchart 2 (Figure 4) explains how the software is designed to operate.

Listing 2 shows the assembly code used to implement the time-based kernel. The 68HC05L4 was chosen to demonstrate the use of both timers in the software.

Code is integrated into this kernel in modules. Each of these modules is entered like a subroutine and so must finish with the RTS command.

Note that the slots not filled with user tasks also must have an RTS.

This implementation has only eight time slots; however, this can be extended by making the task counter larger.

Listing 2 shows simple tasks in order to demonstrate where the user's tasks are placed. Each task toggles a different port pin on port B of the device.

A good example of the time-based kernel in operation is in the application note titled *Telephone Handset with DTMF using the MC68HC05F4*, Motorola document number AN488/D. In this example, the kernel has been used, along with flags on entry to each routine, to control the program flow.

Also note that, when developing software to integrate into the kernel, worst case timing analysis is required to ensure correct operation.

SUMMARY

In summary, the priority-based kernel offers a very simple way to execute software modules in an application, where the number of tasks may vary depending on the conditions resulting from a particular operation. Tasks are selected to execute merely by setting a bit in one of the task request registers, provided the user's software modules are positioned correctly in the task table.

The time-based kernel provides a means of executing a number of tasks at specific, regular time intervals. The execution of the task, once the kernel has entered the time slot automatically, is dependent on flags being set to control the software. This could be useful in an application where time of day events require recording.

