August 28, 1996

Revision 0.1

REDUCED POWER OPTIONS FOR THE 80960HA/HD/HT PROCESSOR

ABSTRACT

Intel's 80960HA/HD/HT processor (abbreviated "Hx") does not support a low power "Halt" mode, contrary to some prior claims. However, some customers want to reduce power consumption and heat dissipation during idle times. The customer expectations have been set by the 80960JA/JF processor (abbreviated "Jx") that does offer a Halt mode that provides about 90% power reduction during idle times.

This paper explains three techniques to reduce the Hx processor power dissipation during idle times. The three techniques are:

1. Minimum power software loop
2. Reduced clock frequency
3. ONCE mode

The vertical bar in the left margin indicates new information since the last revision.

TOPICS COVERED IN THIS DOCUMENT

SUMMARY

These three options provide increasing levels of power savings at increasing costs. Costs are in terms of software and hardware complexity, entry and recovery times, and the ability to do useful work during the idle time.

Table 1 summarizes an example of the techniques for the 50 MHz 80960HD. Power dissipation is based on typical ICC Active (Thermal) measurements.

Table 1. Summary of Power Reduction Options for 80960HD-50 Processor (Operating at 50 MHz Internal Clock, 25 MHz Bus Clock)

OPTION	POWER DISSIPATION	ENTRY TIME	RECOVERY TIME	PROCESSING DURING IDLE	REQ'D MODS
Software Loop	2.8 W	2 µs	4 µs	No	Software
Slow clock	2.1 W	900 µs	900 µs	Yes	Hardware & software
ONCE mode	0.1 W	2 µs	400 µs	No	Hardware & software
Max Power	4.8 W				
Typical Power	3.0 W				

Power Saving vs. Cost

The highest power savings come at the highest cost in terms of system complexity. The "Power Dissipation" column of Table 1 shows the software loop to be the least effective power reduction, and the ONCE mode to be the most effective. The "Req'd Mods" column indicates the software loop requires only a software change whereas the slow clock and ONCE modes require combinations of software and hardware, implying board-level changes. See the technique description sections for the details.

Entry and Recovery Time

If your application requires fast entry and recovery time, the software loop technique offers the fastest of both. The ONCE mode offers the next fastest latencies, followed by the slow clock mode.

Useful Processing During Idle

Only the slow clock technique permits the processor to continue executing useful code during application idle times, albeit at a reduced clock speed. The other two techniques cannot execute anything useful during idle. Typical idle time processing could include low intensity I/O polling, keypad scanning, memory diagnostics, or thermal control.

You can combine the slow clock and software loop techniques to reduce idle power even further, but the processor would no longer be available for useful work during idle.

MINIMUM POWER SOFTWARE LOOP

Description

A small, factory-supplied software loop executes inside the processor, thereby preventing power-consuming external bus accesses. Furthermore, the software loop has been crafted to minimize on-chip bus transitions inside the processor.

An interrupt service routine (ISR) terminates the loop, returning the processor to useful work.

Table 2 shows the power savings you can expect using the software loop technique.

Table 2. Expected Power Savings for the Software Loop Technique (Values based on June 1996 data sheet release. Your results may vary.)

PRODUCT	CORE CLOCK	REDUCED POWER	TYPICAL POWER	PWR SAVINGS
80960HA	25 MHz	1.7 W	1.8 W	**9%**
	33 MHz	2.0 W	2.2 W	**9%**
	40 MHz	2.4 W	2.6 W	**8%**
80960HD	32 MHz	1.9 W	2.1 W	**8%**
	50 MHz	2.8 W	3.0 W	**8%**
	66 MHz	3.5 W	3.8 W	**8%**
80960HT	60 MHz	3.1 W	3.4 W	**8%**
	75 MHz	3.8 W	4.1 W	**8%**

The software loop is the least capable option to reduce power, but it is also least expensive. One customer has successfully used this technique already to reduce heat dissipation enough to stop the system cooling fan during idle times.

Table 3 shows the entry and recovery times for the software loop technique.

Required Modifications

Add the two assembly language routines shown in APPENDIX A - Code For Minimum Power Software Loop to your software. The first routine (“`sleep`”) is the procedure your program calls to invoke the minimum power software loop. The other function (“`wake_up`”) is the ISR which terminates the loop. Include that routine in the ISR of any interrupt intended to terminate the minimum power software loop.

Any interrupt source (either an internal timer or external XINT pins) can be designated to terminate the loop. Otherwise, no hardware modifications are required.

Table 3. Approximate Entry and Recovery Latency Times for the Software Loop Technique (Times assume fast interrupt detect mode, cached ISR, and adequate space in the register cache)

PRODUCT	CORE CLOCK	ENTRY TIME	RECOVERY TIME
80960HA	25 MHz	4 µs	6 µs
	33 MHz	3 µs	5 µs
	40 MHz	3 µs	5 µs
80960HD	32 MHz	3 µs	6 µs
	50 MHz	2 µs	4 µs
	66 MHz	2 µs	4 µs
80960HT	60 MHz	2 µs	4 µs
	75 MHz	1 µs	4 µs

Operation

Cache the ISRs intended to wake up the processor and reserve space in the Register Cache for the interrupt context switch. These preparations reduce the wake up interrupt latency. Save the contents of g14 somewhere (the software loop corrupts that register).

