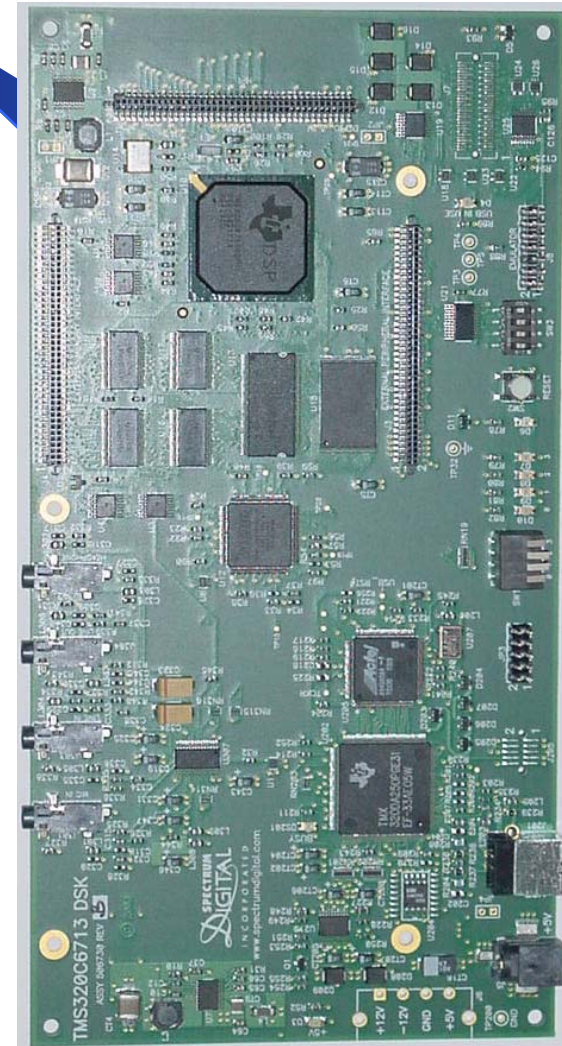


# TMS320C6713 Development System Kit (C6713DSK)

- 225Mhz TMS320C6713 Processor
- 8M SDRAM / 128K Flash
- Audio CODEC
- USB 2.0 JTAG Interface
- Dual Voltage Bus Interface
- C Compiler/Debugger
- All for \$395



### 13.2 Timer Registers

Table 13–2 describes the three registers that configure timer operation.

Table 13–2. Timer Registers

Hex Byte Address			Name and Abbreviation	Description	Section
Timer 0	Timer 1	Timer 2			
01940000	01980000	01AC0000	Timer Control (CTL)	Determines the operating mode of the timer, monitors the timer status, and controls the function of the TOUT pin.	13.2.1
01940004	01980004	01AC0004	Timer Period (PRD)	Contains the number of timer input clock cycles to count. This number controls the TSTAT signal frequency.	13.2.2
01940008	01980008	01AC0008	Timer Counter (CNT)	Current value of the incrementing counter	13.2.3

#### 13.2.1 Timer Control Register (CTL)

Figure 13–2 shows the timer control register. Table 13–3 describes the fields in this register.

Figure 13–2. Timer Control Register (CTL)

31		12		11	10	9	8
Rsvd				TSTAT	INVINP	CLKSRC	C/P
R, +0				R, +0	RW, +0	RW, +0	RW, +0
7	6	5	4	3	2	1	0
HLD	GO	Rsvd	PWID	DATIN	DATOUT	INVOUT	FUNC
RW, +0	RW, +0	R, +0	RW, +0	R, +X	RW, +0	RW, +0	RW, +0