Both kernels encourage group development and module reuse, which together have proven to offer a much more efficient way of developing software.

```
*****************************************************************************
*                                               COPYRIGHT (c) MOTOROLA 1994 *
*                          LISTING 1                                        *
*                          *********                                        *
* FILE NAME: PRIORITY.ASM                                                   *
*                                                                           *
* PURPOSE: The purpose of this software is to provide a means of executing  *
*          a number of user defined tasks, where the order of execution of  *
*          each task is determind by the level of priority that the task is *
*          assigned by the user.                                            *
*                                                                           *
* TARGET DEVICE: 68HC(7)05                                                  *
*                                                                           *
* MEMORY USAGE(bytes)  RAM: 22 BYTES     ROM: 640 BYTES                     *
*                                                                           *
* ASSEMBLER:  IASM05                      VERSION: 3.02                     *
*                                                                           *
* DESCRIPTION:  This Priority Scheduler uses 3 task request register        *
*               (for 3 different priority levels) to organise the user      *
*               defined tasks into different priorities. Each bit           *
*               in each of the 3 task request registers corresponds         *
*               to one task in a Task Table, located at the end of the      *
*               program. The user is simply required to enter a task into   *
*               the appropriate position in the task table and set the      *
*               corresponding bit in the correct task request register.     *
*               The prefix PS refers to PRIORITY SCHEDULER.                 *
*                                                                           *
* AUTHOR: Joanne Santangeli   LOCATION: EKB    LAST EDIT DATE:  9/DEC/94    *
*                                                                           *
* UPDATE HISTORY                                                            *
* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                     *
* ---      ------    ---------    ---------------------                     *
* 1.O      JS        9/12/94      INITIAL RELEASE                           *
*                                                                           *
*===========================================================================*
* Motorola reserves the right to make changes without further notice to any *
* product herein to improve reliability, function, or design. Motorola does *
* not assume any  liability arising  out  of the  application or use of any *
* product,  circuit, or software described herein;  neither  does it convey *
* any license under its patent rights  nor the  rights of others.  Motorola *
* products are not designed, intended,  or authorized for use as components *
* in  systems  intended  for  surgical  implant  into  the  body, or  other *
* applications intended to support life, or  for any  other application  in *
* which the failure of the Motorola product  could create a situation where *
* personal injury or death may occur. Should Buyer purchase or use Motorola *
* products for any such intended  or unauthorized  application, Buyer shall *
* indemnify and  hold  Motorola  and its officers, employees, subsidiaries, *
* affiliates,  and distributors harmless against all claims costs, damages, *
* and expenses, and reasonable  attorney  fees arising  out of, directly or *
* indirectly,  any claim of personal injury  or death  associated with such *
* unintended or unauthorized use, even if such claim alleges that  Motorola *
* was negligent regarding the  design  or manufacture of the part. Motorola *
* and the Motorola logo* are registered trademarks of Motorola Ltd.         *
*****************************************************************************
```

```
********************************
* MEMORY AND PORT DECLARATIONS *
********************************

ROM             EQU     $180            ;User ROM are for the 705C9
RAM             EQU     $50             ;RAM are for 705C9
VECTOR          EQU     $3FF4           ;Start of vector addresses
TABLE           EQU     $400            ;Start address of task table

PORTA           EQU     $00             ;Port A declaration
DDRA            EQU     $04             ;Port A Data Direction declaration
PORTB           EQU     $01             ;Port B declaration
DDRB            EQU     $05             ;Port B Data Direction Register

BRATE           EQU     $0D             ;Baud rate register
SCCR1           EQU     $0E             ;SCI control register 1
SCCR2           EQU     $0F             ;SCI control register 2
SCDAT           EQU     $11             ;SCI data register
SCSR            EQU     $10             ;SCI status register

********************************
* PRIORITY SCHEDULER CONSTANTS *
********************************

LSB             EQU     0               ;Bit 0 of task request registers
DO_TASK         EQU     1               ;Flag to say do Priority 1 task
TRY_PR3         EQU     2               ;Flag to say check Priority 3
GO_PR1          EQU     3               ;Flag to say go back to Priority 1

*************************
* EXAMPLE TASK CONSTANT *
*************************

FINAL           EQU     4               ;To indicate last time round Task D

                ORG     RAM

********************************
* PRIORITY SCHEDULER VARIABLES *
********************************

JUMPLONG        RMB     8               ;Space to write a procedure in RAM
PR_LEVEL        RMB     1               ;Holds the priority level number
TASKREQ         RMB     3               ;Task request register
SHADOWTASK      RMB     3               ;Copy of the task request register
ADD_POINTER     RMB     1               ;Points to address in task table
SHIFTCNT        RMB     3               ;Number of shifts done on
TASKTEMP        RMB     1               ;Copy of SHADOWTASK for BRSET comm
SYSFLAG         RMB     1               ;Location for system holding flags
SETTASKS        RMB     1               ;In SCI routine to set tasks to run
```

```
*************************
* EXAMPLE TASK VARIABLES *
*************************

DELAY_VAR       RMB     1               ;Variable used in example routine
TIME_ON         RMB     1               ;Variable used in example routine
NUM_ON_LEDS     RMB     1               ;Controls seq of LEDS in example
APP_FLAG_REG    RMB     1               ;Varaible used in example routine
TEMP            RMB     1               ;Used in SCI interrupt service routine
TEMPLO          RMB     1               ;Used in SCI interrupt service routine
TEMPHI          RMB     1               ;Used in SCI interrupt service routine

                ORG     ROM

****************
* MAIN PROGRAM *
****************

SCHED05         JSR     INITIAL         ;Initialise Port A & RAM
                CLI                     ;Clear Interrupt Mask
SCHED99         JMP     PSCHED          ;Priority scheduler

**************
* PROCEDURES *
**************

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

* NAME: INITIAL                                                             *
*                                                                           *
* PURPOSE:      To initialise ports and clear all RAM locations used in the *
*               program.                                                    *
*                                                                           *
* SUBROUTINES USED:     CLEAR                                               *
*                                                                           *
* DESCRIPTION: Procedure sets all Port A pins as outputs                    *
*                                                                           *
*****************************************************************************

INITIAL         CLR     PORTA           ;Clear Port A
                LDA     #$FF            ;Set all pins as outputs
                STA     DDRA            ;
                JSR     CLEAR           ;Go to clear RAM locations
                RTS

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

CLEAR           CLRX                    ;
CLEAR05         CLR     RAM,X           ;Clear RAM location
                INCX                    ;Go to next location
                CPX     #$20            ;Cleared all the locations ?
                BLO     CLEAR05         ;If not go clear next location
                RTS                     ;Otherwise, exit
```

```