When your application requires power reduction, call the `sleep` function. No parameters are necessary. The processor will complete any queued bus requests then suspend all external activity and reduce its power consumption.

To resume normal operation, assert any interrupt configured to wake up the processor. The ISR will execute as usual. When the ISR runs its course and returns, the processor will branch out of the internal software loop, return from the `sleep` function, and resume normal operation.

Restore g14 if necessary.

Alternately, interrupts can be serviced without terminating the minimum power software loop provided the ISRs do not execute the `wake_up` routine and g14 is not corrupted.

Cost

Memory must absorb about 35 additional words of code. The instruction cache must accommodate the `wake_up` ISR`.` No useful processing is available during the idle time.

Restrictions

None.

REDUCED CLOCK FREQUENCY

Description

Power dissipation declines with declining clock frequency. The reduced clock frequency technique slows the system clock, thereby reducing power consumption of the processor and the entire system. As such, this technique may offer the most power savings, depending on your application.

The Hx processor includes an internal phase-locked loop (PLL) clocking circuit. Sudden, large changes in the input clock (CLKIN) frequency cause the PLL to lose lock, resulting in bus timing violations and likely system crashes. The Cypress CY2291 and CY2292 “Three-PLL Clock Generator” chips slew the frequency slowly enough for the Hx PLL to track without degrading the bus timings.

Digital input pins on the CY2291 and CY2292 clock generators select the clock frequency.

You can reduce the clock speed to any frequency above the processor minimum operating frequency. See the “Restrictions” section (page 7) for those frequency limits.

Table 4 shows the power savings of the Hx processor alone; system power dissipation will be reduced as well.

Table 4. Expected Power Savings for the Reduced Clock Frequency Technique. (Values based on June 1996 data sheet release. Your results may vary.)

PRODUCT	NOMINAL INPUT CLOCK	MINIMUM FREQ.	REDUCED POWER	TYPICAL POWER	PWR SAVINGS
80960HA	25 MHz	20 MHz	1.4 W	1.8 W	**39%**
	33 MHz	20 MHz		2.2 W	**51%**
	40 MHz	20 MHz		2.6 W	**58%**
80960HD	16 MHz	16 MHz	2.1 W	2.1 W	**0%** [1]
	25 MHz	16 MHz		3.0 W	**31%**
	33 MHz	16 MHz		3.8 W	**45%**
80960HT	20 MHz	16 MHz	2.7 W	3.4 W	**21%**
	25 MHz	16 MHz		4.1 W	**35%**

(1) The 32 MHz 80960HD processor already operates at the minimum permissible frequency and cannot be slowed any further.

Table 5 shows the entry and recovery times for the reduced clock frequency technique.

Table 5. Approximate Entry and Recovery Latencies for the Reduced Clock Frequency Technique.

PRODUCT	NOMINAL INPUT CLOCK	ENTRY TIME	RECOVERY TIME
80960HA	25 MHz	500 µs	500 µs
	33 MHz	1200 µs	1200 µs
	40 MHz	1900 µs	1900 µs
80960HD	16 MHz	n/a [1]	n/a [1]
	25 MHz	900 µs	900 µs
	33 MHz	1600 µs	1600 µs
80960HT	20 MHz	400 µs	400 µs
	25 MHz	900 µs	900 µs

(1) The 32 MHz 80960HD processor already operates at the minimum permissible frequency and cannot be slowed any further.

Useful processing can be done during reduced power times, albeit at a reduced clock frequency.

You can realize further power reductions by combining this reduced clock frequency with the software loop technique. Of course, doing so eliminates any useful processor work during idle periods.

Required Modifications

Decide what minimum clock frequency your application will use. See the clock frequency limitations under “Restrictions” on page 7.

Design your system using either a Cypress CY2291 or CY2292 “Three-PLL Clock Generator” chip. The CY2291 provides a 32.768 kHz output and a 24 kHz or 32 kHz “FLOPPYCLK” output. The CY2292 omits these extra outputs and comes in a smaller package. The clock frequencies (8 to 100 MHz) are custom programmed at the factory. APPENDIX B - Cypress Contact Information shows how to obtain the chip data sheet.

Map the clock generator frequency selection bits into memory somewhere. Include a data bus buffer for the frequency selection pins S[2:0]. Add chip select logic in the memory controller to enable the buffer.

See APPENDIX C - Reduced Clock Frequency Hardware Block Diagram for a drawing of these modifications.

If necessary, decouple the CPU speed from I/O port speeds with FIFO or other dual-port buffers. Asynchronous I/O channels such as network ports probably do not slow down with your application’s CPU clock. Data received during idle times could arrive too fast for the CPU to handle until the clock resumes normal speed. You may need to add buffering to capture and hold the data intact until the CPU can work it off. A similar scenario may be true for outgoing data, too.

DRAM controllers – specifically the refresh timers – may need modifications to insure reliable refreshes at the slower clock speeds.

Some simple software is required to control the clock generator. Determining the system clock frequency dynamically may require more innovative software and perhaps hardware such as an external timer whose clock speed does not change with the processor clock.

Operation

When the application calls for power reduction, reduce the clock frequency by writing the Cypress-defined code to the S[2:0] clock select pins. The system clock frequency will transition to the new frequency.

Use an interrupt or poll an external I/O port or timer to signal a return to normal clock speed.