Table 13–3. Timer Control Register (CTL) Field Descriptions

No.	Bitfield	Description	Section
31–12	Rsvd	Reserved.	
11	TSTAT	Timer status. Value of timer output.	13.6
10	INVINP	TINP inverter control. Only affects operation if CLKSRC = 0. INVINP = 0: Uninverted TINP drives timer. INVINP = 1: Inverted TINP drives timer.	13.5
9	CLKSRC	Timer input clock source CLKSRC = 0: External clock source drives the TINP pin. CLKSRC = 1: Internal clock source. For C62x/C67x: CPU clock/4 For C64x: CPU clock/8	13.5
8	C/P	Clock/pulse mode C/P = 0: Pulse mode. TSTAT is active one CPU clock after the timer reaches the timer period. PWID determines when it goes inactive. C/P = 1: Clock mode. TSTAT has a 50% duty cycle with each high and low period one countdown period wide.	13.6
7	HLD	Hold. Counter may be read or written regardless of HLD value. HLD = 0: Counter is disabled and held in the current state. HLD = 1: Counter is allowed to count.	13.3
6	GO	GO bit. Resets and starts the timer counter. GO = 0: No effect on the timers. GO = 1: If HLD = 1, the counter register is zeroed and begins counting on the next clock.	13.3
5	Rsvd	Reserved.	
4	PWID	Pulse width. Only used in pulse mode (C/P = 0). PWID = 0: TSTAT goes inactive one timer input clock cycle after the timer counter value equals the timer period value. PWID = 1: TSTAT goes inactive two timer input clock cycles after the timer counter value equals the timer period value.	13.6
3	DATIN	Data in: Value on TINP pin	13.5, 13.9
2	DATOUT	Data output When FUNC = 0: The DATOUT is driven on TOUT. When FUNC = 1: The TSTAT is driven on TOUT after inversion by INVOUT.	13.9

## 2.1.2 CPLD Registers

The 4 CPLD memory-mapped registers allows users to control CPLD functions in software. On the 6713 DSK the registers are primarily used to access the LEDs and DIP switches and control the daughter card interface. The registers are mapped into EMIF CE1 data space at address 0x90080000. They appear as 8-bit registers with a simple asynchronous memory interface. The following table gives a high level overview of the CPLD registers and their bit fields:

The table below shows the bit definitions for the 4 registers in CPLD.

**Table 1: CPLD Register Definitions**

Offset	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	USER_REG	USR_SW3 R	USR_SW2 R	USR_SW1 R	USR_SW0 R	USR_LED3 R/W 0(Off)	USR_LED2 R/W 0(Off)	USR_LED1 R/W 0(Off)	USR_LED0 R/W 0(Off)
1	DC_REG	DC_DET R	0	DC_STAT1 R	DC_STAT0 R	DC_RST R 0(No reset)	0	DC_CNTL1 R/W 0(low)	DC_CNTL0 R/W 0(low)
4	VERSION	CPLD_VER[3:0] R				0	BOARD_VERSION[2:0] R		
6	MISC	SCR_5 R/W 0	SCR_4 R/W 0	SCR_3 R/W 0	SCR_2 R/W 0	SCR_1 R/W 0	FLASH_PAGE R/W 0 (Flash A19=0)	McBSP1 ON/OFF Board R/W 0 (Onboard)	McBSP0 ON/OFF Board R/W 0 (Onboard)

## 2.1.3 USER\_REG Register

USER\_REG is used to read the state of the 4 DIP switches and turn the 4 LEDs on or off to allow the user to interact with the DSK. The DIP switches are read by reading the top 4 bits of the register and the LEDs are set by writing to the low 4 bits.


**Table 2: CPLD USER\_REG Register**

Bit	Name	R/W	Description
7	USER_SW3	R	User DIP Switch 3(1 = Off, 0 = On)
6	USER_SW2	R	User DIP Switch 2(1 = Off, 0 = On)
5	USER_SW1	R	User DIP Switch 1(1 = Off, 0 = On)
4	USER_SW0	R	User DIP Switch 0(1 = Off, 0 = On)
3	USER_LED3	R/W	User-defined LED 3 Control (0 = Off, 1 = On)
2	USER_LED2	R/W	User-defined LED 2 Control (0 = Off, 1 = On)
1	USER_LED1	R/W	User-defined LED 1 Control (0 = Off, 1 = On)
0	USER_LED0	R/W	User-defined LED 0 Control (0 = Off, 1 = On)

# Single thread Scheduler


All processes called once each sample

```
void main(void) {  
  
    init_routines();  
  
    done = 0;  
    while (!done) {  
  
        perform_process1(); // Highest priority process  
        perform_process2();  
        perform_process3(); // Lowest priority  
  
        wait_for_sample_period(); //waste time here waiting for sample period to expire  
  
    }  
}
```



