



80960RM/RN Processor Initialization: Programming Guide & Initialization Code

Application Note

August 1998





Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

The 80960 RM/RN may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Copyright © Intel Corporation, 1998

*Third-party brands and names are the property of their respective owners.

Contents

1.0	Introduction.....	5
1.1	References	5
2.0	Initialization Modes.....	5
2.1	Mode 0 Initialization.....	6
2.2	Mode 1 and Mode 2	6
2.3	Mode 3 Initialization (Default Mode).....	6
2.4	Core Initialization.....	7
2.4.1	IBR Fetching.....	7
2.4.2	Internal Initialization Processing.....	7
3.0	Programming Differences between the 80960RP/RD and the 80960RM/RN.....	8
3.1	Programming Differences.....	9
3.2	New Registers/Changed Registers	9
4.0	Recommended Register Settings.....	14
5.0	Initialization Code	15
5.1	init.s	16
5.2	rm.s	20
5.3	main.c.....	21
5.4	evrm_hw.c.....	21
5.5	rm_ibr.c	24
5.6	rm_init.ld.....	25
5.7	evrm.h	26
5.8	bld.bat.....	32

Figures

1	Internal Memory Map for Sample Initialization Code.....	16
---	---	----

Tables

1	80960RM/RN Initialization Modes	6
2	Address/Data Bus for an IBR fetch from an 8-Bit Region	7
3	Processor Fetching Between the IBR and User Code	8

1.0 Introduction

This document describes the initialization flow of the 80960RM/RN's core processor, recommended initialization code, and initialization and memory-mapped register differences from the 80960RP/RD processors. It is intended to supplement the initialization information found in the user's manual. For static configuration bits, default settings are recommended that mirror Intel's internal validation settings. Users may find this paper helpful in the early stages of software/hardware development for an 80960RM/RN application.

1.1 References

Functional descriptions for the 80960RM/RN product can be found in:

i960[®] RM/RN I/O Processor Developer's Manual, Intel order # 273158-001

Electrical specifications for the 80960RM/RN product are found in:

80960RM I/O Processor Data Sheet, Intel order # 273156-001

80960RN I/O Processor Data Sheet, Intel order # 273157-001

Hardware design specifications for the 80960RM/RN product are found in:

i960[®] RM/RN I/O Processor Design Guide, Intel order # 273139-001

2.0 Initialization Modes

The 80960RM/RN processors contain four different initialization modes, as detailed in [Table 1](#). The initialization mode is determined by the level of two pins, **RST_MODE#** and **RETRY**, when **P_RST#** is asserted.

Table 1. 80960RM/RN Initialization Modes

Initialization Mode	RST_MODE #	P_RST#	Primary PCI Interface Action	i960 Core Processor Action	Next Step
Mode 0 (Recommended for Motherboard Usage Only)	0	0	Accepts Transactions	Held in Reset	Host processor must clear the Core Processor Reset bit in the EBCR to allow the Core Processor to execute the initialization code.

Table 1. 80960RM/RN Initialization Modes

Mode 1 (Not Recommended)	0	1	Retries all Configuration Cycle Transactions	Held in Reset	An external agent must clear the Core Processor Reset bit in the EBCR to allow the Core Processor to execute the initialization code, or clear the Configuration Cycle Retry bit in the EBCR to allow the Primary PCI Interface to accept configuration transactions.
Mode 2 (Not Recommended)	1	0	Accepts Transactions	Initializes/Executes 80960RM/RN Initialization Code	N/A
Mode 3 (default)	1	1	Retries all Configuration Cycle Transactions	Initializes/Executes 80960RM/RN Initialization Code	The Core Processor must clear the Configuration Cycle Retry bit in the EBCR to allow the Primary PCI Interface to accept configuration transactions.

2.1 Mode 0 Initialization

The usage of Mode 0 is recommended for motherboard designs only. This mode is sometimes known as “Bridge Mode”, since the 80960RM/RN will function as a bridge-only with the core held in reset.

2.2 Mode 1 and Mode 2

Intel does not recommend the use of Mode 1 and Mode 2.

2.3 Mode 3 Initialization (Default Mode)

The Core Processor must clear the Configuration Cycle Retry bit in the EBCR to allow the Primary PCI Interface to accept configuration transactions. [Section 2.4, “Core Initialization” on page 6](#) describes what occurs when booting in mode 3, and [Section 5.0, “Initialization Code” on page 15](#) contains sample code that can be used when booting in mode 3.

2.4 Core Initialization

2.4.1 IBR Fetching

When the 80960RM/RN is brought out of reset, it first executes internal self-test (if the user has it enabled based on the level of the **STEST** pin at reset). After passing self-test, the processor runs an external bus confidence test, which checks external bus functionality. This test reads eight words from the Initialization Boot Record (IBR) and performs a checksum on the words (for more detailed information on checksum see Chapter 11 in the *i960[®] Rx Microprocessor User’s Manual*).

On the 80960RM/RN the IBR is located at 0xFEFFFF30. The processor starts by fetching the third and fourth words of the IBR. The low order bytes of these two words gives the processor information about the IBR region bus width and whether it is big or little endian:

- The 80960RM/RN requires all accesses to be 32-bits wide. Make sure that the data at address 0xFEFFFF38 in the IBR is set to 0x00000080.
- The 80960RM/RN requires all accesses to be little endian. Make sure that the data at address 0xFEFFFF3C in the IBR is set to 0x00000000.

Once these two words are fetched, the processor uses their values to configure the IBR memory region (PMCON14_15, 0xE0000000 - 0xFFFFFFFF). The rest of the IBR fetch will use the new memory configuration.

Table 2 indicates what the user should see on the bus during the IBR fetch of 80960RM/RN processors.

Table 2. Address/Data Bus for an IBR fetch from an 8-Bit Region

Address on Bus	Data	Access Type
0xFEFFFF38	PMCON14_15, byte 2	1 word
0xFEFFFF3C	PMCON14_15, byte 3	1 word
0xFEFFFF5C	Bus Confidence Self-Test Check Word 5	1 word
0xFEFFFF58	Bus Confidence Self-Test Check Word 4	1 word
0xFEFFFF54	Bus Confidence Self-Test Check Word 3	1 word
0xFEFFFF50	Bus Confidence Self-Test Check Word 2	1 word
0xFEFFFF4C	Bus Confidence Self-Test Check Word 1	1 word
0xFEFFFF48	Bus Confidence Self-Test Check Word 0	1 word
0xFEFFFF44	PRCB Pointer	1 word
0xFEFFFF40	First Instruction Pointer (IP)	1 word
0xFEFFFF44	PRCB Pointer	1 word

2.4.2 Internal Initialization Processing

Once an 80960RM/RN processor finishes fetching the IBR (assuming that it passed the external bus confidence test), it uses pointers from the IBR to fetch data from the control table, interrupt table, etc., and cache it internally. Table 3 details the data being fetched.

