Motorola HC12

CPU Awareness and True-Time Simulation

Revised 07/17/2003

metrowerks
Software Starts Here

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	`http://www.metrowerks.com`
Sales	Voice: 800-377-5416 Fax: 512-996-4910 Email: sales@metrowerks.com
Technical Support	Voice: 800-377-5416 Email: support@metrowerks.com

Table of Contents

1

Introduction

This manual explains the Metrowerks target software simulator.

Read the Release Notes

Before you use your CodeWarrior^{TM} IDE simulator, you should read its product release notes. The release notes include important last-minute information about new features, problem workarounds, or incompatibilities that may not be included in this manual.

Simulator Target Component

This section helps you start using the Simulator Target Component.

Introduction

Simulator software simulates a target system. The simulator consists of a CPU simulator, a memory simulator, and several simulated I/O devices. The simulator lets you set the simulated environment: memory, I/O-device placement, code, and so forth.

The simulator includes a universal timing facility that components can use to simulate realistic timing conditions. This facility lets components take control after a certain number of clock cycles or processor instructions.

You load the simulator driver as part of loading the simulator target component.

Simulator Setup

This section explains how to load the Simulator target.

Default Target Setup

As with any other target, you can load the simulator target component from the CodeWarrior IDE *Target* menu. Alternatively, you can use the `PROJECT.INI` file (Listing 1.1 on page 6) to set the simulator target component as the default target.

Listing 1.1 Example of PROJECT.INI File

```
[HI-WAVE]
Window0=Source      0   0  60  30
Window1=Assembly   60   0  40  30
Window2=Procedur    0  30  60  25
Window3=Register   60  30  40  30
Window4=Memory     60  60  40  40
Window5=Data        0  55  60  23
Window6=Data        0  78  60  22
Target=Sim
```

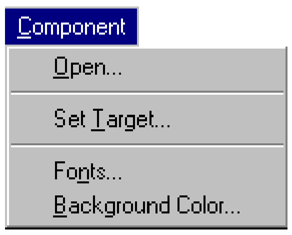
NOTE The HI-WAVE User's Guide has additional information about the `PROJECT.INI` file.

Loading the Simulator Target

The `PROJECT.INI` file line **`Target=Sim`** sets the target to be the simulator target component.

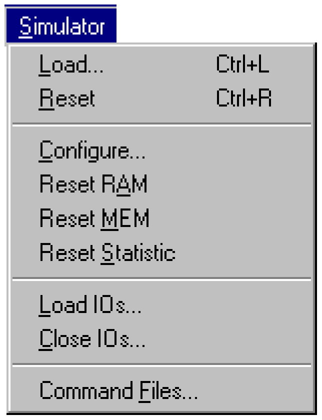
If the `PROJECT.INI` file does not set a target, or if it sets a different target, you can use the main menu to select the simulator. Select **Component > Set Target...** , as Figure 1.1 depicts. Choose **Simulator** from the list of possible targets.

Figure 1.1 The Component Menu



After loading, the **Simulator** (Figure 1.2 on page 7) replaces the **Target** menu.

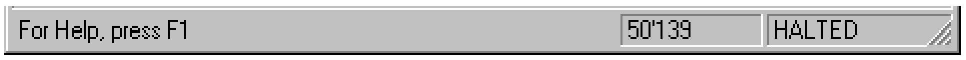
Figure 1.2 The Simulator Menu



The HI-WAVE Status Bar for the Simulator

Once you have loaded the Simulator Target Component, the HI-WAVE status bar (Figure 1.3) shows status and other information. As well as execution status, it includes a context-sensitive menu help line, and target- specific information like the number of CPU cycles (64 bits) since the application started.

Figure 1.3 The Debugger Status Bar



Simulator Target Component Features

This section explains the major features of the Simulator Target Component.

Introduction

The memory configuration facility is an integral part of HI-WAVE's advanced target configuration possibilities. The memory is divided into blocks. Amemory manager handles the list of memory blocks. The memory configuration facility offers you some degree of automation, but does not restrict the flexibility of manual adjustment. The memory configuration facility lets you specify types and properties of memory blocks, such as RAM, ROM, and so forth.