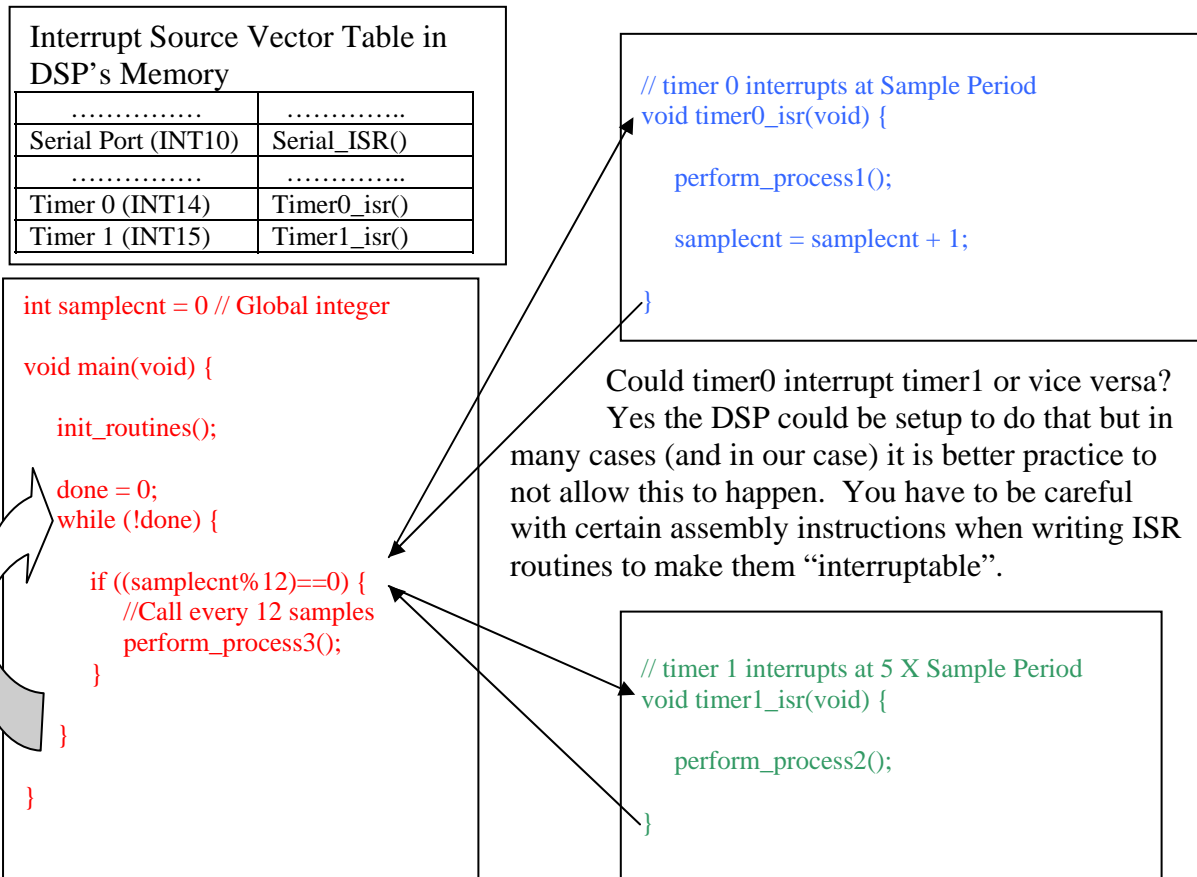
What if the three processes have different sample rates? With a single thread scheduler the following would work but all sample rates would have to be a multiple of the fastest rate.

```
void main(void) {  
    init_routines();  
    done = 0;  
    samplecnt = 0  
    while (!done) {  
  
        perform_process1(); // Call every sample  
  
        if ((samplecnt%5)==0) perform_process2(); // Call every 5 samples  
  
        if ((samplecnt%12)==0) perform_process3(); // Call every 12 samples  
  
        wait_for_sample_period(); //waste time here waiting for sample period to expire  
        samplecnt = samplecnt+1;  
  
    }  
}
```



Here the problem becomes that all processes (in this case all three) need to complete in the time of one sample period. For example at samplecnt = 60 all three processes are called during a single sample period. What if process 3 takes longer than a single sample rate?