Write the code to the S[2:0] clock select pins to resume normal clock speed.

There is no digital indication on the clock generator to show when the clock generator reaches a particular frequency. Optionally, a timer or application-specific behavior may be able to indicate when normal clock speeds have been resumed.

Cost

Existing board designs, layouts, and routings must be revised. More hardware is needed to interface the Cypress frequency select pins to the data bus. More hardware may be required to decouple CPU and I/O speeds from one another.

This technique requires the longest entry and recovery latencies of any proposed in this white paper. However, useful processor work can be done while the clock generator transitions between frequencies.

Restrictions

The maximum allowable clock ramp rate is no problem; Cypress specifies an adequately slow transition rate.

The entire system must be able to work at the selected slow and fast clock frequencies.

The 32 MHz (16 MHz input clock) version of the 80960HD processor cannot benefit from this technique because the processor cannot operate below 16 MHz.

All A-step versions of the 80960HA/HD/HT processor can operate down to 16 MHz input clock. The B-step versions of the 80960HD and 80960HT can also operate down to 16 MHz input clock. However, the B-step 80960HA can operate down to only 20 MHz according to the Hx Specification Update (errata) sheet. These minimum frequency limits are reflected in the power savings and latency tables.

ONCE MODE

Description

Freezing the processor and its internal clocks in stasis offers the most power savings possible short of shutting off VCC. The Hx processor features an On-Circuit Emulator (“ONCE”, pronounced “ahns”) mode that completely disables the processor, its internal clocks, and floats the output pins. ONCE mode effectively removes the Hx processor from the socket, which was originally designed to allow an external emulator probe to assume complete control of the processor signals.

The processor must be reset to enter ONCE mode, then reset again and run through the normal boot-up sequence to recover. ONCE mode is invoked by asserting the ONCE# pin while RESET# is asserted and for at least one clock cycle after RESET# pin deasserts.

Table 6 shows the power savings of the ONCE mode.

Table 6. Expected Power Savings for the ONCE Mode Technique. (Values based on June 1996 data sheet release.)

PRODUCT	NOMINAL INPUT CLOCK	REDUCED POWER	TYPICAL POWER	PWR SAVINGS
80960HA	25 MHz	0.1 W (all speeds)	1.8 W	**95%**
	33 MHz		2.2 W	**96%**
	40 MHz		2.6 W	**97%**
80960HD	16 MHz		2.1 W	**96%**
	25 MHz		3.0 W	**97%**
	33 MHz		3.8 W	**98%**
80960HT	20 MHz		3.4 W	**98%**
	25 MHz		4.1 W	**98%**

Table 7 shows the entry and recovery times for the ONCE mode technique.

Table 7. Approximate Entry and Recovery Latencies for the ONCE Mode Technique. (Assumes 1000 clock cycles of user boot code following processor initialization.)

PRODUCT	NOMINAL INPUT CLOCK	ENTRY TIME	RECOVERY TIME
80960HA	25 MHz	4 µS	500 µS
	33 MHz	3 µS	400 µS
	40 MHz	3 µS	300 µS
80960HD	16 MHz	3 µS	750 µS
	25 MHz	2 µS	400 µS
	33 MHz	2 µS	300 µS
80960HT	20 MHz	2 µS	600 µS
	25 MHz	1 µS	300 µS

The 10,000 clock cycles required to power up and stabilize the PLL dominate the recovery latency times. The recovery latency numbers have been rounded conservatively; your user boot-up code will determine your exact latency.

The processor can do no useful work while in ONCE mode. However, a secondary bus master (such as a DMA controller) can use the system resources without interference from the processor.

Required Modifications

Design an external ONCE mode controller which consists of...

- an addressable port from the processor to invoke ONCE mode,
- a state machine to generate the ONCE and reset sequences,
- a large (10,000 clock cycle) timer to allow the PLL to stabilize during recovery, and
- an input pin for an external interrupt or timer to initiate recovery.

The addressable port can be a one-bit register that invokes ONCE mode when set. A simpler solution is to eliminate the register and invoke ONCE when the processor writes to a particular address range.

For example, decode the upper nibble of the address bus and the ADS# and W/R# signals. Invoke ONCE when the processor writes to memory Region 15 (0xFXXXXXXX). Typically, read-only ROMs or EPROMs reside in Region 15, so writing to that region would be unique to the ONCE controller. If read/write memory like FLASH or EEPROMs are used in that region, use another read-only or unused memory region.

The state machine can fit into a common 22V10 programmable logic device (PLD) or equivalent. APPENDIX D - ONCE Mode Design outlines the high level design.

A timer based on a resistor-capacitor (RC) network, either something like a common power-on reset circuit or a one-shot timer using the 555 timer chip, can provide the 10,000 clock cycle delay. A digital timer capable of counting 10,000 clock cycles would be prohibitively expensive.

An event must trigger the ONCE controller to restore the processor. An interrupt or hardware timer signal must be qualified by a low pass filter (LPF) to ignore spurious noise. A software LPF can be added to the ONCE controller to require a continuously valid input signal for several clocks before being recognized.

All processor output pins float (hi-Z) during ONCE mode. Pull the ADS# and DEN# signals high to prevent sporadic memory accesses. Optionally, pull the LOCK#, HOLDA, and FAIL# signals inactive (as well as any others necessary for your application) to prevent unpredictable system behavior. Include any pull-ups or pull-downs in the load analysis for these pins.