**************************************************************************
* NAME: PSCHED                                                           *
*                                                                        *
* PURPOSE:      This procedure is the control routine for the priority   *
*               scheduler. It controls which priority level task request *
*               register is inspected at what time.                      *
*                                                                        *
* ENTRY CONDITIONS:     The prioritys' task request registers will have  *
*                       been filled with flags corresponding to tasks in *
*                       the task table that the user wishes to execute, or *
*                       indeed if a task has set another task to execute, a *
*                       flag will be set in the task request register.   *
*                       All the RAM locations and port A will have been  *
*                       initialised.                                     *
*                                                                        *
* EXIT CONDITIONS:      This procedure is never exited.                  *
*                                                                        *
* SUBROUTINES USED:     PRIOR_1, PRIOR_2, PRIOR_3OR3, PRIOR_3, WRITERAM, *
*                       COPY, CHECKBIT0, SHIFTREG, INCSHIFT, CLRSHIFT,   *
*                       INC_LEVEL, UPDATE.                               *
*                                                                        *
* EXTERNAL VARIABLES USED:      JUMPLONG, PR_LEVEL, TASKREQ, SHADOWTASK, *
*                               ADD_POINTER, SHIFTCNT, TASKTEMP, SYSFLAG, *
*                               NUM_ON_LEDS, TIME_ON, NUM_FLASH, DELAY_VAR. *
*                                                                        *
* DESCRIPTION:  1. When a priority level is to be operated on, a copy will *
*                  be made of the corresponding task request register. The *
*                  original will then be cleared so that it can be updated *
*                  when new tasks require execution.                     *
*               2. Priority 1 will be checked first, starting form bit 0 *
*               3. After all these tasks have been checked and executed, *
*                  one Priority 2 task will be executed.                 *
*               4. If there are no Priority2 tasks at this time, a Priority *
*                  3 task will be executed.                              *
*               5. Every time a task has been executed, the bit in the   *
*                  copied task request register, which corresponds to the *
*                  task, shall be cleared.                               *
*               6. When any one of the copied task request registers is  *
*                  declared totally empty, it shall be updated again by  *
*                  copying the original corresponding task request register *
*                  In this way, any new tasks that require execution may be *
*                  given a time slot in which to execute.                *
*               7. After either  a Priority 2 task or Priority 3 task has *
*                  been executed, the scheduler will then go back and check *
*                  the updated Priority 1 task request register. If there *
*                  are any Priority 1 tasks to be executed, they will all *
*                  be executed before any further Priority 2 or Priority 3 *
*                  tasks.                                                *
*               8. The whole process will then be repeated.              *
**************************************************************************
```

```
PSCHED          JSR     PRIOR_1        ;Examine & Execute Priority 1 tasks
PSCHED05        JSR     PRIOR_2        ;Examine Priority 2  task reqest reg
PSCHED10        JSR     PRIOR_2OR3     ;Executes one Priority 2 or 3 task
                BRSET   TRY_PR3,SYSFLAG,PSCHED15 ;Go to examine Priority 3
                BRA     PSCHED         ;Go back to Priority 1
PSCHED15        JSR     PRIOR_3        ;Examine Priority 3
PSCHED99        BRA     PSCHED10       ;Go & execute a Priority 2 or 3 task

*****************************************************************************
*                                                                           *
* NAME: PRIOR_1                                                             *
*                                                                           *
* PURPOSE:      To examine the Priority 1 task request register and execute *
*               all the Priority 1 tasks set to execute at that time.       *
*                                                                           *
* EXIT CONDITIONS:      All Priority 1 task set to execute at that time     *
*                       have been completed.                                *
*                                                                           *
*****************************************************************************

PRIOR_1         CLRX                     ;
                STX     PR_LEVEL         ;Set priority level to 1
                JSR     COPY             ;Copy task req reg to a temp loc
                LDA     SHADOWTASK,X     ;Read this temporary location
                BEQ     PRIOR1_99        ;If its empty, go try Priority 2
PRIOR1_05       JSR     CHECKBIT0        ;Otherwise,go check bit 0
                BRSET   DO_TASK,SYSFLAG,PRIOR1_10;If bit 0 set,go do a task
                BRA     PRIOR1_15        ;Otherwise shift right
PRIOR1_10       JSR     WRITERAM         ;Go write subroutine in RAM
                JSR     JUMPLONG         ;Go execute the correct task
                INC     ADD_POINTER      ;Update address pointer
                BCLR    DO_TASK,SYSFLAG ;Clear flag to say done the task
PRIOR1_15       JSR     SHIFTREG         ;Shift tempoary register to right
                LDA     SHADOWTASK,X     ;Read the temporary register
                BEQ     PRIOR1_99        ;If reg now empty,go to Priority 2
                JSR     INCSHIFT         ;Otherwise, increment shift counter
                LDA     SHIFTCNT,X       ;Read value in shift counter
                CMP     #$07             ;Completed max number of shifts ?
                BHI     PRIOR1_99        ;If so, go try Priority 2
                BRA     PRIOR1_05        ;If not, try next bit in Priority 1
PRIOR1_99       RTS
```

```
*****************************************************************************
*                                                                           *
* NAME: PRIOR_2                                                             *
*                                                                           *
* PURPOSE:   To examine the Priority 2 task request register                *
*                                                                           *
* ENTRY CONDITIONS:     All priority 1 tasks have been executed.            *
*                                                                           *
* EXIT CONDITIONS:      A flag is set to say either, go execute one Priority*
*                       task, or go examine the Priority 3 task request     *
*                       register.                                           *
*                                                                           *
*****************************************************************************

PRIOR_2         JSR     CLRSHIFT        ;Clear previous shift counter
                JSR     INC_LEVEL       ;Increment priority level
                LDA     SHIFTCNT,X      ;Read present shift counter
                BNE     PRIOR2_05       ;If it <> 0,update address pointer
                JSR     COPY            ;Copy task req reg to a temp loc
PRIOR2_05       JSR     UPDATE          ;Update address pointer
                ADD     #$10            ;Set address pointer to start of
                STA     ADD_POINTER     ;correct section in the task table
                LDX     PR_LEVEL        ;
                LDA     SHADOWTASK,X    ;Read the temporary location
                BEQ     PRIOR2_10       ;If its empty, set flag TRY_PR3
                BRA     PRIOR2_99       ;Otherwise, exit
PRIOR2_10       BSET    TRY_PR3,SYSFLAG ;Set flag to say try Priority 3
PRIOR2_99       RTS
```

```
*****************************************************************************
*                                                                           *
* NAME: PRIOR_2OR3                                                          *
*                                                                           *
* PURPOSE: To execute either one Priority 2 or Priority 3 task.             *
*                                                                           *
* ENTRY CONDITIONS:     Flag set to say execute either a Priority 2 or      *
*                       Priority 3 task.                                    *