Table 3. Processor Fetching Between the IBR and User Code (Sheet 1 of 2)

Relative Memory Address	Data	Data Size
prcb_ptr + 0x0	Fault Table Base Address	1 word
prcb_ptr + 0x4	Control Table Base Address	1 word
prcb_ptr + 0x8	AC Register Initial Image	1 word
prcb_ptr + 0xC	Fault Configuration Word	1 word
prcb_ptr + 0x10	Interrupt Table Base Address	1 word
prcb_ptr + 0x14	System Procedure Table Base Address	1 word
prcb_ptr + 0x1C	Interrupt Stack Pointer	1 word

Table 3. Processor Fetching Between the IBR and User Code (Sheet 2 of 2)

prcb_ptr + 0x20	Instruction Cache Configuration Word	1 word
int_tbl_ptr + 0x3E4	NMI Vector #	1 word
sys_proc_tbl_ptr + 0xC	Supervisor Stack Pointer Base	1 word
prcb_ptr + 0x24	Register Cache Configuration Word	1 byte
ctrl_tbl_ptr + 0x10	IMAP0	1 word
ctrl_tbl_ptr + 0x14	IMAP1	1 word
ctrl_tbl_ptr + 0x18	IMAP2	1 word
ctrl_tbl_ptr + 0x1C	ICON	1 word
ctrl_tbl_ptr + 0x20	PMCON0_1	1 word
ctrl_tbl_ptr + 0x28	PMCON2_3	1 word
ctrl_tbl_ptr + 0x30	PMCON4_5	1 word
ctrl_tbl_ptr + 0x38	PMCON6_7	1 word
ctrl_tbl_ptr + 0x40	PMCON8_9	1 word
ctrl_tbl_ptr + 0x48	PMCON10_11	1 word
ctrl_tbl_ptr + 0x50	PMCON12_13	1 word
ctrl_tbl_ptr + 0x58	PMCON14_15	1 word
ctrl_tbl_ptr + 0x68	Trace Controls (TC)	1 word
ctrl_tbl_ptr + 0x6C	Bus Configuration Control (BCON)	1 word
IP	Start Fetching User Code	1 word

Once the 80960RM/RN finishes fetching the pertinent information from the Initial Memory Image (IMI), it uses the Instruction Pointer (IP) from the IBR to start fetching user code. Processor-initiated initialization is complete at this point. The user code is then required to execute the rest of the initialization as outlined in [Section 5.0, “Initialization Code” on page 15](#).

3.0 Programming Differences Between the 80960RP/RD and the 80960RM/RN

The 80960RM/RN is very similar to the 80960RP/RD in many respects, however there were quite a few changes and additions made to the 80960RM/RN. This section lists many of the changes that programmers will need to take into account, not only in their initialization code, but in their application code as well.

3.1 Programming Differences

Some of the differences between the 80960RP/RD and the 80960RM/RN that programmers will need to take note of in their code are listed below. Note that other differences may exist and the programmer should be sure to fully review the *i960[®] RM/RN I/O Processor Developer’s Manual*.

- The flash/ROM memory controller channels 0 & 1 are restricted to being 8-bits wide on the 80960RM/RN. The 80960RP/RD supports 32-, 16-, and 8-bit wide flash/ROM. The flash bus can still support a UART and an LED.

- All of the PMCON's on the 80960RM/RN must be programmed to support 32-bit wide data regions. Thus IBR accesses from the core will be issued as words and the memory controller will handle converting them to byte accesses to the flash and then back to words on the internal bus.
- The 80960RM/RN supports only SDRAM, which must be programmed differently than the memory supported by the 80960RP/RD.
- The 80960RM/RN has a larger data cache and instruction cache than the 80960RP/RD. Therefore the action of cache controlling instructions (*icctl*, *dcctl*, *sysctl*) will be different.
- The internal bus is no longer an 80960 local bus, it is a PCI-like bus. It cannot be seen external to the chip. Use the memory controller bus to debug initialization code.
- On the 80960RM/RN the direct addressing window can be mapped at either 0x00002000 - 0x7FFFFFFF or in the upper 2Gbytes at 0x80002000 - 0xFFFFFFFF by setting a bit in the ATUCR.
- On the 80960RP/RD the primary ATU had the first 4K of its memory region directed to the messaging unit. The 80960RM/RN has added a bit in the ATUCR that will allow any access on the secondary PCI bus falling in the first 4K of the secondary ATU to be forwarded through the PCI-to-PCI bridge, claimed by the PATU and given to the messaging unit. Note that for this setup to work, the PATU and the SATU base addresses need to be set to the same location.
- On the 80960RM/RN, the PATU and the SATU can be programmed through the ATUCR to ignore the type of read access when matching on a Delayed Read Completion.
- The ATU's will now report PCI-like errors when errors occur on the internal bus.
- The 80960RM/RN contains a new feature that allows the programmer to swap the PCI function number of the ATU and the Bridge. The default is the same as the 80960RP/RD, Bridge is function 0 and ATU is function 1.
- VGA Addressing support has been removed.
- **XINT6#** and **XINT7#** are no longer accessible external to the chip.
- The Bus Interface Unit is new to the 80960RM/RN. It contains several registers that control the performance of the interface between the 80960 core processor and the internal bus. The default settings of these registers support maximum performance.

3.2 New Registers/Changed Registers

This section lists registers that did not exist on the 80960RP/RD and have been added to the 80960RM/RN, and registers that existed on the 80960RP/RD but have had bits either added or taken away on the 80960RM/RN. Please refer to the *i960® RM/RN I/O Processor Developer's Manual* for the specifics of these new registers.

- PCI-to-PCI Bridge:
 - Queue Control Register(QCR) - New Register

— Extended Bridge Control Register(EBCR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
9	Reserved	Secondary PCI Bus 64-Bit Capable
8	Reserved	Primary PCI Bus 64-Bit Capable
5	Reset Local Bus	Reset Internal Bus
4	Sync Mode	Reserved

— Primary Bridge Interrupt Status Register(PBISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
5	Reserved	Detected Parity Error

— Secondary Bridge Interrupt Status Register(SBISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
6	Reserved	Secondary Bus Reset Occurred
5	Reserved	Detected Parity Error

— Secondary Arbitration Control Register(SACR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
16	Secondary Arbiter Enable	Reserved

— Secondary Decode Enable Register(SDER) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
15	Reserved	S_SERR# Detected Interrupt Mask
14	Reserved	Secondary PCI Master Abort Interrupt Mask
13	Reserved	Secondary PCI Target Abort (Master) Interrupt Mask
12	Reserved	Secondary PCI Target Abort (Target) Interrupt Mask
11	Reserved	Secondary PCI Master Parity Error Interrupt Mask

Bit Number	80960RP/RD Definition	80960RM/RN Definition
10	Reserved	P_SERR# Asserted Interrupt Mask
9	Reserved	Primary PCI Master Abort Interrupt Mask
8	Reserved	Primary PCI Target Abort (Master) Interrupt Mask
7	Reserved	Primary PCI Target Abort (Target) Interrupt Mask
6	Reserved	Primary PCI Master Parity Error Interrupt Mask
5	Reserved	Secondary Detected Parity Error Bit Interrupt Mask
4	Reserved	Primary Detected Parity Error Bit Interrupt Mask
3	Reserved	Secondary Bus Reset Occurred Interrupt Mask
1	Secondary Positive Memory Decode Enable	Reserved
0	Secondary Positive I/O Decode Enable	Reserved