Construct the software such that the entire processing state can be stored before invoking ONCE mode. The software must know where to resume execution after recovery, which is not a trivial undertaking. Note that the re-boot recovery does not necessarily have to decompress memory images or copy code and tables from ROM to RAM if the RAM image has been preserved intact during ONCE mode.

Operation

Save the processing state in RAM memory. Write to the controller to invoke ONCE mode. The processor power reduces within a few clock cycles.

An external event or timer signals the controller to recover from ONCE mode. The controller resets the processor. Automatic processor initialization proceeds as usual. Following initialization, user software recognizes that an ONCE mode recovery is in progress and restores the processing state from memory.Normal execution resumes.

Cost

The ONCE mode technique is the most expensive and complex of any in this white paper. Existing logic designs, board layouts, and routings must be revised. An external ONCE controller must be added to execute the ONCE mode entry and recovery sequences. That controller logic must be designed. APPENDIX D - ONCE Mode Design offers an example guide.

This technique imposes the longest unproductive recovery latency since no useful work can be done while the processor is recovering.

Extensive software modifications are required to restore the processing state after recovery from ONCE mode.

Restrictions

The system must tolerate floating processor output pins during ONCE mode. Any signals that could disrupt the system by floating must be actively driven or passively pulled to a safe level. For example, if ADS# floated low then high again, the memory system could access a random address. Similarly, if DEN# floated low, data bus drivers could be enabled unnecessarily.

The system must provide RAM memory to store the software state during ONCE mode. The RAM must be incorruptible while the processor is in ONCE mode.

The CPU RESET# signal is separate from system RESET# signal. The external controller must reset the processor independent of the rest of the system.

APPENDIX A - CODE FOR MINIMUM POWER SOFTWARE LOOP

The code for the minimum power software loop is written in ASM960 and comes in two parts:

1. `‘sleep’` - the procedure to invoke the software loop
2. `‘wake_up’` - the Interrupt Service Routine (ISR) to terminate the software loop

An electronic version of this code is available through your Intel sales representative.

```
############################################################################
#                                                                          #
# Function:  sleep                                 Date: 3/13/96           #
#                                                                          #
# Author:  Intel Corporation                                               #
#                                                                          #
# Disclaimer:  Intel provides this AS IS, WITHOUT ANY WARRANTY,            #
#              INCLUDING THE WARRANTY OF MERCHANTABILITY OR                #
#              FITNESS FOR A PARTICULAR PURPOSE, and makes no              #
#              guarantee or representations regarding the use              #
#              of, or the results of the use of, the software              #
#              and documentation in terms of correctness,                  #
#              accuracy, reliability, currentness, or                      #
#              otherwise, and you rely on the software,                    #
#              documentation, and results solely at your own               #
#              risk.                                                       #
#                                                                          #
# Overview:                                                                #
# This code reduces power consumption during idle periods by               #
# preventing external bus accesses and I_cache accesses.  A small          #
# loop executes from the Instruction Queue inside the processor.           #
# No useful processing is done during idle.                                #
#                                                                          #
# Use the companion Interrupt Service Routine "wake_up" to exit this       #
# idle loop and resume normal operation.                                   #
#                                                                          #
# Use:                                                                     #
# Call this function to invoke reduced idle power dissipation.             #
# Execute Interrupt Service Routine "wake_up" to resume processing.        #
#                                                                          #
# Note:                                                                    #
# If g14 is not available in your application, you can use g7, g11,        #
# or g13 without affecting the power dissipation.  Otherwise, any          #
# modification of this code will increase the power dissipation.           #
#                                                                          #
# Resources Modified:  g14, I_cache enabled (then restored to former       #
#                      condition before return)                            #
#                                                                          #
# Example Supports:  80960HA/HD/HT                                         #
#                                                                          #
# Revision History:                                                        #
#-----------------------------------------------------------------------   #
############################################################################


sleep:
            #enable I_cache
            mov     sf2, r4
            mov     0, sf2          #sf2 = CCON

            #initialize g14
            mov     0x1f, g14

.align 4

            b       loop
            .word   0xFFFFFFFF      #fill the Instruction Queue with "1's"
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF

breakloop:  b       endloop

loop:       cmpis   g14, 0x1f
            ble     loop            #loop until g14 > 0x1f
            bo      breakloop
            .word   0xFFFFFFFF      #fill the Instruction Queue with "1's"
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF
            .word   0xFFFFFFFF

            #restore the I_cache
endloop:    mov r4, sf2             #sf2 = CCON
            syncf

            ret
```

```
############################################################################
#                                                                          #
# ISR:  wake_up                                  Date: 3/13/96             #
#                                                                          #
# Author:  Intel Corporation                                               #
#                                                                          #
# Disclaimer:  Intel provides this AS IS, WITHOUT ANY WARRANTY,            #
#              INCLUDING THE WARRANTY OF MERCHANTABILITY OR                #
#              FITNESS FOR A PARTICULAR PURPOSE, and makes no              #
#              guarantee or representations regarding the use              #
#              of, or the results of the use of, the software              #
#              and documentation in terms of correctness,                  #
#              accuracy, reliability, currentness, or                      #
#              otherwise, and you rely on the software,                    #
#              documentation, and results solely at your own               #
#              risk.                                                       #
#                                                                          #
# Overview:                                                                #
# This Interrupt Service Routine (ISR) inserts a value into register       #
# g14, causing the "sleep" function to exit its loop and resume            #
# normal processing.                                                       #
#                                                                          #
# Use:                                                                     #
# Vector to this ISR when an interrupt occurs that you intend to wake      #
# up the processor from it's idle loop.                                    #
#                                                                          #
# Optionally, add this code to the end of any ISR you intend to wake       #
# the processor.  Any ISR that leaves g14 greater than 0x1F will           #
# force the processor to break out of the idle loop.                       #
#                                                                          #
# Resources Modified:  g14                                                 #
#                                                                          #
# Example Supports:  80960HA/HD/HT                                         #
#                                                                          #
# Revision History:                                                        #
#------------------------------------------------------------------------  #
############################################################################


wake_up:
            lda        0x20, r4           #any value > 0x1f
            mov        r4, g14
            ret
```