*                                                                           *
* EXIT CONDITIONS:      Either a Priority  2 task or a Priority 3 task has  *
*                       been executed.                                      *
*                                                                           *
*****************************************************************************

PRIOR_2OR3      BRSET   TRY_PR3,SYSFLAG,PRIOR23_99;If TRY_PR3 set, exit
                BRSET   GO_PR1,SYSFLAG,PRIOR23_20;If GO_PR1 set go PRIOR23
PRIOR23_05      JSR     CHECKBIT0       ;Otherwise try bit 0 in reg
                BRSET   DO_TASK,SYSFLAG,PRIOR23_10;If bit 0 set, go do task
                JSR     SHIFTREG        ;Otherwise, shift reg to the right
                JSR     INCSHIFT        ;Increment shift counter
                BRA     PRIOR23_05      ;Go check next bit
PRIOR23_10      JSR     WRITERAM        ;Go to write procedure in RAM
                JSR     JUMPLONG        ;Go to execute the task
                BCLR    DO_TASK,SYSFLAG ;Clear flag to say done task
                JSR     SHIFTREG        ;Shift reg to the right
                LDA     SHADOWTASK,X    ;Read the temporary location
                BEQ     PRIOR23_15      ;If now empty, go to PRIOR23_10
                JSR     INCSHIFT        ;Otherwise,increment shift counter
                LDA     SHIFTCNT,X      ;Read value of shift counter
                CMP     #$07            ;Done max number of shifts ?
                BLS     PRIOR23_20      ;If not, go to PRIOR23_15
PRIOR23_15      JSR     CLRSHIFT        ;Go clear shift counter
PRIOR23_20      CLRA                    ;Set address pointer back to
                STA     ADD_POINTER     ;start of Priority 1 addresses
                BCLR    GO_PR1,SYSFLAG  ;Clear flag, go back to Priority 1
PRIOR23_99      RTS
```

```
***************************************************************************
*                                                                         *
* NAME: PRIOR_3                                                           *
*                                                                         *
* PURPOSE:      To examine the Priority 3 task request register           *
*                                                                         *
*                                                                         *
* ENTRY CONDITIONS:     All the Priority 1 and Priority 2 tasks set to    *
*                       execute at that time have been completed.         *
*                                                                         *
* EXIT CONDITIONS:      A flag is set to say either go execute a Priority 3 *
*                       or go back to check Priority 1 task request register*
*                                                                         *
***************************************************************************

PRIOR_3         JSR     INC_LEVEL       ;Increment priority level
                LDA     SHIFTCNT,X      ;Read shift counter
                BNE     PRIOR3_05       ;If empty,go update address pointer
                JSR     COPY            ;Copy task req reg to a temp loc
PRIOR3_05       JSR     UPDATE          ;Update address pointer
                ADD     #$20            ;Set pointer to correct section
                STA     ADD_POINTER     ;in the task table
                BCLR    TRY_PR3,SYSFLAG ;Clear flag
                LDX     PR_LEVEL        ;Read the temporary task
                LDA     SHADOWTASK,X    ;request register
                BEQ     PRIOR3_10       ;If empty set flag,go to Priority 1
                BRA     PRIOR3_99       ;Othwise,go try bit 0
PRIOR3_10       BSET    GO_PR1,SYSFLAG  ;
PRIOR3_99       RTS
```

```
*****************************************************************************
*                                                                           *
* NAME: WRITERAM                                                            *
*                                                                           *
* PURPOSE:      To write a subroutine in RAM so that the scheduler can      *
*               access a 16-bit address, which is the address of the task in*
*               the task table.                                             *
*                                                                           *
* ENTRY CONDITIONS:     A flag has been set to say a task is to be executed *
*                                                                           *
* EXIT CONDITIONS:      The task corresponding to the bit set in the copy   *
*                       of the task request register has been executed.     *
*                                                                           *
* DESCRIPTION:          The opcode for "JSR" is copied to memory. Then the  *
*                       high byte and low byte are copied to different      *
*                       memory locations. Then the opcode for "RTS" is      *
*                       copied to memory. We then carry out the subroutine  *
*                       at the address in the task table.                   *
*                                                                           *
*****************************************************************************

WRITERAM        LDX     ADD_POINTER     ;Read the address in task table
                LDA     #$CD            ;Read the opcode for "JSR"
                STA     JUMPLONG        ;Copy it to location in memory
                LDA     TASKTABLE,X     ;Read the high byte of address
                STA     JUMPLONG+1      ;Copy this to next loc in JUMPLONG
                INCX                    ;Increment address
                STX     ADD_POINTER     ;
                LDA     TASKTABLE,X     ;Read the low byte of the address
                STA     JUMPLONG+2      ;Copy this to next loc in JUMPLONG
                LDA     #$81            ;Read in the opcode for "RTS"
                STA     JUMPLONG+3      ;Copy this at next loc in JUMPLONG
WRITERAM99      RTS

*****************************************************************************
*                                                                           *
* NAME: COPY                                                                *
*                                                                           *
* PURPOSE: Makes a copy of the original task request register.              *
*                                                                           *
*****************************************************************************

COPY            LDX     PR_LEVEL        ;Read the task request register
                LDA     TASKREQ,X       ;
                STA     SHADOWTASK,X    ;Copy it to a temporary location
                CLR     TASKREQ,X       ;Clear original
                RTS
```

```
*************************************************************************
*                                                                       *
* NAME: CHECKBIT0                                                       *
*                                                                       *
* PURPOSE:      Checks the first bit in the task request register to see if *
*               it is set. If so, a flag is set to say a task is to be  *
*               executed. If not the address pointer in the task table is *
*               updated to point to the next task in the task table.    *
*                                                                       *
*************************************************************************

CHECKBIT0       LDX     PR_LEVEL        ;Copy temporary location
                LDA     SHADOWTASK,X    ;to another temporary location so
                STA     TASKTEMP        ;can do a BRSET command
                BRSET   LSB,TASKTEMP,CHECK05;Bit 0 set,go execute a task
                INC     ADD_POINTER     ;Otherwise update address pointer
                INC     ADD_POINTER     ;to point to next task in task table
                BRA     CHECK99         ;
CHECK05         BSET    DO_TASK,SYSFLAG ;Set flag to say do a task
CHECK99         RTS

*************************************************************************
*                                                                       *
* NAME: SHIFTREG                                                        *
*                                                                       *
* PURPOSE:      This subroutine shifts the copied task request register one *
*               place to the right, so that it can search for a bit set in *
*               position zero.                                          *
*                                                                       *
*************************************************************************

SHIFTREG        LDX     PR_LEVEL        ;Perform logical shift right on
                LDA     SHADOWTASK,X    ;temporary location