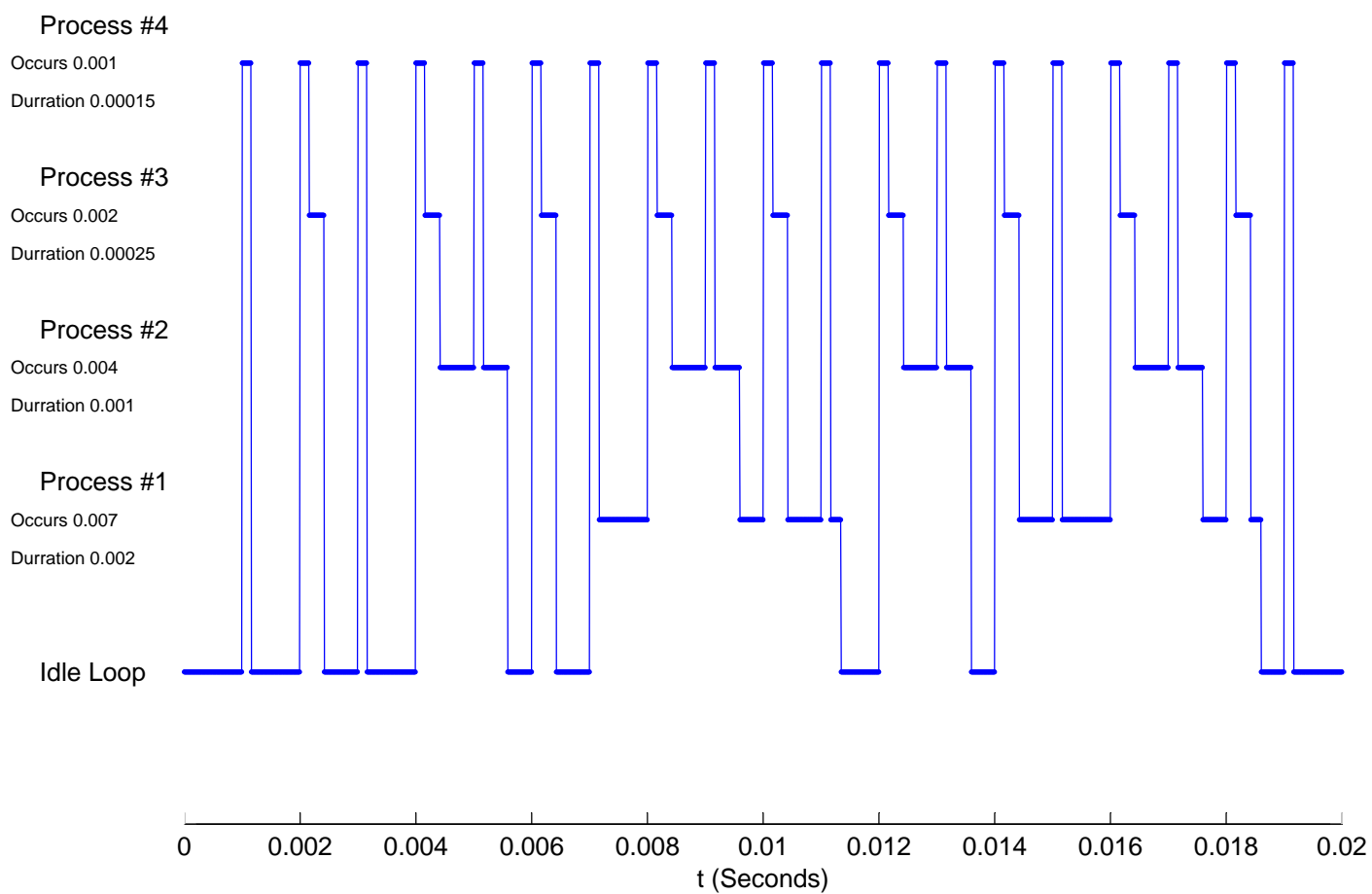
# Hardware Interrupt Scheduler



The while loop inside main() now becomes the low priority processing loop. Also called the "background" loop. Process 1 and Process 2 have the highest priority. When either timer 0 or timer 1 counts down to 0, the DSP's hardware automatically stops the current code running in the background loop and jumps to the function specified in the Interrupt Vector Table, in this case timer0\_isr() or timer1\_isr(). When the DSP is done running the instructions in the corresponding interrupt service routine (ISR), the DSP's hardware automatically returns and continues processing where it left off in the background loop.

