March, 1991

H8/330

Application Note
Power-Down Operation

Tom Hampton

INTRODUCTION

The H8/330 devices have three different power-down states of operation that significantly reduce power consumption by stopping some (or all) of the on-chip functions. These three modes differ not only for power consumption reduction, but also in how the entry and exit methods. Figure 1 shows a simple flow chart of the processing states for the H8/300 CPU.

This flow chart describes the processing sequence for all exception processing as well as power-down modes of operation. Table 1 shows power consumption during all modes of operation while Table 2 shows an overview of the individual modes for power-down operation.

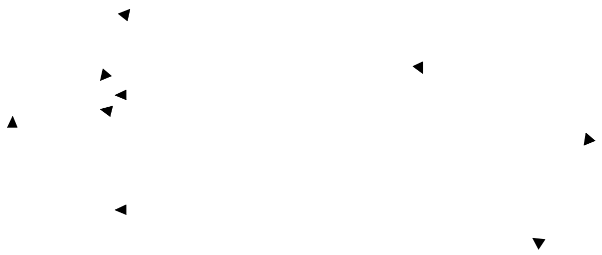

Figure 1: H8/330 Processing States

	Typical	Maximum		Frequency
Normal	12	25	mA	6MHz
	16	30	mA	8MHz
	20	40	mA	10MHz
Sleep Mode	8	15	mA	6MHz
	10	20	mA	8MHz
	12	25	mA	10MHz
Standby Modes	0.01	5.0	µA	

Table 1: H8/330 Power Consumption

Mode	Entrance	Clock	CPU	CPU Registers	On-Chip Peripherals	On-Chip RAM	I/O Ports	Exiting
Sleep Mode	Execute SLEEP Instruction	Run	Halt	Held	Run	Held	Held	Interrupt $\overline{\text{RES}}$ $\overline{\text{STBY}}$
Software Standby Mode	SSBY=1, Execute SLEEP Instruction	Halt	Halt	Held	Halt and Initialized	Held	Held	$\overline{\text{NMI}}$ $\overline{\text{IRQ2}}$-$\overline{\text{IRQ0}}$ $\overline{\text{RES}}$ $\overline{\text{STBY}}$
Hardware Standby Mode	$\overline{\text{STBY}}$ pin Active	Halt	Halt	Not Held	Halt and Initialized	Held	High Impedance	$\overline{\text{STBY}}$ high, then pulse $\overline{\text{RES}}$

Table 2: H8/330 Power-Down Modes

HARDWARE STANDBY MODE

The "Hardware Standby" mode of power-down operation is controlled by an external input pin ($\overline{STBY}$) on the H8/330. When the input to the $\overline{STBY}$ pin is made active, the H8/330 enters the hardware standby mode after completion of the current instruction.

Operation of the CPU and all on-chip peripherals are stopped completely during this mode of operation. The system oscillator is also stopped, to reduce power consumption to its minimum, so that no clock is supplied to any of the parts (CPU or peripherals) of the H8/330. Not only is the system clock stopped, but all the I/O ports on the H8/330 are placed into a high-impedance state. This inhibits the I/O ports from dribing or sourcing any external devices.

Only the on-chip RAM of the H8/330 is maintained during this mode of operation. Whatever values are placed into this RAM area are retained while nothing else is saved. (For further power reduction savings while still maintaining RAM data, please refer to the Special Considerations section later in this document.)

The H8/330 device remains in this mode of operation since the $\overline{STBY}$ pin remains active, despite the state of any other inputs (including $\overline{RES}$). The only way to remove the H8/330 from this mode of operation is with the following sequence:

1. release the $\overline{STBY}$ pin to the inactive state,
2. reset the device by pulsing the $\overline{RES}$ pin (see Figure 2 for the timing relationships on performing this function).

Since you are resetting the H8/330, you probably will only use this mode of operation when the H8/330 is used to initialize some external devices and then go to sleep until the external devices requires operations from the H8/330. Because the on-chip RAM is maintained during this mode of operation, you can place software semaphores in the RAM that will allow the initialization routines of the H8/330 to decide whether to do a complete re-initialization (as from a power-on condition) or a re-initialization of only itself (as in waking up from the hardware standby mode).