                LSRA                    ;
                STA     SHADOWTASK,X    ;
                RTS

*************************************************************************
*                                                                       *
* NAME: INCSHIFT                                                        *
*                                                                       *
* PURPOSE:      This routine increments the shift counter of the priority *
*               level being operated on. A maximum of 7 shifts is       *
*               allowed in an 8-bit register, so this controls how many *
*               more bits in the register to check for a set bit.       *
*                                                                       *
*************************************************************************

INCSHIFT        LDX     PR_LEVEL        ;
                LDA     SHIFTCNT,X      ;Read shift counter
                INCA                    ;Increment shift counter
                STA     SHIFTCNT,X      ;
                RTS
```

```
***************************************************************************
*                                                                         *
* NAME: CLRSHIFT                                                          *
*                                                                         *
* PURPOSE:      To clear the present priority's shift counter before      *
*               starting work on another.                                 *
*                                                                         *
***************************************************************************

CLRSHIFT        LDX     PR_LEVEL        ;Clear previous priority shift
                LDA     SHIFTCNT,X      ;counter
                CLRA                    ;
                STA     SHIFTCNT,X      ;
                RTS

***************************************************************************
*                                                                         *
* NAME: INC_LEVEL                                                         *
*                                                                         *
* PURPOSE:      Increments the priority level when finished working on the *
*               present one.                                              *
*                                                                         *
***************************************************************************

INC_LEVEL       LDX     PR_LEVEL        ;Increment prority level
                INCX                    ;
                STX     PR_LEVEL        ;
                RTS

***************************************************************************
*                                                                         *
* NAME: UPDATE                                                            *
*                                                                         *
* PURPOSE:      Sets the address pointer to the start of the section in   *
*               the task table which holds the addresses for the tasks    *
*               in that priority.                                         *
*                                                                         *
***************************************************************************

UPDATE          LDX     PR_LEVEL        ;
                LDA     SHIFTCNT,X      ;Update address pointer to point
                LDX     #$02            ;to start of correct section
                MUL                     ;in the task table
                RTS
```

```
**************
* TASK TABLE *
**************

                ORG     TABLE

TASKTABLE       FDB     TASKA
                FDB     DUMMY           ;Unused entries point to dummy tasks
                FDB     DUMMY
                FDB     TASKD
                FDB     DUMMY
                FDB     DUMMY
                FDB     TASKG
                FDB     DUMMY

                FDB     DUMMY
                FDB     DUMMY
                FDB     DUMMY
                FDB     TASKL
                FDB     DUMMY
                FDB     DUMMY
                FDB     DUMMY
                FDB     DUMMY

FDB     DUMMY
                FDB     DUMMY
                FDB     DUMMY
                FDB     DUMMY
                FDB     TASKU
                FDB     DUMMY
                FDB     DUMMY
                FDB     TASKX

*****************************************************************************
*                          * TASKS FOLLOW *                                 *
*****************************************************************************

DUMMY           RTS                     ;Dummy task

TASKA           LDA     #$01            ;Example module
                STA     PORTB
                RTS


TASKD           LDA     #$10            ;Load in decimal 16
TASKD_05        STA     NUM_ON_LEDS     ;Store this value in memory
TASKD_10        LDA     NUM_ON_LEDS     ;Read this value
                BNE     TASKD_12        ;If not empty, go to decrement
                BSET    FINAL,APP_FLAG_REG;Set flag to exit after o/p a zero
                BRA     TASKD_15        ;Go to copy vaue back to memory
TASKD_12        DECA                    ;Decrement number shown on LEDs
```

```
TASKD_15         STA      NUM_ON_LEDS       ;Copy value back to memory
                 LSLA                       ;Shift left
                 LSLA                       ;    "
                 LSLA                       ;    "
                 LSLA                       ;    "
                 STA      PORTA             ;Send value to Port A
                 LDA      #$25              ;Load in HEX 25
                 STA      TIME_ON           ;Store this value in memory
TASKD_20         JSR      DELAY             ;Go to DELAY subroutine
                 DEC      TIME_ON           ;Decrement the value in TIME_ON
                 LDA      TIME_ON           ;Read the value
                 BNE      TASKD_20          ;If <> 0, go  back to delay again
                 BRSET    FINAL,APP_FLAG_REG,TASKD_99;If flag set, exit
                 BRA      TASKD_10          ;Otherwise, go to output next number
TASKD_99         BCLR     FINAL,APP_FLAG_REG;Clear flag before leaving routine
                 RTS                        ;Exit
***************************************************************************

DELAY            LDA      #$FF              ;Simple delay routine
OUTLP            DECA                       ;Keep looping round OUTLP until
                 BNE      OUTLP             ;accumulator is zero
                 INC      DELAY_VAR         ;Increment counter
                 LDA      DELAY_VAR         ;Read counter value
                 CMP      #$CC              ;Does it equal HEX CC
                 BLS      DELAY             ;If not go back and start agin
DELAY99          RTS                        ;Otherwise, exit

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

TASKG            LDA      #$04              ;Example module
                 STA      PORTB
                 RTS

TASKL            LDA      #$08              ;Example module
                 STA      PORTB
                 RTS

TASKU            LDA      #$10              ;Example module
                 STA      PORTB
                 RTS

TASKX            LDA      #$20              ;Example module
                 STA      PORTB
                 RTS

*********************************
* SCI INTERRUPT SERVICE ROUTINE *
*********************************

DATA             JSR      GETDATA           ;Checks for received data
                 STA      TEMP              ;Store received ASCII data in temp
                 AND      #$0F              ;Convert LSB of ASCII char to HEX
                 ORA      #$30              ;$3(LSB) =  "LSB"
                 CMP      #$39              ;3A-3F need to change to 41-46
```

```
                BLS     ARN1            ;Branch if 30-39 OK
                ADD     #7              ;Add offset
ARN1            STA     TEMPLO          ;Store LSB of HEX in TEMPLO
                LDA     TEMP            ;Read the original ASCII data
                LSRA                    ;Shift right 4 bits
                LSRA                    ;
                LSRA                    ;
                LSRA                    ;
                ORA     #$30            ;ASCII for N is $3N
                CMP     #$39            ;3A-3F need to change to 41-46
                BLS     ARN2            ;Branch if 30-39
                ADD     #7              ;Add offset
ARN2            STA     TEMPHI          ;MS nibble of HEX to TEMPHI
                LDA     #$0D            ;Load HEX value for "<LF>"
                BSR     SENDATA         ;Line feed
                LDA     #$24            ;Load HEX value "$"
                BSR     SENDATA         ;Print dollar sign
                LDA     TEMPHI          ;Get high half of HEX value
                BSR     SENDATA         ;Print