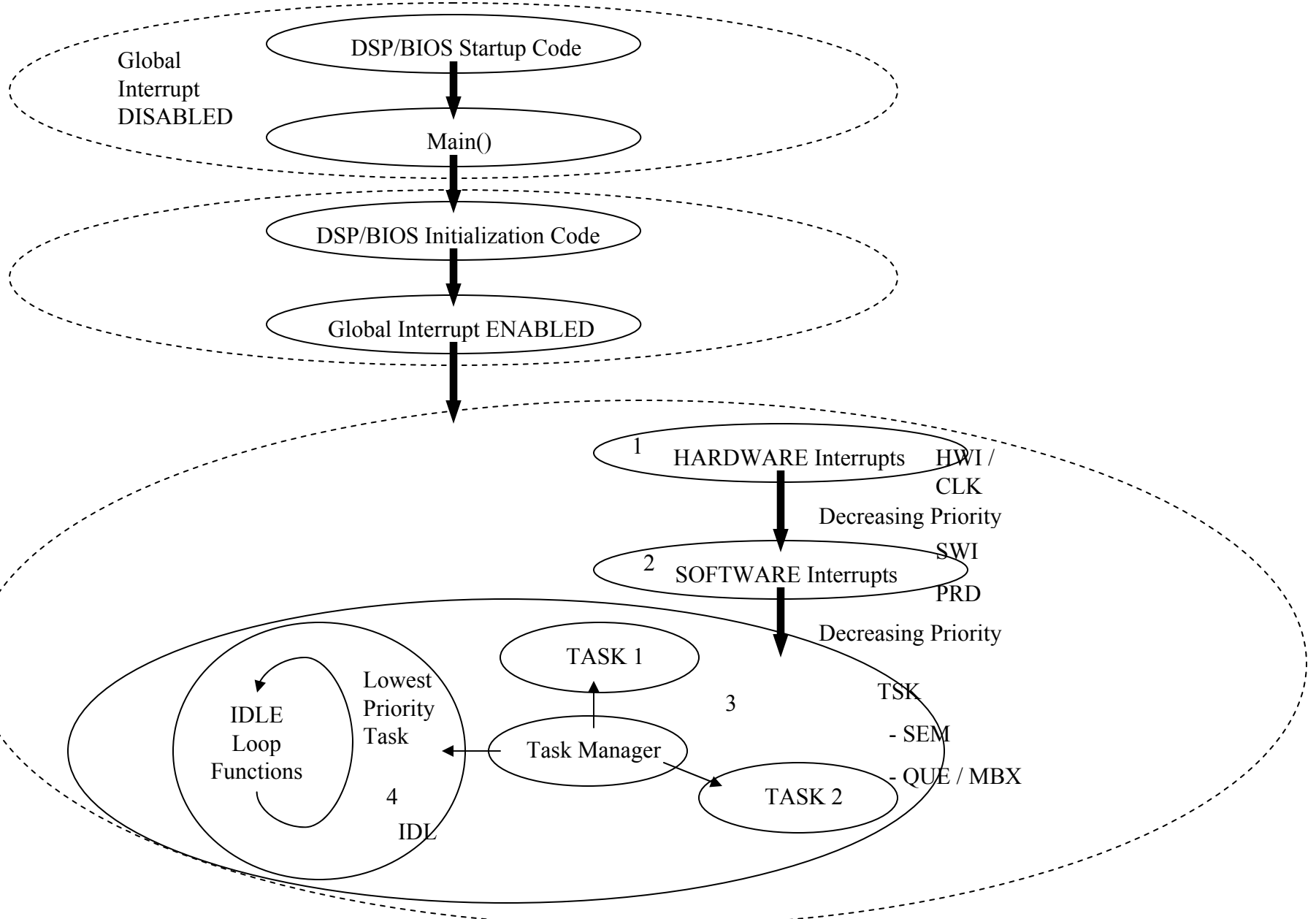
If timer 0 and timer 1 timeout at exactly the same time, timer 0 has the highest priority so its code will run first to completion and then timer 1's code will be executed. If a timer interrupt occurs while the other timer's interrupt service routine code is running, the running code continues to completion and then the other timer's code is executed.