- Performance Monitoring Unit: All registers are new.
- Address Translation Unit:
 - Primary ATU Queue Control Register(PAQCR) - New Register
 - Secondary ATU Queue Control Register(SAQCR) - New Register
 - Primary ATU Interrupt Mask Register(PATUIMR) - New Register
 - Secondary ATU Interrupt Mask Register(PATUIMR) - New Register
 - Primary Inbound ATU Limit Register(PIALR) - Defaults to 8K, rather than 4K
 - Secondary Inbound ATU Limit Register(SIALR) - Defaults to 8K, rather than 4K
 - ATU Control Register(ATUCR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
21	Reserved	Bridge Function Number
20	Reserved	SATU DRC Alias
19	Reserved	PATU DRC Alias
18	Reserved	Direct Addressing Upper 2Gbytes Translation Enable
17	Reserved	S_SERR# Manual Assertion

Bit Number	80960RP/RD Definition	80960RM/RN Definition
16	Reserved	P_SERR# Manual Assertion
15	Reserved	ATU Discard Timer Status
12	Reserved	Secondary Bus Messaging Unit Access Enable
11	Secondary PCI Boot Mode	Reserved
10	Reserved	S_SERR# Interrupt Enable
9	Secondary Outbound Forwarding Enable	P_SERR# Interrupt Enable
6	Expansion ROM Width	Reserved
5	Secondary ATU PCI Error Interrupt Enable	Reserved
4	Primary ATU PCI Error Interrupt Enable	Reserved

— Primary ATU Interrupt Status Register(PATUISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
10	Reserved	P_SERR# Asserted
9	Reserved	Detected Parity Error
7	Reserved	Internal Bus Master Abort
6	80960 Memory Fault	Reserved
5	80960 Bus Fault	Reserved
4	P_SERR# Asserted	P_SERR# Detected

— Secondary ATU Interrupt Status Register(SATUISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
10	Reserved	S_SERR# Asserted
9	Reserved	Detected Parity Error
7	Reserved	Internal Bus Master Abort
6	80960 Memory Fault	Reserved
5	80960 Bus Fault	Reserved
4	S_SERR# Asserted	S_SERR# Detected

— Secondary ATU Status Register(SATUSR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
13	Reserved	Master Abort
12	Reserved	Target Abort (master)
11	Reserved	Target Abort (target)
8	Reserved	Master Parity Error

— Some ATU limit and base address registers have been changed to control the alignment and size of the defined memory region to a larger area than the registers allowed on the 80960RP/RD.

- Messaging Unit: No new registers.
- DMA Unit: No new registers.
- Memory Controller: All registers are new.
- Internal Arbitration Unit:
 - Internal Arbitration Control Register(IACR) - replaces LBACR
 - Master Latency Timer Register(MLTR) - replaces LBALCR
 - Multi-Transaction Timer Register(MTTR) - New Register
- Bus Interface Unit: All registers are new.
- I²C Unit: No new registers.
- Interrupt Controller: No new registers.
- Interrupt Controller:
 - XINT6 Interrupt Status Register(X6ISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
5	Reserved	Application Accelerator Interrupt Pending
4	Reserved	Performance Monitor Interrupt Pending
3	External XINT6# Interrupt Pending	Reserved

— XINT7 Interrupt Status Register(X7ISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
4	External XINT7# Interrupt Pending	Reserved
0	APIC Interrupt Pending	Reserved

— NMI Interrupt Status Register(NISR) - Changed Bits:

Bit Number	80960RP/RD Definition	80960RM/RN Definition
11	Reserved	Bus Interface Unit Error
10	Reserved	Application Accelerator Unit Error
0	80960 Core Error	Memory Controller Error

- Application Accelerator: All new registers.
- APIC Unit: This unit does not exist on the 80960RM/RN.

4.0 Recommended Register Settings

The register settings listed below are those that will be used in Intel's internal validation of the 80960RM/RN. These are the recommended settings for 80960RM/RN applications.

Register Name	80960RM/RN Internal Bus Address	Recommended Register Value	Description
Bridge Unit: Bridge Control Register (BCR)	0x103E	0x0B23	<ul style="list-style-type: none"> • Discard Timer SERR# Enable • Secondary Discard Timer Value is 2^{10} • Primary Discard Timer Value is 2^{10} • Master Abort Mode Enabled • ISA Disabled • Secondary SERR# Enabled • Secondary Parity Error Response Enabled
Bridge Unit: Extended Bridge Control Register (EBCR)	0x1040	Default	<ul style="list-style-type: none"> • Secondary DAC Medium Decode Disabled • Upstream Prefetchable Memory Enabled • Posting Enabled
Bridge Unit: Secondary Decode Enable Register (SDER)	0x105C	Default	<ul style="list-style-type: none"> • Interrupts Masked • Primary Memory Space Disabled
Bridge Unit: Queue Control Register (QCR)	0x105E	Default	<ul style="list-style-type: none"> • DRC Alias Disabled • MWI Alias Disabled • Upstream MRL Prefetch Size is 32 bytes • Downstream MRL Prefetch Size is 32 bytes

Register Name	80960RM/RN Internal Bus Address	Recommended Register Value	Description
ATU: ATU Configuration Register (ATUCR)	0x1288	0x00001106	<ul style="list-style-type: none"> • Bridge is Function 0, ATU is Function 1 • SATU DRC Aliasing Disabled • PATU DRC Aliasing Disabled • Direct Addressing Window Enabled for the PATU and located in the Lower 2-Gbytes of Memory • Secondary Bus Messaging Unit Access Enabled • ATU BIST, P_SERR# and S_SERR# Interrupts Disabled • Primary and Secondary Outbound ATU's Enabled
ATU: Secondary ATU Command Register (SATUCMD)	0x1298	0x0356	<ul style="list-style-type: none"> • Fast Back-to-Back Cycles Enabled • S_SERR# Enabled • Parity Error Checking Enabled for DMA Channel 2 and SATU • MWI Enabled for DMA Channel 2 • SATU Enabled as a Bus Master • SATU Enabled to Respond to PCI Memory Addresses • I/O Transactions Disabled
ATU: Primary ATU Queue Control Register (PAQCR)	0x12B4	Default	<ul style="list-style-type: none"> • Memory Write Non-Full Queue has 8 bytes or More Free
ATU: Secondary ATU Queue Control Register (SAQCR)	0x12B8	Default	<ul style="list-style-type: none"> • Memory Write Non-Full Queue has 8 bytes or More Free
ATU: Primary ATU Interrupt Mask Register (PATUIMR)	0x12BC	Default	<ul style="list-style-type: none"> • Interrupts Masked
ATU: Secondary ATU Interrupt Mask Register (SATUIMR)	0x12C0	Default	<ul style="list-style-type: none"> • Interrupts Masked