                LDA     TEMPLO          ;Get low half of HEX value
                BSR     SENDATA         ;Print

                CLRX                    ;These seven lines demonstrate
                CLR     SETTASKS        ;how flags are set in the Priority 1
                BSET    0,SETTASKS      ;(X=0) task request regiser in order
                BSET    1,SETTASKS      ;to set the corresponding tasks to
                BSET    2,SETTASKS      ;run. SETTASKS is used as a temporary
                LDA     SETTASKS        ;register since the operation
                STA     TASKREQ,X       ;BSET   0,TASKREQ,0, for instance,
                RTI                     ;cannot be done.

GETDATA         BRCLR   5,SCSR,GETDATA  ;RDRF = 1 ?
                LDA     SCDAT           ;OK, get data
                RTS                     ;

SENDATA         BRCLR   7,SCSR,SENDATA  ;TDRE = 1 ?
                STA     SCDAT           ;OK, send data
                RTS                     ;


SPI             RTI
TIRQ            RTI
IRQ             RTI
SWI             RTI

                ORG     VECTOR

                FDB     SPI             ;SPI interrupt vector
                FDB     DATA            ;SCI interrupt vector
                FDB     TIRQ            ;Timer interrupt vector
                FDB     IRQ             ;External interrrupt vector
                FDB     SWI             ;Software interrupt vector
                FDB     SCHED05         ;Reset interrupt vector
```

```
****************************************************************************
*                                               Copyright (c) Motorola 1993 *
*                                                                          *
*                       LISTING 2                                          *
*                       *********                                          *
*                                                                          *
* File name:  TIME_BASED.ASM                                               *
*                                                                          *
* Purpose: To co-ordinate the timing of exection of different              *
*          modules using the internal Free-Running Counter along           *
*          with the Output Compare or the Core Timer along with the        *
*          Core Timer Overflow funtion.                                    *
*          If the free-running counter is used to co-ordinate the          *
*          timing the tasks, which ever one it is, will be executed        *
*          every 4ms.                                                      *
*          If the Core Timer is used, the tasks will be executed           *
*          every 5.12ms.                                                   *
*                                                                          *
* Target device: 68HC705L4                                                 *
*                                                                          *
* Memory usage: ROM: 236 BYTES     RAM:  8 BYTES                           *
*                                                                          *
* Assembler: IASM05 - Integrated Assembler  Version :  3.02                *
*                                                                          *
* Description: Using the different timing registers inside the MCU         *
*              and setting up separate counters, the time intervals        *
*              between the execution of the different tasks can be        *
*              controlled using the Free-Running Counter along with        *
*              the Output Compare function or the Core Timer Counter       *
*              Register along with the Core Timer Overflow Flag.           *
*              If the programmable timer is used, an interrupt will        *
*              occur when the value in the Ouput Compare Register          *
*              equals the value of the Free-Running Counter.               *
*              If the Core Timer is used, an interrupt will occur          *
*              when the Core Timer Counter register rolls over from        *
*              $FF to $00.                                                 *
*              In this program it is at every 10 interrupts that a         *
*              task is executed.                                           *
*                                                                          *
*                                                                          *
* SUBROUTINES                                                              *
* -----------                                                              *
*                                                                          *
* Author: Joanne Santangeli Location:EKB        Created : 17 Jun 93        *
*                                         Last modified : 26 Aug 93        *
*                                                                          *
* Update history                                                           *
* Rev   Author  Date    Description of change                              *
* ---   ------  ----    ----------------------                             *
* 0.1   JS      26/9/93 INITIAL RELEASE                                    *
*                                                                          *
****************************************************************************
```

```
*****************************************************************************
* Motorola reseves the right to make changes without further notice         *
* to any product herein to improve reliability, function, or design.        *
* Motorola does not assume any liability  arising out of the                *
* application or use of any product , circuit, or software described        *
* herein; neither does it convey any license under its patent rights        *
* nor the right of others. Motorola products are not designed,              *
* intended or authorised for use as components in systems intended          *
* for surgical implant into the body, or other applications intended        *
* to support life, or for any other application in which failure            *
* of the Motorola product could create a situation where personal           *
* injury or death may occur. Should Buyer purchase or use Motorola          *
* products for any such intended or unauthorised application, Buyer         *
* shall indemnify and hold Motorola and its officers, employees             *
* subsidiaries, affiliates, and distributors harmless against all           *
* claims, costs, damages, expenses and reasonable attorney fees             *
* arising out of, directly or indirectly, any claim of personal             *
* injury or death associated with such unint  ended or unauthorised         *
* use, even if such claim alleges that Motorola was negligent               *
* regarding the design or manufacture of the part. Motorola and the         *
* Motorola logo* are registered trademarks of Motorola Ltd.                 *
*****************************************************************************

*********************
* PORT  DECLARATIONS *
*********************

PORTB           EQU     $01             ;Direct address - Port C
DDRB            EQU     $05             ;Data direction register - Port C
**********
* MEMORY *
**********

ROM             EQU     $2100           ;User ROM area in the MC68HC05L4
RAM             EQU     $0050           ;RAM area in the MC68HC05L4
VECTOR          EQU     $3FF6           ;Start of vector address

***************************
* CORE TIMER DECLARARTIONS *
***************************

TS_CTCSR        EQU     $08             ;Core Timer Control & Status Register
                                        ;CTOF,RTIF,CTOFE,RTIE,-,-,RT1,RT0
TV_CTCR         EQU     $09             ;Core Timer Counter Register

***********************************
* PROGRAMMABLE TIMER DECLARATIONS *
***********************************

TV_TCHA         EQU     $10             ;Timer A Counter Register (High)
TV_TCLA         EQU     $11             ;Timer A Counter Register (Low)
TV_ACHA         EQU     $12             ;Timer A Alt Counter Register (high)
TV_ACLA         EQU     $13             ;Timer A Alt Counter Register (low)
TV_TCRA         EQU     $0A             ;Timer A Control Register
```

```
TV_TSRA         EQU     $0B              ;Timer A Status Register
TV_ICHA         EQU     $0C              ;Input Capture A Register (High)
TV_ICLA         EQU     $0D              ;Input Capture A Register (Low)
TV_OCHA         EQU     $0E              ;Output Compare A Register (High)
TV_OCLA         EQU     $0F              ;Output Compare A Register (Low)

*******************************************************
* THE FOLLOWING ARE USED TO DETERMINE THE TASK TIMING *
*******************************************************

TW_OCPER        EQU     $C8              ;Output Compare Period set to 200