### Time Load Graph





# DSP/BIOS



Code Composer - [Config1 \*]

File Edit Object View Project Debug Profiler GEL Option Tools PBC DSP/BIOS Window Help

Toolbar with icons for file operations (Save, Print, Copy, Paste, Undo, Redo), navigation (Home, Back, Forward), and development (Build, Run, Stop, Refresh, Search).

Files pane showing a tree structure:

- Files
  - GEL files
  - Projects

System tree structure:

- System
  - Instrumentation
    - LOG - Event Log Manager
    - STS - Statistics Object Manager
  - Scheduling
    - CLK - Clock Manager
    - PRD - Periodic Function Manager
    - HWI - Hardware Interrupt Service Routine Manager
      - HWI\_RESET
      - HWI\_NMI
      - HWI\_RESERVED0
      - HWI\_RESERVED1
      - HWI\_INT4
      - HWI\_INT5
      - HWI\_INT6
      - HWI\_INT7
      - HWI\_INT8
      - HWI\_INT9
      - HWI\_INT10
      - HWI\_INT11
      - HWI\_INT12
      - HWI\_INT13
      - HWI\_INT14
      - HWI\_INT15
    - SWI - Software Interrupt Manager
    - TSK - Task Manager
      - TSK\_idle
    - IDL - Idle Function Manager
    - Synchronization
      - SEM - Semaphore Manager
      - MBX - Mailbox Manager
      - QUE - Atomic Queue Manager
      - LCK - Resource Lock Manager
    - Input/Output
    - CSL - Chip Support Library

Large empty grey area, likely reserved for a diagram or code editor.

EMULATOR DISCONNECTED For Help, press F1

# DSP/BIOS Example

TSK Level

Arrows in the TSK and between the TSK and HWI Levels indicate direction of multi-thread communication.

User Created PRDs, SWIs or TSKs call for example SendWireless(..) to send a message to the PC. SendWireless, places the message in the Queue, SendStrmsgQueue, and the Activates the Semaphore, SEM\_SendStrmsg\_rdy.

Semaphore: SEM\_SendStrmsg\_rdy  
Queue: SendStrmsgQueue

User Defined Receive TSK waits for a new character by suspending (or blocking) itself until the semaphore, SEM\_UART1RecChar\_rdy is set active by HWI 4's function. The new character is then read from the Queue, UART1RecCharQueue. After receiving this character, the task loops back to the beginning of its code and again blocks itself to wait for the next character to be sent.

Semaphore: SEM\_UART1RecChar\_rdy  
Queue: UART1RecCharQueue

TSK\_UART  
Function: uarttsk

- Blocks itself from running until the semaphore, SEM\_SendStrmsg\_rdy, is set active.
- Fills global array "txbuffer" with the characters given in the Queue, SendStrmsgQueue.
- Enables Timer1 to Send the Characters out at the specified baud rate.
- Blocks itself from running until the Semaphore, SEM\_SendStrmsg\_done, is set active.
- Loops back to the top to wait for next message to send.

HWI Level

Global char Array: txbuffer

Semaphore: SEM\_SendStrmsg\_done

HWI\_INT4  
Function: extint4\_cisr  
Receives an interrupt from the MAX3100 chip when there is a new character received on the RS-232 side of the serial port.

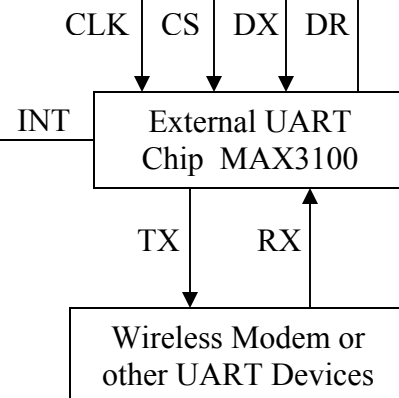
HWI\_INT15  
Function: timer1\_cisr  
Timer 1 is setup to send out a new 8 bit character on the SPI serial port at the specified baud rate.

Read SPI registers      Write to SPI registers

DSP's SPI (McBSP)  
Serial Port

Hardware Level

Arrows in the Hardware Level indicate actual signals and their Direction.



RS-232 standard serial port. (The standard serial port on the back of you PC.)