APPENDIX B - CYPRESS CONTACT INFORMATION

Cypress Semiconductor Corporation
3901 North First Street
San Jose, CA 95134
USA
(408) 943-2600
(408) 943-6822 - FAX

World Wide Web Homepage:

www.cypress.com

World Wide Web Data Sheet:

www.cypress.com:80/cypress/prodgate/timi/cy2291.html

World Wide Web Sales Contacts:

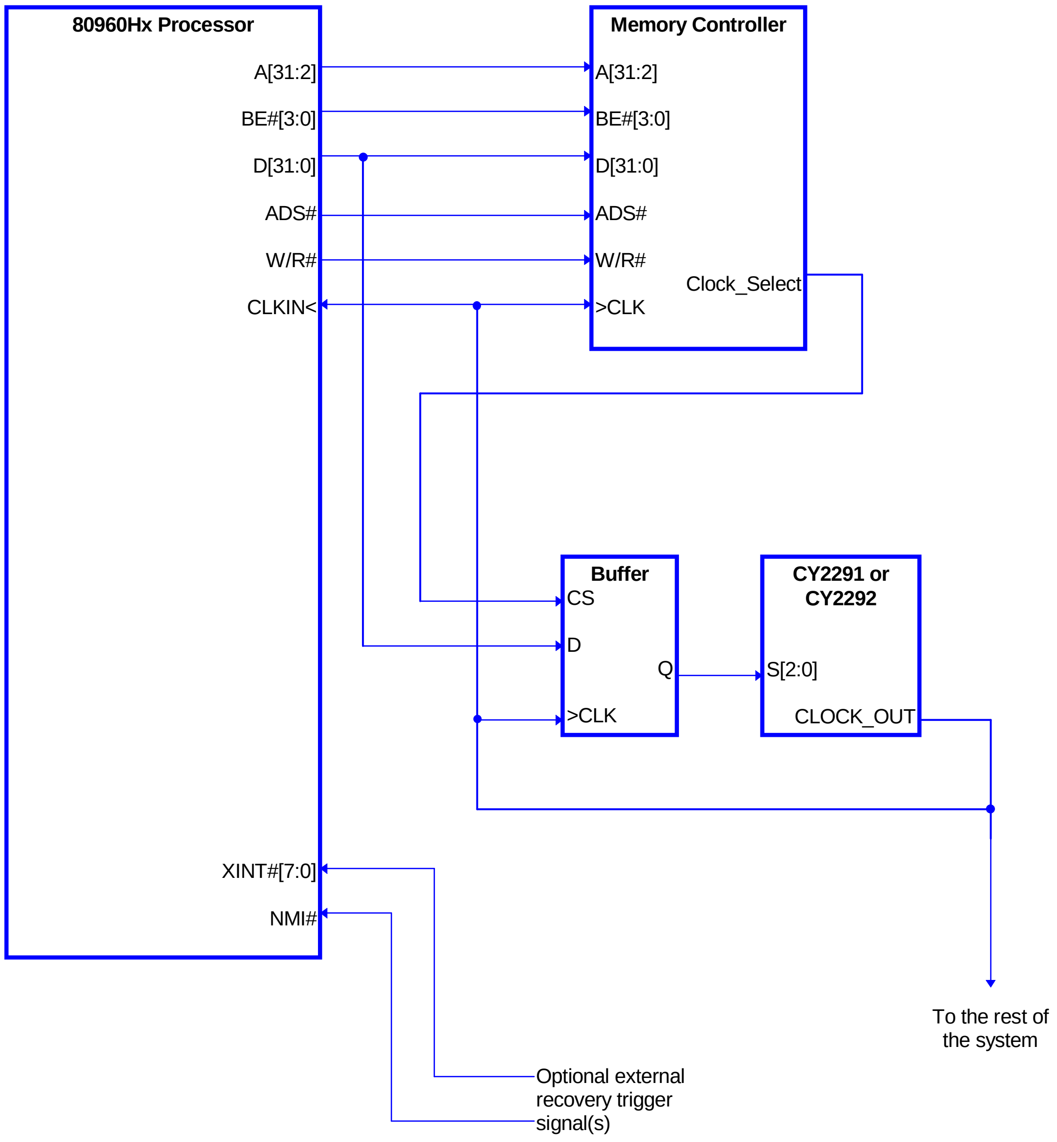
www.cypress.com:80/cypress/ww_sales/sale_top.htm