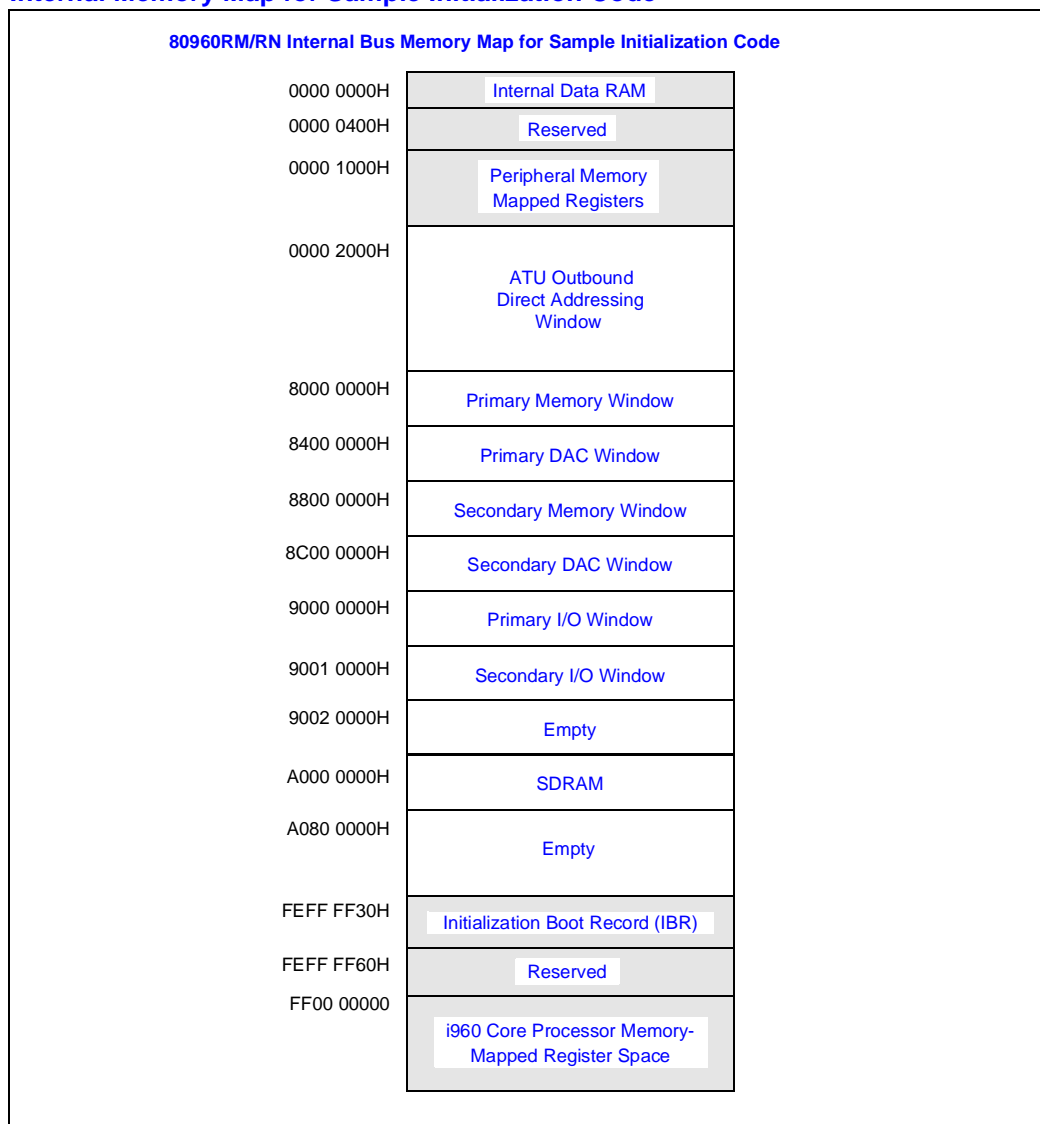
5.0 Initialization Code

80960RM/RN initialization code is available in its entirety from your Intel Field Representative as a zipped electronic file. This code has been customized for a specific board, but is easily portable to other systems. The files in this section are an example of the minimum amount of startup code necessary for the 80960RM/RN Microprocessor:

- Startup Routine (“[Internal Memory Map for Sample Initialization Code](#)” on page 16)
- Low-Level 80960 Assembly Code Routines (“[rm.s](#)” on page 20)
- High-Level Startup Code (“[main.c](#)” on page 21)
- Control Table & Register Initialization (“[evrm_hw.c](#)” on page 21)
- Initialization Boot Record File (“[rm_ibr.c](#)” on page 24)
- Linker Directive File (“[rm_init.ld](#)” on page 25)

- Include File (“evrm.h” on page 26)
- Makefile (“bld.bat” on page 32)

Figure 1. Internal Memory Map for Sample Initialization Code



5.1 init.s

```

/*-----*/
/* Startup Routine (init.s) */
/*-----*/
/* initial PRCB */

.global _rom_prcb

```



```

        .align 4 /* or .align 2 */
_rom_prcb:
        .word    boot_flt_table          # 0 - Fault Table
        .word    _boot_control_table    # 4 - Control Table
        .word    0x00001000             # 8 - AC reg mask overflow fault
        .word    0x40000001             # 12 - Flt CFG- Allow Unaligned
        .word    boot_intr_table        # 16 - Interrupt Table
        .word    rom_sys_proc_table     # 20 - System Procedure Table
        .word    0                      # 24 - Reserved
        .word    _intr_stack            # 28 - Interrupt Stack Pointer
        .word    0x00000000             # 32 - Inst. Cache - enable cache
        .word    0x5 # 36 - Register Cache Config.- 5 sets cached

/* ROM system procedure table */
        .equ     supervisor_proc, 2
        .text
        .align 6 /* or .align 2 or .align 4 */
rom_sys_proc_table:
        .space   12                    # Reserved
        .word    _supervisor_stack      # Supervisor stack pointer
        .space   32                    # Preserved
        .word    _default_sysproc       # sysproc 0
        .word    _default_sysproc       # sysproc 1
        .word    _default_sysproc       # sysproc 2
        .word    _default_sysproc       # sysproc 3
        .word    _default_sysproc       # sysproc 4
        .word    _default_sysproc       # sysproc 5
        .word    _default_sysproc       # sysproc 6
        .word    _fault_handler + supervisor_proc # sysproc 7
        .word    _default_sysproc       # sysproc 8
        .space   251*4                 # sysproc 9-259

/* Fault Table */
        .equ     syscall, 2
        .equ     fault_proc, 7
        .text
        .align   4

boot_flt_table:
        .word    (fault_proc<<2) + syscall # 0-Parallel Fault
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 1-Trace Fault
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 2-Operation Fault
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 3-Arithmetic Fault
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 4-Reserved
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 5-Constraint Fault
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 6-Reserved
        .word    0x27f
        .word    (fault_proc<<2) + syscall # 7-Protection Fault
        .word    0x27f