TW_TSPER        EQU     $0A              ;Time Slice Period set to 10

**************************
*  VARIABLE DECLARATIONS *
**************************

                ORG     RAM

TV_TSCP         RMB     1                ;Programmable Timer Slice Counter
TV_TSCC         RMB     1                ;Core Timer Time Slice Counter
TV_TSKCP        RMB     1                ;Programmable Timer Task Counter
TV_TSKCC        RMB     1                ;Core Timer Task Counter
TV_TSKC         RMB     1                ;Task Counter used to find task
TV_OPT          RMB     1                ;Option whether Core or Programmable
                                         ;Timer is used
TV_DTASK        RMB     1                ;To check if a task is to be carried
                                         ;out at that interrupt
TV_STORE        RMB     1                ;Bit 1 of this variable is clear or
                                         ;set depending on if a timer
                                         ;interrupt has occured or not when
                                         ;using the Programmable Timer


                ORG     ROM              ;Absolute address label for this
                                         ;section of ROM (MC68HC705L4)

*****************
*  MAIN PROGRAM *
*****************

T_SCHD05        BSET   0,TV_OPT          ;Set a flag to determine which timer
                LDA     #$FF             ;Set PB7-PB0 as outputs
                STA     DDRB             ;
                CLR     PORTB            ;Clear Port B
                CLR     TV_TSKCC         ;Clear Core Timer Task Counter
                CLR     TV_TSKCP         ;Clear Programmable Timer Task Counter

T_SCHD10        BRSET   0,TV_OPT,T_SCHD99;Branch to choose the
                JMP     T_CORE05         ;Core Timer or the
T_SCHD99        JMP     T_PROG05         ;Programmable Timer
```

```
***************
* SUBROUTINES *
***************

*****************************************************************************
*                                                                           *
* Name: T_PROG05                                                            *
*                                                                           *
* Subroutine: Performs co-ordination of task execution using the            *
*             Output Compare function of the Programmable Timer.            *
*                                                                           *
* Stack space used(bytes): 2                                                *
*                                                                           *
* Subroutines used: T_PRIN05,T_TASK05                                       *
*                                                                           *
* External variables used: TW_OCPER,TW_TSPER,TV_TSKCP,TV_OPT                *
*                                                                           *
* Description: This subroutine initially sets the first Output              *
*              Compare. It then waits for a timer interrupt to which        *
*              it sevices with an interrupt sevice routine. The             *
*              Output Compare is then updated and the Ouput Compare         *
*              flag is cleared. The routine then jumps to a                 *
*              subroutine to find the particular task and                   *
*              carries it out.                                              *
*                                                                           *
*****************************************************************************

T_PROG05        LDA     TV_TSRA         ;Clear Timer Status Register
                LDA     TV_OCLA         ;Compare flag cleared
                LDA     TV_TCLA         ;Timer overflow cleared
                LDA     TV_ICLA         ;Input capture flag cleared
                CLR     TV_OCHA         ;Clear Output Compare (High)
                CLR     TV_OCLA         ;Clear Output Compare (Low)
                CLR     TV_TSCP         ;Clear Time Slice Counter
                LDA     #$40            ;Load ACCA with 01000000
                STA     TV_TCRA         ;Set Output Compare Interrupt enable
PROG10          CLI                     ;Clear Interrupt Mask Bit
PROG15          BRSET   0,TV_DTASK,PROG20;If bit is set,go to task routine
                BRA     PROG15          ;If not set,wait for next interrupt
PROG20          JSR     T_TASK05        ;Jump to task routine
                BCLR    0,TV_DTASK      ;Clear task bit
PROG99          BRA     PROG10          ;Go wait for next interrupt
```

```
****************************************************************************
*                                                                          *
* Name:T_CORE05                                                            *
*                                                                          *
* Subroutine: Performs co-ordination of task execution using the           *
*             Core Timer Counter Register along with the Core Timer        *
*             overflow flag.                                               *
*                                                                          *
* Stack space used(bytes): 4                                               *
*                                                                          *
* Subroutines used: T_CRIN05,T_TASK05                                      *
*                                                                          *
* External varaibles used: TW_TSPER,T_TSKCC                                *
*                                                                          *
* Description: This subroutine initially sets the Core Timer Overflow      *
*              Enable. It then waits for an interrupt (ie. when Core       *
*              Timer Counter Register rolls over frrom $FF to $00 )        *
*              After returning from servicing the interrupt, it            *
*              checks to see if the Task Counter has been written to       *
*              If so, another subroutine is called to find which task      *
*              is to be executed and then this particular task is          *
*              carried out. The routine then waits for the next            *
*              interrupt.                                                  *
*                                                                          *
****************************************************************************

T_CORE05        CLR     TV_TSCC         ;Clear Core Time Slice Counter
                CLRA                    ;Clear ACCA
                STA     TS_CTCSR        ;Verify Overflow Flag is clear
                LDA     #$23            ;Load ACCA with 00100011
                STA     TS_CTCSR        ;Set Core Timer Overflow Enable,
                                        ;RT1 & RT0
CORE10          WAIT                    ;Wait for Interrupt
                BRSET   0,TV_DTASK,CORE20;If task bit set,go to task routine
                BRA     CORE10          ;If not,go wait for next interrupt
CORE20          JSR     T_TASK05        ;Jump to task routine
                BCLR    0,TV_DTASK      ;Clear task bit
                BRA     CORE10          ;Go to wait for next interrupt
```

```
*****************************
* INTERRUPT SERVICE ROUTINES *
*****************************

****************************************************************************
*                                                                          *
* Name: T_PRIN05                                                           *
*                                                                          *
* Subroutine: Checks if a task is to be carried out at this                *
*             interrupt and updates the Output Compare register.           *
*                                                                          *
* Stack space used(bytes): 4                                               *
*                                                                          *
* Subroutines used: none                                                   *
*                                                                          *
* External variables used: TW_TSPER,,TV_TSKCP,TW_OCPER                     *
*                                                                          *
* Description: This interrupt sevice routine finds out if a task           *
*              by incrementing a Time Slice Counter. Each time the         *
*              interrupt sevice routine is called the counter is           *
*              incremented. Only when this counter equals ten, is          *
*              a task carried out.                                         *
*              After deciding whether a task is to be carried out,         *
*              the Output Compare Register is updated, ready to            *