CY2291 / CY2292 Three-PLL Clock Generator Data Sheet Document #: 38-00410

(Cypress information verified 8/12/96)

APPENDIX C - REDUCED CLOCK FREQUENCY HARDWARE BLOCK DIAGRAM

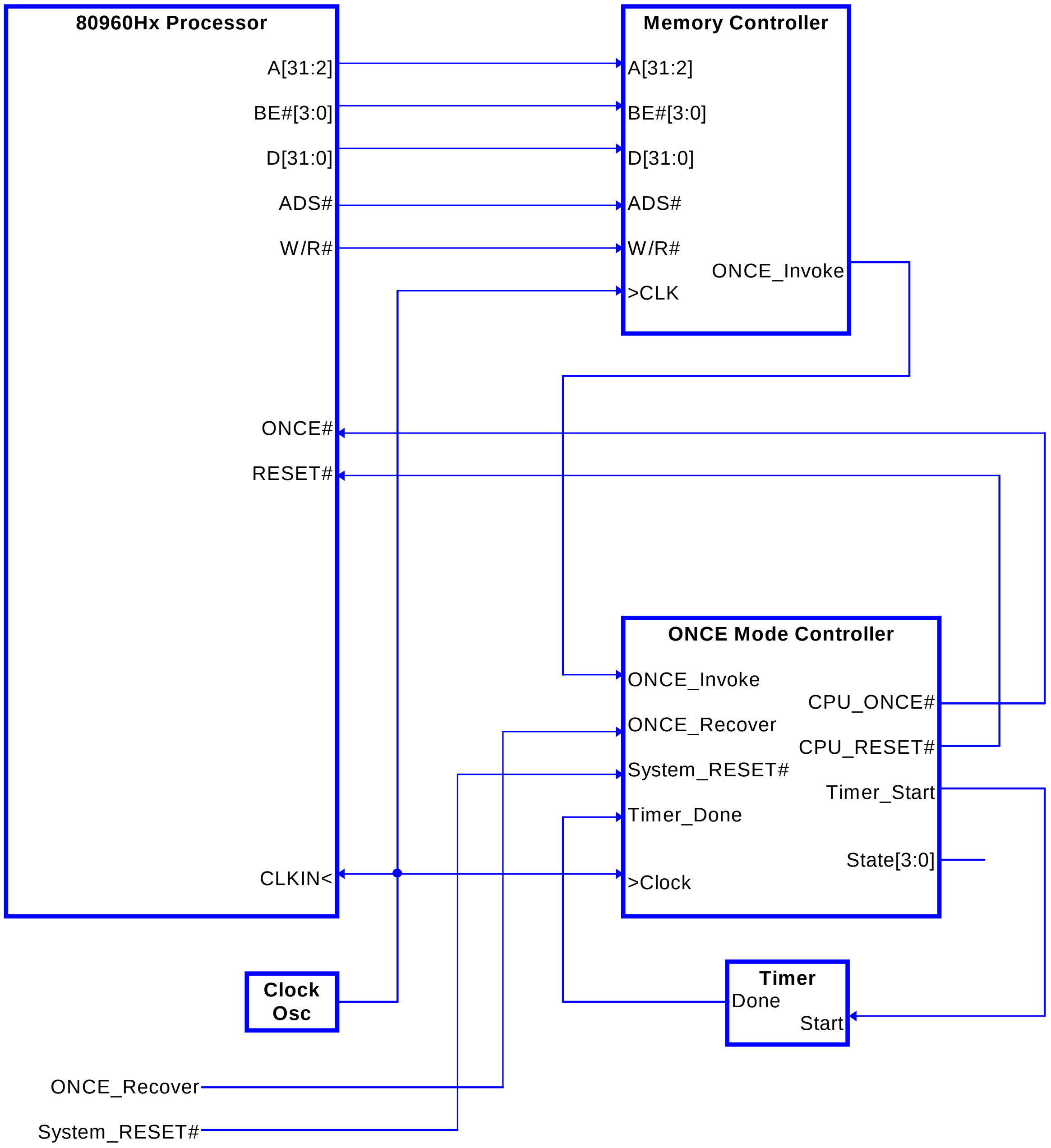
The necessary hardware changes for the reduced clock frequency technique are depicted below.



APPENDIX D - ONCE MODE DESIGN OVERVIEW

Block Diagram of System Using ONCE Mode Controller

An application using the ONCE mode power reduction technique involves the addition of a ONCE mode controller, a 10,000 clock cycle timer, and interface to the 80960 processor and memory controller. Also, the system reset signal and an ONCE_Recover signal are included.



Input and Output Signals

The ONCE controller described here uses five inputs and seven outputs, compatible with a 22V10 PLD.