```



```

* Copy the .data into RAM. The .data has been packed in the ROM after the
* code area. If the copy is not needed (RAM-based monitor), the symbol
* rom_data can be defined as 0 in the linker directives file.
* (Some programs require copying the .data area into RAM.)
*/
    lda    rom_data, g1          # load source of copy
    cmpobe 0, g1, 1f
    lda    __Bdata, g2          # load destination
    lda    __Edata, g3
init_data:
    ldq    (g1), r4
    addo   16, g1, g1
    stq    r4, (g2)
    addo   16, g2, g2
    cmpobl g2, g3, init_data
1:
/* Initialize the BSS area of RAM. */
    lda    __Bbss, g2          # start of bss
    lda    __Ebss, g3          # end of bss
    movq   0, r4
bss_fill:
    stq    r4, (g2)
    addo   16, g2, g2
    cmpobl g2, g3, bss_fill
_reinit:
    ldconst 0x300, r4 # reinitialize sys control
    lda    1f, r5
    lda    _rom_prCB, r6
    sysctl r4, r5, r6
1:
    mov    0, g14
    lda    _user_stack, g0      /* new fp */
    lda    _user_stack, g1      /* new pfp */
    call   move_frame

    ldconst 0x001f2403, r3      /* PC mask */
    ldconst 0x000f0003, r4      /* PC value */
    modpc  r3, r3, r4          /* out of interrupted state */

/* Clear the IPND register */
    lda    0xff008500, g0
    mov    0, g1
    st     g1, (g0)
    callx  _main                #to main routine
terminated:
    fmark                    # cause breakpoint trace fault
    b     terminated

/* move_frame -
    g0 - new frame pointer (FP)
    g1 - new previous frame pointer (PFP)
This routine switches stacks. It should be called using a "local"
call. The new stack pointer (SP) is calculated by finding the
relative offset between the old FP and old SP, then adding this
offset to the new FP.

```

```

*/
move_frame:
    andnot    0xf, pfp, r3    /* old FP */
    mov      g0, r6          /* new FP */
    flushreg
    ld       4(r3), r4       /* old SP */
    subo    r3, r4, r5      /* old SP offset from FP */
1:
    ldq     (r3), r8        /* from old frame */
    addo    16, r3, r3
    stq     r8, (r6)       /* to new frame */
    addo    16, r6, r6
    cmpobl  r3, r4, 1b

    addo    g0, r5, r4      /* new SP */
    st      g1, (g0)       /* store new PFP in new frame */
    st      r4, 4(g0)      /* store new SP in new frame */
    mov     g0, pfp        /* new FP */
    flushreg               /* Not on CX!!! */
    ret

globl _intr_stack
    .globl _user_stack
    .globl _supervisor_stack
    .bss    _user_stack, 0x0200, 6        # default application stack
    .bss    _intr_stack, 0x0200, 6       # interrupt stack
    .bss    _supervisor_stack, 0x0600, 6 # fault (supervisor) stack
    .text
_fault_handler:
    ret
_default_sysproc:
    ret
_intx:
    ret

```

5.2 **rm.s**

```

/*-----*/
/* Low-Level 80960 Assembly Code Routines (rm.s) */
/*-----*/

#include "evrm.h"
.globl _set_mmr32
_set_mmr32:
    st    g1, (g0)
    ret

.globl _set_mmr16
_set_mmr16:
    stos g1, (g0)
    ret

.globl _set_mmr8
_set_mmr8:
    stob g1, (g0)

```

```

    ret
.globl _get_mmr
_get_mmr:
    ld (g0), g1
    mov g1, g0
    ret

```

5.3 main.c

```

/*-----*/
/* High-Level Startup Code (main.c)                */
/*-----*/

extern void init_hardware();
void
main()
{
    init_hardware();
    /* add your test routine here */
}

```

5.4 evrm_hw.c

```

/*-----*/
/*          Control Table & Register                */
/*          Initialization (evrm_hw.c)              */
/*-----*/

#include "evrm.h"
extern void init_sdram(),init_uart_led(), init_atus();
extern void init_bridge();

typedef struct
{
    unsigned control_reg[28];
}CONTROL_TABLE;

const CONTROL_TABLE boot_control_table = {
    /* -- Group 0 -- Breakpoint Registers (reserved by monitor) */
    {0, 0, 0, 0,

    /* -- Group 1 -- Interrupt Map Registers */
    0, 0, 0,          /* Interrupt Map Regs (set by code as needed) */
    0x4000,          /* ICON - enabled, mask unchanged, not cached */

    /* -- Groups 2-5 -- Bus Configuration Registers */
    /*All Memory Regions MUST be set to 32-bits Wide
    for 80960RM/RN. */
    REGION_0_CONFIG, 0,
    REGION_2_CONFIG, 0,

```

```

REGION_4_CONFIG, 0,
REGION_6_CONFIG, 0,
REGION_8_CONFIG, 0,
REGION_A_CONFIG, 0,
REGION_C_CONFIG, 0,
REGION_BOOT_CONFIG, 0,

/* -- Group 6 -- Breakpoint, Trace and Bus Control Registers */
0, /* Reserved */
0, /* BPCON Register (reserved by monitor) */
0, /* Trace Controls */
1} /* BCON Register */
};
/*-----
* Function: void init_hardware()
*
* Action: Sets up target control hardware and peripherals.
*-----*/
void
init_hardware()
{
    init_atus();
    init_bridge();
}
/*-----
* Function: void init_sdram()
*
* Action: Initialize the SDRAM. (called from init.s)
*-----*/
void
init_sdram()
{
    extern void set_mmr32();

/* Execute the SDRAM Initialization Sequence - for specifics, refer to the
SDRAM datasheet */

    set_mmr32(SDBR_ADDR, 0xa0000000);
        /* DRAM Base Address */
    set_mmr32(SBR0_ADDR, 0x80000002);
        /* SDRAM Bank 0: 8M of 64 MBit SDRAM */
        /* See "Programming Values for the SDRAM Boundary
Registers" table in the EAS */
    set_mmr32(SBR1_ADDR, 0x80000002);
        /* SDRAM Bank 1: empty */
    set_mmr32(SDCR_ADDR, 0x0);
        /* See "Drive Strength Programmability Options"
table in the EAS */
    set_mmr32(SDIR_ADDR, 0x3); /* NOP */
    /* Delay for 200 microsecs - see the EAS SDRAM Init section */
    set_mmr32(SDIR_ADDR, 0x2); /* Precharge */
    /* Delay for Trp - see your SDRAM datasheet */
    set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
    /* Delay for Trc - see your SDRAM datasheet */
    set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */

```



```
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x4); /* Auto-Refresh */
/* Delay for Trc - see your SDRAM datasheet */
set_mmr32(SDIR_ADDR, 0x0); /* Mode Register Set, CAS# Latency = 2 */
/* The SDRAM controller assures that a row-activate command is
   issued no sooner than 3 cycles after the MRS command. */

}
/*-----
 * Function:      void init_uart_led()
 *
 * Action:       Initialize the UART and the LED located in
 *               Flash Bank 1. (called from init.s)
 *-----*/
void
init_uart_led()
{
    extern void set_mmr32();

    set_mmr32(FEBR1_ADDR, 0xE0000000); /* LED/UART Address */
    set_mmr32(FBSR1_ADDR, 0x00000001); /* 64 KB */
    set_mmr32(FWSR1_ADDR, 0x00000077);
        /* 20 address to data wait-states
           20 recovery wait-states */
}
/*-----
 * Function:      void init_atus()
 *
 * Action:       Initialize the Primary and Secondary ATU's.
 *-----*/
void
init_atus()
{
    extern void set_mmr32();
    extern void set_mmrl6();

    set_mmr32(PIABAR_ADDR, 0xB0000008);
        /*Primary Inbound ATU Base Address
           memory is prefetchable*/
    set_mmr32(PIALR_ADDR, 0xFFE00000);
        /* Primary Inbound ATU Limit
           2 Mbytes */
    set_mmr32(PIATVR_ADDR, 0xA0000000);
        /* Primary Inbound ATU Translate Value */
    set_mmr32(SIABAR_ADDR, 0xD0000008);
}
```

```

        /*Secondary Inbound ATU Base Address
        memory is prefetchable*/
set_mmr32(SIALR_ADDR, 0xFFE00000);
        /* Secondary Inbound ATU Limit
        2 Mbytes */
set_mmr32(SIATVR_ADDR, 0xA0000000);
        /* Secondary Inbound ATU Translate Value */
set_mmr32(ATUCR_ADDR,0x00001106);
        /* ATU Configuration
        Direct Addressing Enabled on Primary PCI bus
        Secondary Bus Messaging Unit Access Enabled
        Pri&Sec Outbound ATU Enabled      */
set_mmr16(SATUCMD_ADDR,0x0356);
        /* Fast Back-to-Back Cycles Enabled
        S_SERR# Enabled
        Parity Error Checking Enabled for DMA Channel 2 and SATU
        MWI Enabled for DMA Channel 2
        SATU Enabled as a Bus Master
        SATU Enabled to Respond to PCI Memory Addresses
        I/O Transactions Disabled
        */
}
/*-----
* Function:      void init_bridge()
*
* Action:        Initialize the PCI-to-PCI Bridge.
*-----*/
void
init_bridge()
{
    extern void set_mmr16();
    set_mmr16(BCR_ADDR, 0x0B23);
        /*Discard Timer SERR# Enable
        Secondary Discard Timer Value is 210
        Primary Discard Timer Value is 210
        Master Abort Mode Enabled
        ISA Disabled
        Secondary SERR# Enabled
        Secondary Parity Error Response Enabled
        */

    set_mmr16(EBCR_ADDR, 0x10);
        /* Clear the Configuration Retry Bit to Allow the
        Host BIOS to start its configuration*/
}

```

5.5 `rm_ibr.c`

```

/*-----*/
/*      Initialization Boot Record File (rm_ibr.c)      */
/*-----*/

/* * THE INITIALIZATION BOOT RECORD MUST BE LOCATED AT ADDRESS

```



```

    * 0xFFFFFFFF30 BY THE LINKER! */
#include "evrm.h"
extern void start_ip();
extern unsigned rom_prpcb;
extern unsigned checksum;          /* symbol calculated at link time */
#define CS_6 (int) &checksum

typedef struct
{
    unsigned    bus_byte_0;
    unsigned    bus_byte_1;
    unsigned    bus_byte_2;
    unsigned    bus_byte_3;
    void        (*first_inst)();
    unsigned    *prpcb_ptr;
    int         check_sum[6];
}IBR;

    /***Boot Region MUST be set to 32-bits Wide for 80960RM/RN. ***/

const IBR init_boot_record = {
    0x00000000, /* PMCON14_15, byte 0 */
    0x00000000, /* PMCON14_15, byte 1*/
    0x00000080, /* PMCON14_15, byte 2*/
    0x00000000, /* PMCON14_15, byte 3*/
    start_ip, /* First Instruction Pointer */
    &rom_prpcb, /* PRCB Pointer */
    -2,          /* Checksum word #1 */
    0,          /* Checksum word #2 */
    0,          /* Checksum word #3 */
    0,          /* Checksum word #4 */
    0,          /* Checksum word #5 */
    CS_6 /* Checksum word #6: * -(start_ip+rom_prpcb)
        * must be calculated by the linker */
};

```

5.6 rm_init.ld

```

/*-----*/
/*          Linker Directive File (rm_init.ld)          */
/*-----*/

MEMORY
{
    rom:          o=0xfefc0000,l=0x1fc00
    rom_data:    o=0xfefdfc00,l=0x0300 /*Enough space must be reserved in
rom*/

    /*after the text section to hold the */
    /*initial values of the data section. */
    ibr:          o=0xfeffff30,l=0x00cf
    data:        o=0xa0000000,l=0x0300
    bss:         o=0xa0000300,l=0x7d00
}

```

```

SECTIONS
{
  .ibr :
  {
    { evrmibr.o
    } > ibr
  }
  .text :
  {
    {
    } > rom
  }
  .data :
  {
    {
    } > data
  }
  .bss :
  {
    {
    } > bss
  }
}

rom_data = __Etext;
_checksum = -(_rom_prcb + _start_ip);
HLL()

/* Rommer script embedded here:
**move $0 .text 0
**move $0
**move $0 .ibr 0x3FF30
**mkimage $0 $0.ima
**ihex $0.ima $0.hex mode16
**map $0
**quit
*/

```

5.7 evrm.h

```

/*-----*/
/*          Include File (evrm.h)          */
/*-----*/

/* 80960JN Core memory mapped register addresses */
#define DLMCON_ADDR    0xff008100
#define LMAR0_ADDR    0xff008108
#define LMMR0_ADDR    0xff00810c
#define LMAR1_ADDR    0xff008110
#define LMMR1_ADDR    0xff008114
#define IPB0_ADDR     0xff008400
#define IPB1_ADDR     0xff008404
#define DAB0_ADDR     0xff008420
#define DAB1_ADDR     0xff008424
#define BPCON_ADDR    0xff008440
#define IPND_ADDR     0xff008500
#define IMSK_ADDR     0xff008504
#define ICON_ADDR     0xff008510
#define IMAPO_ADDR    0xff008520
#define IMAPI_ADDR    0xff008524
#define IMAPI2_ADDR   0xff008528
#define PMCON0_ADDR   0xff008600