*              for another interrupt and the Output Compare Flag           *
*              is cleared.                                                 *
*                                                                          *
****************************************************************************

T_PRIN05        BRCLR   6,TV_TSRA,PRIN99;Checks for Output Compare Flag
                INC     TV_TSCP         ;Inrement Time Slice Counter
                LDA     TV_TSCP         ;Read the Time Slice Counter
                CMP     #TW_TSPER       ;Compare contents of ACCA with 10
                BLO     PRIN10          ;If < 10, branch back to T_SCHED10
                CLR     TV_TSCP         ;If = 10, clear Time Slice Counter
                INC     TV_TSKCP        ;Increment Task Counter
                BSET    0,TV_DTASK      ;Set task bit
PRIN10          LDA     TV_OCLA         ;Read high byte of Output Compare
                ADD     #TW_OCPER       ;Load #200 into ACCA
                STA     TV_OCLA         ;Store in Output Compare (Low)
                LDA     TV_OCHA         ;Read Output Compare (High)
                ADC     #$00            ;Add the contents of the Carry bit
                STA     TV_OCHA         ;Store at Output Compare (High)
                LDA     TV_OCLA         ;Read Output Compare (low)
                STA     TV_OCLA         ;Write back to Output Compare (low)
PRIN99          RTI                     ;Return from Timer Interrupt
```

```
*************************************************************************
*                                                                       *
* Name:T_CRIN05                                                         *
*                                                                       *
* Subroutine: This routine finds if a tassk is to be carried out at     *
*             this interrupt. It also clears the Core Timer Overflow    *
*             flag.                                                     *
*                                                                       *
* Stack space used (bytes) : 4                                          *
*                                                                       *
* Subroutines used: none                                                *
*                                                                       *
* External varaibles used: TW_TSPER,TV_TSKCC                            *
*                                                                       *
* Description: Initially finds if Time Slice Counter equals             *
*              Time Slice Period. If so, the Slice counter is cleared   *
*              and the Task Counter is incremented. The Core Timer      *
*              Overflow Flag is then reset.                             *
*                                                                       *
*************************************************************************

T_CRIN05        INC     TV_TSCC         ;Increment Core Time Slice Counter
                LDA     TV_TSCC         ;Read Time Slice Counter
                CMP     #TW_TSPER       ;Compare this to Time Slice Period
                BLO     CRIN10          ;If < 10,go to update status register
                CLR     TV_TSCC         ;If = 10, clear Time Slice Counter
                INC     TV_TSKCC        ;Increment Core Task Counter
                BSET    0,TV_DTASK      ;Set task bit
CRIN10          LDA     #$23            ;Load ACCA with 00100011
                STA     TS_CTCSR        ;Clear Overflow Flag
                RTI                     ;Return from Interrupt

*************************************************************************
*                                                                       *
* Name: T_TASK05                                                        *
*                                                                       *
* Subroutine: Routine to find out which task is to be done  and         *
*             carries it out accordingly.                               *
*                                                                       *
* Stack space used(bytes): 4                                            *
*                                                                       *
* Subroutines used: none                                                *
*                                                                       *
* External varaibles used: TV_TSKCC,TV_TSKCP                            *
*                                                                       *
* Description: Depending on which bit contains a zero in the Task       *
*              Counter determines which task is to be carried out.      *
*              The task to be executed detected and carried out.        *
*              Each example task shown here each writes a logic         *
*              high to a different pin at Port B to demonstrate how     *
*              the tasks are scheduled.                                 *
*************************************************************************
```

```
**************
* TASK TABLE *
**************

T_TASK05        LDA     TV_TSKCC                ;Read Core Timer Task Counter
                BNE     TASK15                  ;Check if Core Timer or
TASK10          LDA     TV_TSKCP                ;Programmable has been used
TASK15          STA     TV_TSKC                 ;Stores task in memory
                BRCLR   0,TV_TSKC,TASK20        ;If bit 0 clear,go to Task A
                BRCLR   1,TV_TSKC,TASK25        ;If bit 1 clear,go to Task B
                BRCLR   2,TV_TSKC,TASK30        ;If bit 2 clear,go to Task C
                BRCLR   3,TV_TSKC,TASK35        ;If bit 3 clear,go to Task D
                BRCLR   4,TV_TSKC,TASK40        ;If bit 4 clear,go to Task E
                BRCLR   5,TV_TSKC,TASK45        ;If bit 5 clear,go to Task F
                BRCLR   6,TV_TSKC,TASK50        ;If bit 6 clear,go to Task G
                BRCLR   7,TV_TSKC,TASK55        ;If bit 7 clear,go to Task H
                CLRA                            ;Clear Port B if Task
                STA     PORTB                   ;Counter at #$FF
                RTS                             ;Return from routine
TASK20          JSR     T_20                    ;Jump to first module
                RTS                             ;
TASK25          JSR     T_25                    ;Jump to second module
                RTS                             ;
TASK30          JSR     T_30                    ;Jump to third module
                RTS                             ;
TASK35          JSR     T_35                    ;Jump to fourth module
                RTS                             ;
TASK40          JSR     T_40                    ;Jump to fifth module
                RTS                             ;
TASK45          JSR     T_45                    ;Jump to sixth module
                RTS                             ;
TASK50          JSR     T_50                    ;Jump to seventh module
                RTS                             ;
TASK55          JSR     T_55                    ;Jump to eighth module
                RTS                             ;
```

```
****************
* TASKS FOLLOW *
****************

T_20            LDA     #$01                    ;Example module
                STA     PORTB
                RTS

T_25            LDA     #$02                    ;Example module
                STA     PORTB
                RTS

T_30            LDA     #$04                    ;Example module
                STA     PORTB
                RTS

T_35            LDA     #$08                    ;Example module
                STA     PORTB
                RTS

T_40            LDA     #$10                    ;Example module
                STA     PORTB
                RTS

T_45            LDA     #$20                    ;Example module
                STA     PORTB
                RTS

T_50            LDA     #$40                    ;Example module
                STA     PORTB
                RTS

T_55            LDA     #$80                    ;Example module
                STA     PORTB
                RTS

IRQ             RTI
SWI             RTI

                ORG     VECTOR

                FDB     T_PRIN05                ;Programmable Interrupt Vector
                FDB     T_CRIN05                ;Core Timer Interrupt Vector
                FDB     IRQ                     ;Hardware Int
                FDB     SWI                     ;Software Int
                FDB     T_SCHD05                ;RESET Interrupt Vector
```

How to reach us:

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE (602) 244-6609
INTERNET: http://Design-NET.com
USA/EUROPE: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036. 1-800-441-2447
JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, Toshikatsu Otsuki, 6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 03-3521-8315
HONG KONG: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

MOTOROLA

AN1262/D