Inputs

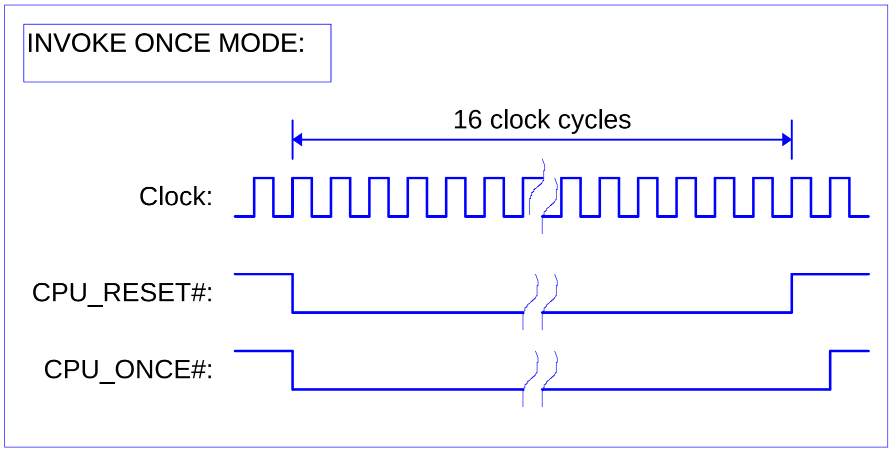
SIGNAL NAME	DESCRIPTION
ONCE_Invoke	Command to invoke ONCE mode
ONCE_Recover	Command to recover from ONCE mode
System_RESET#	Master RESET# signal on the board
Timer_Done	Report from timer that its delay has elapsed
Clock	System clock

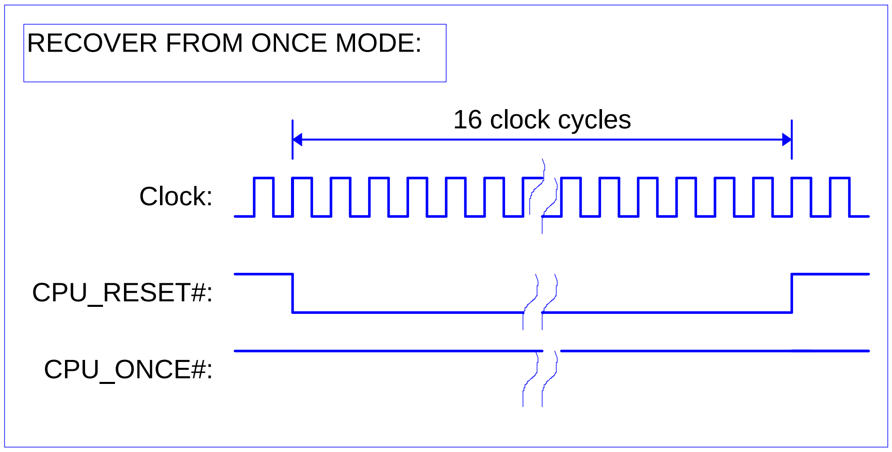
Outputs

SIGNAL NAME	DESCRIPTION
CPU_ONCE#	Controls 80960Hx processor ONCE# pin
CPU_RESET#	Controls 80960Hx processor RESET# pin
Timer_Start	Command to start the timer delay; expect response on Timer_Done
State[3:0]	Moore model state counter bits

Waveforms

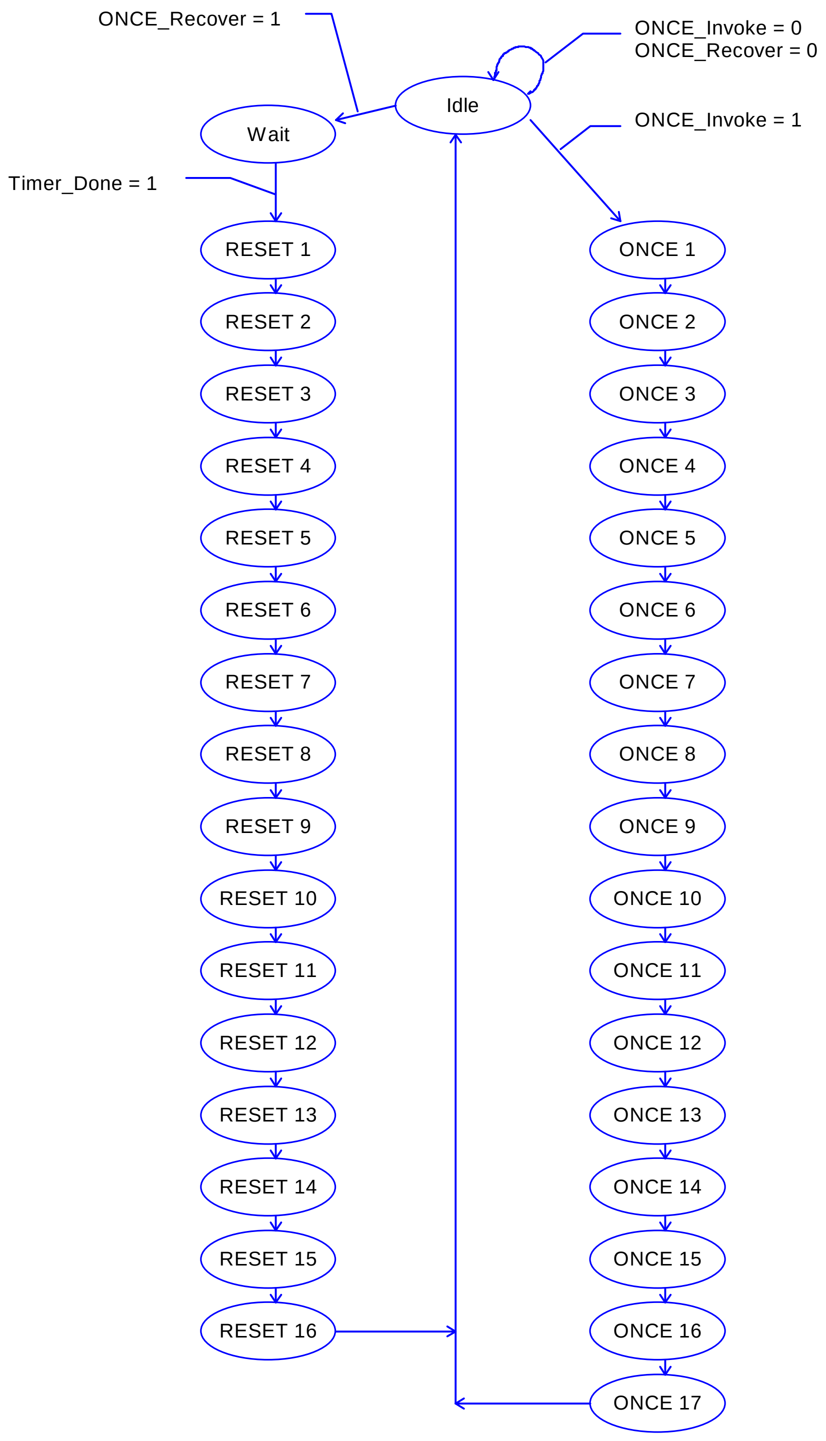
The ONCE controller must produce the following waveforms to invoke and recover from ONCE mode.