```



```
#define PMCON2_ADDR 0xff008608
#define PMCON4_ADDR 0xff008610
#define PMCON6_ADDR 0xff008618
#define PMCON8_ADDR 0xff008620
#define PMCON10_ADDR 0xff008628
#define PMCON12_ADDR 0xff008630
#define PMCON14_ADDR 0xff008638
#define BCON_ADDR 0xff0086fc
#define PRCB_ADDR 0xff008700
#define ISP_ADDR 0xff008704
#define SSP_ADDR 0xff008708
#define DEVID_ADDR 0xff008710
#define TRR0_ADDR 0xff000300
#define TCR0_ADDR 0xff000304
#define TMR0_ADDR 0xff000308
#define TRR1_ADDR 0xff000310
#define TCR1_ADDR 0xff000314
#define TMR1_ADDR 0xff000318

/* PCI-to-PCI Bridge Unit 0x00001000H through 0x000010FFH */
#define VIDR_ADDR 0x00001000
#define DIDR_ADDR 0x00001002
#define PCMDR_ADDR 0x00001004
#define PSR_ADDR 0x00001006
#define RIDR_ADDR 0x00001008
#define CCR_ADDR 0x00001009
#define CLSR_ADDR 0x0000100C
#define PLTR_ADDR 0x0000100D
#define HTR_ADDR 0x0000100E
/* Reserved 0x0000100F through 0x00001017 */
#define PBNR_ADDR 0x00001018
#define SBNR_ADDR 0x00001019
#define SUBBNR_ADDR 0x0000101A
#define SLTR_ADDR 0x0000101B
#define IOBR_ADDR 0x0000101C
#define IOLR_ADDR 0x0000101D
#define SSR_ADDR 0x0000101E
#define MBR_ADDR 0x00001020
#define MLR_ADDR 0x00001022
#define PMBR_ADDR 0x00001024
#define PMLR_ADDR 0x00001026
/* Reserved 0x00001028 through 0x00001033 */
#define BSVIR_ADDR 0x00001034
#define BSIR_ADDR 0x00001036
/* Reserved 0x00001038 through 0x0000103D */
#define BCR_ADDR 0x0000103E
#define EBCR_ADDR 0x00001040
#define SISR_ADDR 0x00001042
#define PBISR_ADDR 0x00001044
#define SBISR_ADDR 0x00001048
#define SACR_ADDR 0x0000104C
#define PIRSR_ADDR 0x00001050
#define SIOBR_ADDR 0x00001054
#define SIOLR_ADDR 0x00001055
#define SMBR_ADDR 0x00001058
```

```

#define SMLR_ADDR 0x0000105A
#define SDER_ADDR 0x0000105C
#define QCR_ADDR 0x0000105E
/* Reserved 0x00001060 through 0x000010FFH */

/* Performance Monitoring Unit 0x00001100 through 0x000011FF */
#define GTMR_ADDR 0x00001100
#define ESR_ADDR 0x00001104
#define EMISR_ADDR 0x00001108
/* Reserved 0x0000110C */
#define GTSR_ADDR 0x00001110
#define PECR1_ADDR 0x00001114
#define PECR2_ADDR 0x00001118
#define PECR3_ADDR 0x0000111C
#define PECR4_ADDR 0x00001120
#define PECR5_ADDR 0x00001124
#define PECR6_ADDR 0x00001128
#define PECR7_ADDR 0x0000112C
#define PECR8_ADDR 0x00001130
#define PECR9_ADDR 0x00001134
#define PECR10_ADDR 0x00001138
#define PECR11_ADDR 0x0000113C
#define PECR12_ADDR 0x00001140
#define PECR13_ADDR 0x00001144
#define PECR14_ADDR 0x00001148
/* Reserved 0x00001150 through 0x000011FF */

/* Address Translation Unit 0x00001200H through 0x000012FFH */
#define ATUVID_ADDR 0x00001200
#define ATUDID_ADDR 0x00001202
#define PATUCMD_ADDR 0x00001204
#define PATUSR_ADDR 0x00001206
#define ATURID_ADDR 0x00001208
#define ATUCCR_ADDR 0x00001209
#define ATUCLSR_ADDR 0x0000120C
#define ATULT_ADDR 0x0000120D
#define ATUHTR_ADDR 0x0000120E
#define ATUBISTR_ADDR 0x0000120F
#define PIABAR_ADDR 0x00001210
/* Reserved 0x00001214 */
/* Reserved 0x00001218 */
/* Reserved 0x0000121C */
/* Reserved 0x00001220 */
/* Reserved 0x00001224 */
/* Reserved 0x00001228 */
#define ASVIR_ADDR 0x0000122C
#define ASIR_ADDR 0x0000122E
#define ERBAR_ADDR 0x00001230
/* Reserved 0x00001234 */
/* Reserved 0x00001238 */
#define ATUILR_ADDR 0x0000123C
#define ATUIPR_ADDR 0x0000123D
#define ATUMGNT_ADDR 0x0000123E
#define ATUMLAT_ADDR 0x0000123F
#define PIALR_ADDR 0x00001240

```



```
#define PIATVR_ADDR 0x00001244
#define SIABAR_ADDR 0x00001248
#define SIALR_ADDR 0x0000124C
#define SIATVR_ADDR 0x00001250
#define POMWVR_ADDR 0x00001254
/* Reserved 0x00001258 */
#define POIOWVR_ADDR 0x0000125C
#define PODWVR_ADDR 0x00001260
#define POUDR_ADDR 0x00001264
#define SOMWVR_ADDR 0x00001268
#define SOIOWVR_ADDR 0x0000126C
/* Reserved 0x00001270 */
#define ERLR_ADDR 0x00001274
#define ERTVR_ADDR 0x00001278
/* Reserved 0x0000127C */
/* Reserved 0x00001280 */
/* Reserved 0x00001284 */
#define ATUCR_ADDR 0x00001288
/* Reserved 0x0000128C */
#define PATUISR_ADDR 0x00001290
#define SATUISR_ADDR 0x00001294
#define SATUCMD_ADDR 0x00001298
#define SATUSR_ADDR 0x0000129A
#define SODWVR_ADDR 0x0000129C
#define SOUDR_ADDR 0x000012A0
#define POCCAR_ADDR 0x000012A4
#define SOCCAR_ADDR 0x000012A8
#define POCCDR_ADDR 0x000012AC
#define SOCCDR_ADDR 0x000012B0
#define PAQCR_ADDR 0x000012B4
#define SAQCR_ADDR 0x000012B8
#define PATUIMR_ADDR 0x000012BC
#define SATUIMR_ADDR 0x000012C0
/* Reserved 0x000012C4 through 0x000012FF */

/* Messaging Unit 0x00001300H through 0x000013FFH */
/* Reserved 0x00001300 */
/* Reserved 0x00001304 */
/* Reserved 0x00001308 */
/* Reserved 0x0000130C */
#define IMR0_ADDR 0x00001310
#define IMR1_ADDR 0x00001314
#define OMR0_ADDR 0x00001318
#define OMR1_ADDR 0x0000131C
#define IDR_ADDR 0x00001320
#define IISR_ADDR 0x00001324
#define IIMR_ADDR 0x00001328
#define ODR_ADDR 0x0000132C
#define OISR_ADDR 0x00001330
#define OIMR_ADDR 0x00001334
/* Reserved 0x00001338 through 0x0000134F */
#define MUCR_ADDR 0x00001350
#define QBAR_ADDR 0x00001354
/* Reserved 0x00001358 */
/* Reserved 0x0000135C */
```

```

#define IFHPR_ADDR 0x00001360
#define IFTPR_ADDR 0x00001364
#define IPHPR_ADDR 0x00001368
#define IPTPR_ADDR 0x0000136C
#define OFHPR_ADDR 0x00001370
#define OFTPR_ADDR 0x00001374
#define OPHPR_ADDR 0x00001378
#define OPTPR_ADDR 0x0000137C
#define IAR_ADDR 0x00001380
/* Reserved 0x00001384 through 0x000013FF */

/* DMA Controller 0x00001400H through 0x000014FFH */
#define CCR0_ADDR 0x00001400
#define CSR0_ADDR 0x00001404
/* Reserved 0x00001408 */
#define DAR0_ADDR 0x0000140C
#define NDAR0_ADDR 0x00001410
#define PADR0_ADDR 0x00001414
#define PUADR0_ADDR 0x00001418
#define LADR0_ADDR 0x0000141C
#define BCR0_ADDR 0x00001420
#define DCR0_ADDR 0x00001424
/* Reserved 0x00001428 through 0x0000143F */
#define CCR1_ADDR 0x00001440
#define CSR1_ADDR 0x00001444
/* Reserved 0x00001448 */
#define DAR1_ADDR 0x0000144C
#define NDAR1_ADDR 0x00001450
#define PADR1_ADDR 0x00001454
#define PUADR1_ADDR 0x00001458
#define LADR1_ADDR 0x0000145C
#define BCR1_ADDR 0x00001460
#define DCR1_ADDR 0x00001464
/* Reserved 0x00001468 through 0x0000147F */
#define CCR2_ADDR 0x00001480
#define CSR2_ADDR 0x00001484
/* Reserved 0x00001488 */
#define DAR2_ADDR 0x0000148C
#define NDAR2_ADDR 0x00001490
#define PADR2_ADDR 0x00001494
#define PUADR2_ADDR 0x00001498
#define LADR2_ADDR 0x0000149C
#define BCR2_ADDR 0x000014A0
#define DCR2_ADDR 0x000014A4
/* Reserved 0x000014A8 through 0x000014FF */

/* Memory Controller 0x00001500H through 0x000015FFH */
#define SDIR_ADDR 0x00001500
#define SDCR_ADDR 0x00001504
#define SDBR_ADDR 0x00001508
#define SBR0_ADDR 0x0000150C
#define SBR1_ADDR 0x00001510
/* Reserved 0x00001514 through 0x00001530 */
#define ECCR_ADDR 0x00001534
#define ELOG0_ADDR 0x00001538