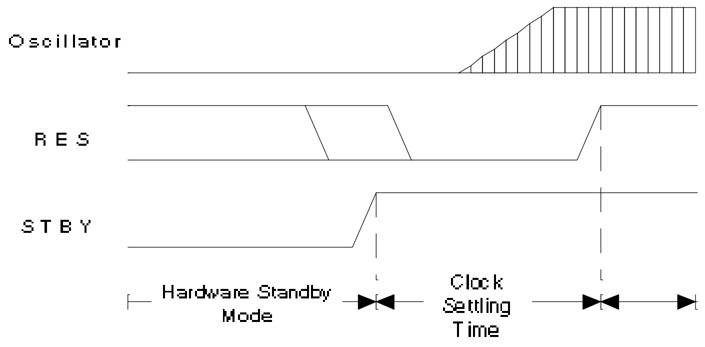

Figure 2: Exiting Hardware Standby Mode

SOFTWARE STANDBY MODE

The software standby mode of operation is very similar to the hardware standby mode, the same power consumption savings are available in either mode. Like the hardware standby mode of operation, the CPU and all on-chip peripherals are stopped completely during the software standby mode. The difference between the two modes is how they are entered and exited, and in how the CPU's registers and the I/O ports are handled.

The software standby mode of operation is controlled via software operation instead of hardware. There are two power-down functions controlled in software by programming the System Control Register (See Figure 3); the entering of the software standby mode and the time delay when leaving the software standby mode.

This mode of operation is entered by setting the "software standby bit" (SSBY) in the System Control Register (SYSCR) and then executing the SLEEP instruction. When the SLEEP instruction is executed, the SSBY bit is tested to find its value. If this bit is not set, then the H8/330 enters the "Sleep" mode of operation (discussed later in this document). If this bit is set, then the H8/330 enters the software standby mode of operation.

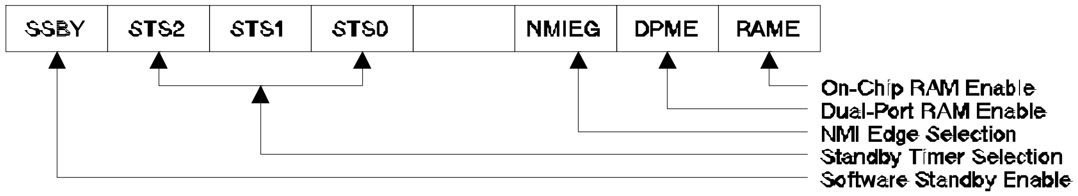

Figure 3: System Control Register

Before executing the SLEEP instruction, the user must program not only the SSBY bit in the SYSCR, but also the "Standby Timer Select" (STS_2-STS_0) bits. Since the on-chip oscillator is stopped during this mode of operation, enough time must be allowed to allow the oscillator to re-start (AC parameter t_{OSC2}). The user can control this time by programming these three bits. By setting them to different values, the user controls how many clock cycles the CPU delays between recognizing the external interrupt signal and starting the exception processing service routine (see Table 3).

Unlike the hardware standby mode, this mode of operation maintains the registers of the CPU. This allows program execution to continue at the location following the SLEEP instruction when the H8/330 is released from the software standby mode. Also during this mode of operation, the I/O ports are maintained in their current states instead of being re-initialized. But, the on-chip peripherals (such as timers, serial channel, etc.) are reset and must be re-initialized whenever the H8/330 is released from software standby mode.

Since the on-chip peripherals are not operating during the software standby mode, it is only external interrupts ($\overline{NMI}$ or $\overline{IRQ_2}$-$\overline{IRQ_0}$) that can awaken the H8/330 and return it to its normal operating sequence. This is handled just like any other exception sequence. The interrupting device is serviced after the oscillator settling time delay by the exception processing routine and operation is returned to the location following the SLEEP instruction.

This mode is probably the most useful of the power-down modes of operation because it offers the most power consumption savings. The CPU and on-chip peripherals are stopped while external devices (and on-chip I/O ports) are still allowed to function. This allows the user to have the rest of his system monitor external events while the CPU remains inactive.

Of course, you can always leave the software standby mode of operation by resetting the H8/330 or by entering the hardware standby mode.

			Settling Time	System Clock Frequency (MHz)						
STS2	STS	STS0		10	8	6	4	2	1	0.5
0	0	0	8192	0.8	1.0	1.3	2.0	4.1	8.2	**16.4**
0	0	1	16384	1.6	2.0	2.7	4.1	8.2	**16.4**	32.8
0	1	0	32768	3.3	4.1	5.5	8.2	**16.4**	32.8	65.5
0	1	1	65536	6.6	8.2	**10.9**	**16.4**	32.8	65.5	131.1
1	-	-	131072	**13.1**	**16.4**	21.8	32.8	65.5	131.1	262.1

Table 3: Standby Timer Select Values

SLEEP MODE

The “Sleep” mode of power-down operation is controlled by software. During this mode, operation of the H8/300 CPU core is halted while the rest of the on-chip functions remain active. Because of this, the “Sleep” mode offers the least amount of power consumption savings.