State Machine Description

State Bubble Diagram



State Diagram Definitions

NAME	CPU_RESET#	CPU_ONCE#	Timer_Start	State[3:0]
Idle	high	high	low	0
Wait	low	high	high	0
ONCE 1	low	low	low	0
ONCE 2	low	low	low	1
ONCE 3	low	low	low	2
ONCE 4	low	low	low	3
ONCE 5	low	low	low	4
ONCE 6	low	low	low	5
ONCE 7	low	low	low	6
ONCE 8	low	low	low	7
ONCE 9	low	low	low	8
ONCE 10	low	low	low	9
ONCE 11	low	low	low	10
ONCE 12	low	low	low	11
ONCE 13	low	low	low	12
ONCE 14	low	low	low	13
ONCE 15	low	low	low	14
ONCE 16	low	low	low	15
ONCE 17	high	low	low	0
RESET 1	low	high	low	0
RESET 2	low	high	low	1
RESET 3	low	high	low	2
RESET 4	low	high	low	3
RESET 5	low	high	low	4
RESET 6	low	high	low	5
RESET 7	low	high	low	6
RESET 8	low	high	low	7
RESET 9	low	high	low	8
RESET 10	low	high	low	9
RESET 11	low	high	low	10
RESET 12	low	high	low	11
RESET 13	low	high	low	12
RESET 14	low	high	low	13
RESET 15	low	high	low	14
RESET 16	low	high	low	15

Logic Pseudo Code

```
“Inputs
       ONCE_Invoke;
       ONCE_Recover;
       System_RESET#;
       Timer_Done;
       clock;

“Outputs
       CPU_ONCE#     istype ‘buffer, reg_D’;
       CPU_RESET#    istype ‘buffer, reg_D’;
       Timer_Start   istype ‘buffer, reg_D’;
       State[3:0]    istype ‘buffer, reg_D’;

“State register assignment
       (see STATE DEFINITIONS table)

Equations
       CPU_RESET#    = System_RESET; ”reset CPU when the system resets

state_diagram

state Idle:          if ONCE_Recover then Wait  else
                     if ONCE_Invoke then ONCE_1  else
                     Idle;
state Wait:          if Timer_Done then RESET_1  else Wait;
state RESET_1:       RESET_2;
state RESET_2:       RESET_3;
state RESET_3:       RESET_4;
state RESET_4:       RESET_5;
state RESET_5:       RESET_6;
state RESET_6:       RESET_7;
state RESET_7:       RESET_8;
state RESET_8:       RESET_9;
state RESET_9:       RESET_10;
state RESET_10:      RESET_11;
state RESET_11:      RESET_12;
state RESET_12:      RESET_13;
state RESET_13:      RESET_14;
state RESET_14:      RESET_15;
state RESET_15:      RESET_16;
state RESET_16:      Idle;

state ONCE_1:        ONCE_2;
state ONCE_2:        ONCE_3;
state ONCE_3:        ONCE_4;
state ONCE_4:        ONCE_5;
state ONCE_5:        ONCE_6;
state ONCE_6:        ONCE_7;
state ONCE_7:        ONCE_8;
state ONCE_8:        ONCE_9;
state ONCE_9:        ONCE_10;
state ONCE_10:       ONCE_11;
state ONCE_11:       ONCE_12;
state ONCE_12:       ONCE_13;
state ONCE_13:       ONCE_14;
state ONCE_14:       ONCE_15;
state ONCE_15:       ONCE_16;
state ONCE_16:       ONCE_17;
state ONCE_17:       Idle;

end
```

REFERENCES

The following documents provide more information on the i960 Hx microprocessor family.

1. *i960® Hx Microprocessor User's Manual*, Intel order number 272484.
2. *80960HA/HD/HT Embedded 32-bit Microprocessor Advance Data Sheet*, order number 272495. Also available in the *i960® Processors and Related Products* handbook, Intel order number 272084.