```



```
#define ELOG1_ADDR 0x0000153C
#define ECAR0_ADDR 0x00001540
#define ECAR1_ADDR 0x00001544
#define ECTST_ADDR 0x00001548
#define FEBR0_ADDR 0x0000154C
#define FEBR1_ADDR 0x00001550
#define FBSR0_ADDR 0x00001554
#define FBSR1_ADDR 0x00001558
#define FWSR0_ADDR 0x0000155C
#define FWSR1_ADDR 0x00001560
#define MCISR_ADDR 0x00001564
#define RFR_ADDR 0x00001568
/* Reserved 0x0000156C through 0x000015FF */

/* Internal Arbitration Unit 0x00001600H through 0x0000163FH */
#define IACR_ADDR 0x00001600
#define MLTR_ADDR 0x00001604
#define MTTR_ADDR 0x00001608
/* Reserved 0x0000160C through 0x0000163F */

/* Bus Interface Unit 0x00001640 through 0x0000167F */
#define BIUCR_ADDR 0x00001640
#define BIUISR_ADDR 0x00001644
/* Reserved 0x00001648 through 0x0000167F */

/* I2C Bus Interface Unit 0x00001680H through 0x000016FFH */
#define ICR_ADDR 0x00001680
#define ISR_ADDR 0x00001684
#define ISAR_ADDR 0x00001688
#define IDBR_ADDR 0x0000168C
#define ICCR_ADDR 0x00001690
#define IBMR_ADDR 0x00001694
/* Reserved 0x00001698 through 0x000016FF */

/* PCI And Peripheral Interrupt Controller 0x00001700H through 0x0000177FH
*/
#define NISR_ADDR 0x00001700
#define X7ISR_ADDR 0x00001704
#define X6ISR_ADDR 0x00001708
#define PDDIR_ADDR 0x00001710
/* Reserved 0x00001714 through 0x000017FF */

/* Application Accelerator Unit 0x00001800 through 0x000018FF */
#define ACR_ADDR 0x00001800
#define ASR_ADDR 0x00001804
#define ADAR_ADDR 0x00001808
#define ANDAR_ADDR 0x0000180C
#define SAR1_ADDR 0x00001810
#define SAR2_ADDR 0x00001814
#define SAR3_ADDR 0x00001818
#define SAR4_ADDR 0x0000181C
#define DAR_ADDR 0x00001820
#define ABCR_ADDR 0x00001824
#define ADCR_ADDR 0x00001828
#define SAR5_ADDR 0x0000182C
```

```

#define SAR6_ADDR 0x00001830
#define SAR7_ADDR 0x00001834
#define SAR8_ADDR 0x00001838
/* Bus configuration */
/**** The bus width for ALL Memory regions of the 80960RM/RN must be set
to 32-bits. ****/
#define BUS_WIDTH320x00800000

/* Region Configuration */
#define REGION_0_CONFIGBUS_WIDTH32
#define REGION_2_CONFIGBUS_WIDTH32
#define REGION_4_CONFIGBUS_WIDTH32
#define REGION_6_CONFIGBUS_WIDTH32
#define REGION_8_CONFIGBUS_WIDTH32
#define REGION_A_CONFIGBUS_WIDTH32
#define REGION_C_CONFIGBUS_WIDTH32
#define REGION_E_CONFIGBUS_WIDTH32
#define REGION_BOOT_CONFIGBUS_WIDTH32

```

5.8 bld.bat

```

/*-----*/
/* Example 7. Makefile (bld.bat) */
/*-----*/
echo -AJF > options.cl
echo -DTARGET -I. -I../.../hdil/common >> options.cl
copy init.s tmp1.c
ic960 @options.cl -DALLOW_UNALIGNED -E tmp1.c > tmp1.s
asm960 -AJF -o init.o tmp1.s
del tmp1.?
ic960 @options.cl -c main.c
ic960 @options.cl -c evrm_hw.c
copy rm.s tmp2.c
ic960 @options.cl -E tmp2.c > tmp2.s
asm960 -AJF -o rm.o tmp2.s
del tmp2.?
ic960 @options.cl -o evrmibr.o -c rm_ibr.c
del lst_evrm.dos
echo init.o >> lst_evrm.dos
echo main.o >> lst_evrm.dos
echo evrm_hw.o >> lst_evrm.dos
echo rm.o >> lst_evrm.dos
echo evrmibr.o >> lst_evrm.dos
lnk960 -AJF -m -Ninito.map -z -o inito -Trm_init @lst_evrm.dos
cof960 -lpz inito
rom960 rm_init inito
del lst*.dos
del options.cl
del *.o

```