This mode of operation is controlled by executing the SLEEP instruction during the normal program operation. When this occurs, the H8/300 CPU is placed into a “halt” state with no further activity taking place. This halt state is similar to the situation where the CPU may be in an indefinite “wait” state except that no control signals are active.

Since the CPU is halted, no change in the I/O ports will occur (meaning that their current values will be held). Though the CPU is no longer running, all values in the registers are held in their current state. By doing this, the CPU is allowed to continue its operations directly from the location following the SLEEP instruction (after processing a return from the sleep mode).

Though the CPU is halted, the system clock is still allowed to run. This means that the on-chip peripherals can still function; the timers, the serial channel, the A/D converter, and the Dual-Port RAM can still do all their normal operations. In fact the H8/330 device gets out of the sleep mode of operation.

Whenever any of the on-chip peripherals generate an interrupt or an external interrupt is input to the device, the CPU is awakened from its sleep mode and processing continues as normal (see Figure 1 for flow details). The interrupting device is serviced during the exception processing routine and operation is returned to the location following the SLEEP instruction.

Like the Software Standby Mode, you can always leave the sleep mode of operation by resetting the H8/330 or by entering the hardware standby mode.

SPECIAL CONSIDERATIONS

RAM RETENTION

The H8/330 also offers the ability for the user to maintain the contents of the on-chip RAM and CPU registers with a low voltage input to the device.

During either of the standby modes (hardware or software) of operation, the user can drop his supply voltage to +2.0 volts DC and still be assured that the contents of the on-chip RAM will not be lost. To use this feature correctly, the user must ensure that he disables the on-chip RAM (by clearing the RAME bit in the SYSCR) just before entering the standby mode. While in the standby modes of operation, the user can now reduce his supply voltage (thus further reducing current consumption in his system). During the software standby mode of operation, the user cannot only maintain the RAM contents but also the contents of the CPU’s registers while the low voltage is applied.

Before releasing the H8/330 from either standby mode of operation, it is the responsibility of the user to ensure that the proper operating voltage (V_{CC}=+5.0V ±10%) be available.

EXTERNAL OSCILLATOR

In most systems (or microcontrollers), it is the oscillator that is the main concern when attempting to reduce power consumption. Though peripheral and CPU functions are stopped, since the oscillator continues to operate small power savings are observed. The H8/330 overcomes this concern by providing its own on-chip oscillator that is stopped during the standby mode of operation.

If your system uses an external oscillator to drive the H8/330 device and you still wish to enjoy the power consumption savings that the H8/330 offers, you still can. In instances such as this, the H8/330 would accept the external clock input and stop the internal clock from being provide to the on-chip peripherals during the power-down modes. Here the oscillator stabilization time (AC parameter t_{OSC2}) becomes effectively 0 ms. You can now program the Standby Timer Select bits in the SYSCR to “000” to reduce the delay when coming out of the software standby mode to its absolute minimum.

APPLICATION EXAMPLE

Software Standby Mode

In this example, we will use the NMI input to suggest when the H8/330 should be in a power-down state. Since the NMI input is high, we would like the H8/330 to continue normal operations. When the NMI input goes low, we want to enter the software standby mode. This is possible because we can sense both edges of the NMI input on the H8/330. For the sake of programming the Standby Timer Select bits, lets assume that the H8/330 is operating at a clock frequency of 6MHz. In discussion of the software, we will talk only about programming that is required and not discuss peripheral initialization at all (refer to Figure 4 for a flow chart of the operations sequence).

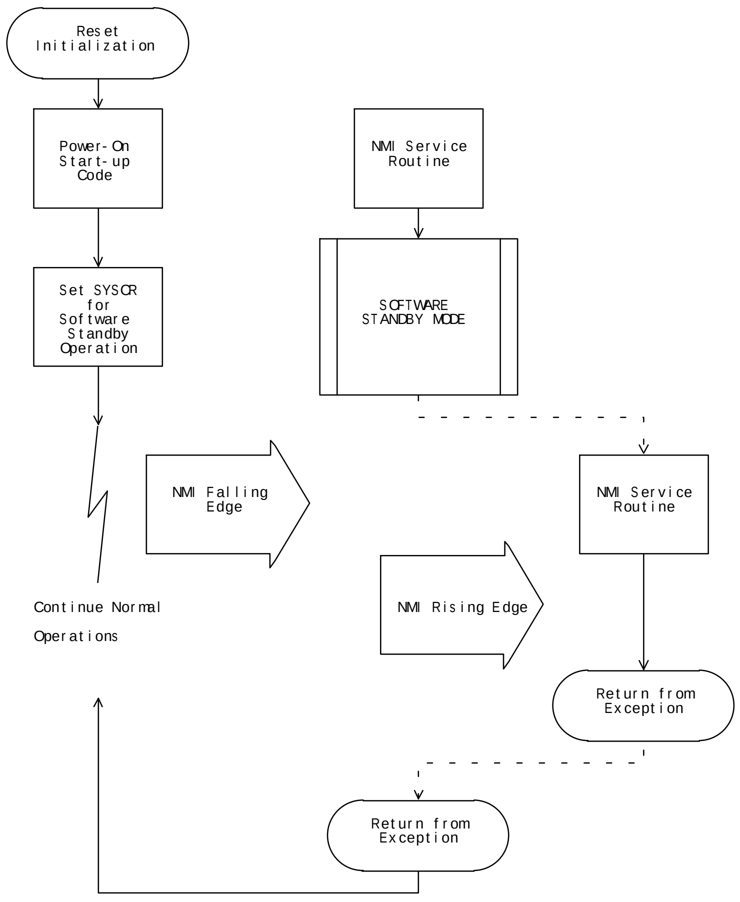

Figure 4: Application Example Processing Flowchart

During the normal operating sequence, the H8/330 would go through the process of initializing all its peripherals and other functions for normal operation. Since the System Control Register defaults to having the NMI edge selection for falling edge, no programming of that bit is necessary at this time. We will take this opportunity to program the SYSCR for the proper STS values. We know that the t_{OSC2} value is 10 msec from the AC characteristics of the H8/330. This calculates out to 60,000 t-states at 6MHz. To allow for this number of clock cycles, we must program STS_2-STS_0 to a value of "011." This will allows 10.9 msec to elapse for oscillator stabilization.

Whenever the falling edge of the NMI signal is recognized, the H8/330 will begin the processing of the NMI exception processing service routine. During this service routine we must do three basic operations; figure out whether we are going into or out of software standby mode, change the state of the NMI edge selection, and execute the SLEEP instruction (if we are going into the standby mode). Optionally we could also enable or disable the on-chip RAM if we were going to reduce voltage to the H8/330. After that we would return from this exception processing service routine to our normal operation (a flow chart of the NMI service routine is shown in Figure 5).

For our discussion of the software, please refer to Listing 1. In the main routine, the only thing we really need to do is to program the SYSCR with the values necessary for the NMI edge selection and the standby timer selection (for oscillator stabilization time). Initially we want to capture the falling edge of the NMI input and set the STS bits for a count of 65536. This requires the programming of "101110X1" into the SYSCR (refer to Figure 3 for a description). With this programmed into the SYSCR, we can continue with our normal processing.

Whenever the falling edge of the NMI signal is observed (see Figure 6), the H8/330 will begin processing the NMI exception processing service routine. Since this routine must handle both placing the H8/330 into the software standby mode as well as recovering from it, we must first decide which one it is. To do this we can test the state of the NMIEG bit. If this bit is a "0," then we can assume that we have detected the falling edge and that we are going to go into the software standby mode. Before we execute the SLEEP instruction we would need to program the NMIEG bit to "1" so that we can now monitor for the rising edge of the $\overline{\text{NMI}}$ signal. Optionally, if we are going to reduce the V_{CC} level we would need to clear the RAME bit in the SYSCR now before we execute the SLEEP instruction.

After executing the SLEEP instruction the H8/330 is now in the software standby mode of operation awaiting the input of the rising edge on the $\overline{\text{NMI}}$ signal. When the rising edge is detected (see Figure 7), the H8/330 starts the internal counter for the standby timer and delays further processing until the counter has timed out. At this point the H8/330 begins processing the NMI exception processing service routine again.

We still need to test the NMIEG bit to decide whether we are going into the standby mode or coming out of it. If this bit is a "1," then we can assume that we are coming out of the standby mode. Here, we would want to change the NMI edge selection from rising edge to falling edge. If we had disabled the on-chip RAM, we would want to make sure that we re-enabled it for use. Afterward we merely return from the NMI service routine (which incidentally returns us to the NMI service routine that we were performing to go into standby mode).

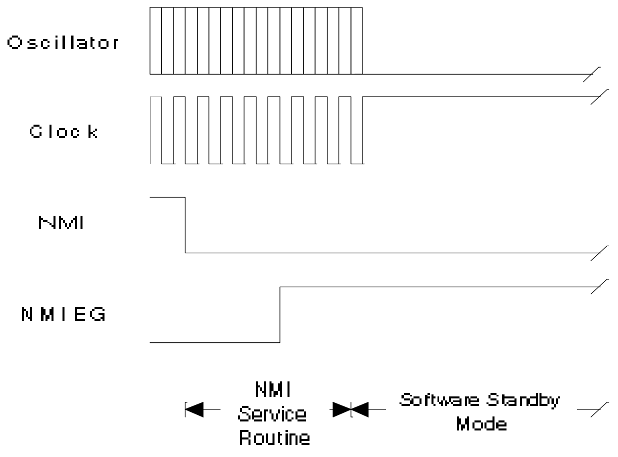

Figure 6: Application Example
NMI Falling Edge Timing

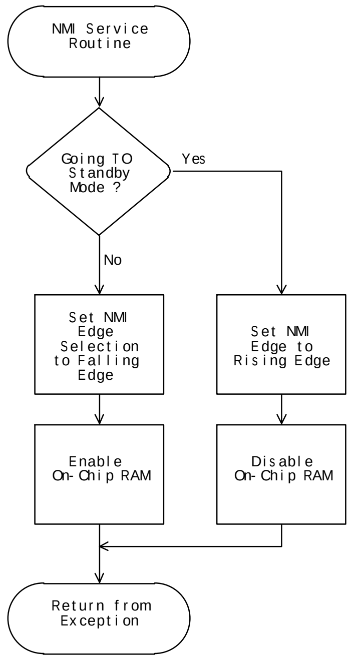

Figure 5: Application Software Flowchart
(NMI Service Routine)

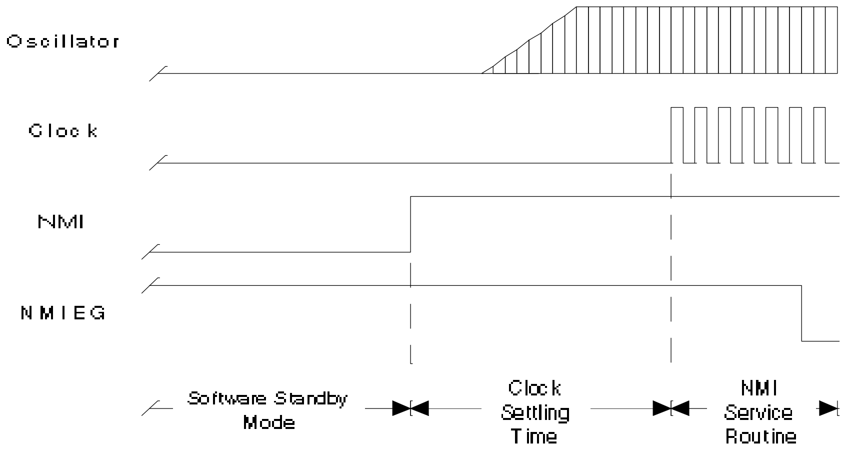

Figure 7: Application Example
NMI Rising Edge Timing

Listing 1: Application Example NMI Service Routine

```
;H8/330 Power-Down Application Example

;NMI Service Routine

nmi_service:
             btst.b        #2,@h'ffc4                ;test nmieg bit in SYSCR
             beq           falling_edge              ;going into power-down

;coming out of power-down
rising_edge:
             bclr.b        #2,@h'ffc4                ;set nmieg for falling edge
             bset.b        #0,@h'ffc4                ;enable on-chip RAM
             rte                                     ;return from processing
                                                     ; to previous NMI routine

;going into power-down
falling_edge:
             bset.b        #2,@h'ffc4                ;set nmieg for rising edge
             bclr.b        #0,@h'ffc4                ;disable on-chip RAM
             sleep                                   ;go to power-down mode
             rte                                     ;return from processing
                                                     ; to normal operation

             .end
```