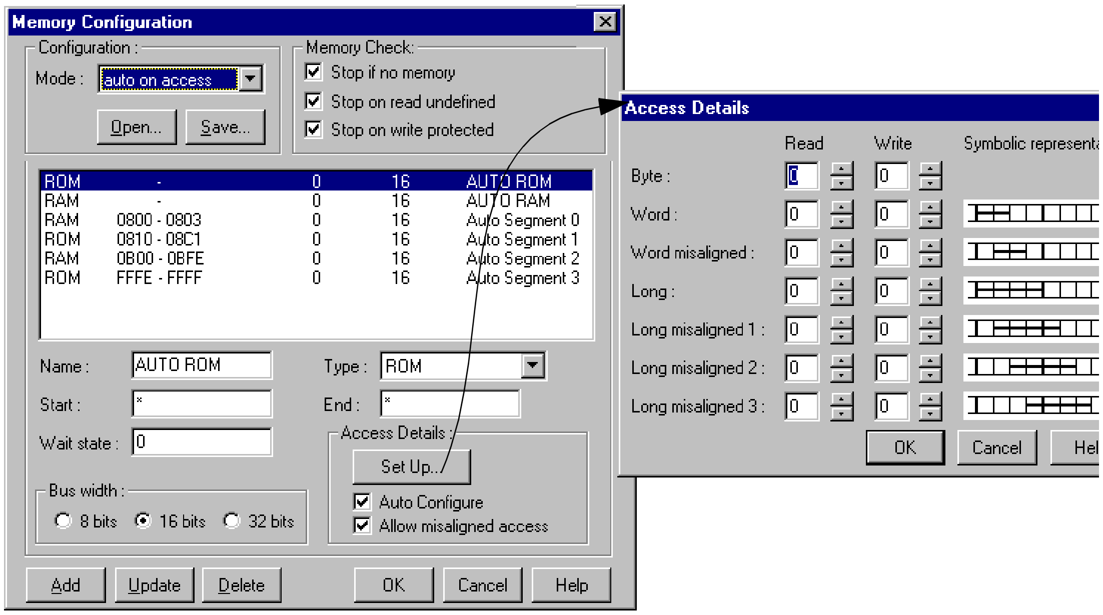
The memory configuration facility uses a binary file format to read and set the target configuration. The extension for binary files is `.mem`; the default memory file is `default.mem`. (The subsection "Format of the Default Memory Configuration File" includes Listing 1.2 on page 14, the EBNF-syntax definition of the file format.)

Memory Configuration Dialog Box Features

The memory configuration dialog box (Figure 1.4 on page 8) lets you perform these memory-block operations interactively:

- Select the configuration mode for simulation
- Define a memory block name
- Define how the simulator verifies the memory
- Set the type of the memory: RAM, ROM, FLASH, EEPROM or I/O
- Define start and end addresses
- Define the wait state (the time for each read or write access)
- Set the width of the bus that accesses the memory
- Set access details like:
  - *auto configure*: automatically computing read and write access
  - *misaligned access*: allowing misaligned access on words and longs
- Open and save memory configuration
- Add, delete, or update memory blocks

Figure 1.4 The Memory Configuration Dialog Box



Memory Configuration

Configuration : Mode : auto on access

Memory Check: [✓] Stop if no memory; [✓] Stop on read undefined; [✓] Stop on write protected

ROM	-	0	16	AUTO ROM
RAM	-	0	16	AUTO RAM
RAM	0800 - 0803	0	16	Auto Segment 0
ROM	0810 - 08C1	0	16	Auto Segment 1
RAM	0B00 - 0BFE	0	16	Auto Segment 2
ROM	FFFE - FFFF	0	16	Auto Segment 3

Name : AUTO ROM; Type : ROM; Start : *; End : *; Wait state : 0

Bus width : 8 bits [✓ 16 bits] 32 bits

Access Details : [✓] Auto Configure; [✓] Allow misaligned access

Access Details

	Read	Write
Byte :	0	0
Word :	0	0
Word misaligned :	0	0
Long :	0	0
Long misaligned 1 :	0	0
Long misaligned 2 :	0	0
Long misaligned 3 :	0	0

Memory Configuration Modes

Use the **Memory Configuration** dialog box to select the memory configuration mode: **auto configuration on access**, **auto configuration on load,** or **user defined**. Depending on your settings, the the simulator target component initializes target memory as Table 1.1 explains.

Table 1.1 Memory Configuration Modes

Mode	Description
Auto Configuration on Access (Standard Configuration)	Defines target memory as RAM of unlimited size. The *Mode* combo box displays *auto on access*.
Auto Configuration on Load (default)	Defines target memory as RAM and ROM, according to the code and data area defined in a loaded absolute file. Defines code segments as ROM. Defines data segments as RAM. (Memory outside these segments is *not implemented*; access to not-implemented locations result in error messages.) The *Mode* combo box displays *auto on load*.
Manual Configuration: (User Defined)	Defines target memory as RAM, ROM, non-volatile RAM, ... , depending on your configuration. You construct this definition interactively with the Memory Configuration dialog box, or read it in from a file. The *Mode* combo box displays *user defined*.

Memory Configuration Settings

Depending on the configuration mode, the Memory Configuration dialog box lets you redefine memory settings within certain limits. You always must set I/O devices manually.

Standard Configuration: Auto on Access: The Memory Configuration dialog box contains a single RAM entry with unspecified (*) starting and ending addresses. You cannot modify these addresses. You can adjust wait states, and other such settings, only for the whole RAM block.

Auto Configuration on Load: Initially, the dialog box lists a single RAM and a single ROM block, with unspecified (*) starting and ending addresses. You can adjust wait states, and other such settings, separately for RAM and ROM blocks.

For the ELF/DWARF Object file format, the Memory Configuration dialog box lists separate RAM and ROM blocks for each data and code segment in the absolute file, once an application has been loaded. The segmane addresses and lengths determine the starting and ending addresses of each block; you cannot modify these addresses.

Initial attributes of each code and data block come from the corresponding initial RAM and ROM blocks; you can modify these attributes independently.

Manual Configuration: The Memory Configuration dialog box lists an entry for each memory block. You can modify such entries without restriction.

NOTE To simulate an absolute file generated in HIWARE object file format, you must open the Memory Configuration dialog box, set the "**auto on load**" mode, then add a new RAM segment. The start and end addresses of this segment must match the associated `.prm` file. Once you close the dialog box, you can load your application and start a simulation.

Open Memory Block

Click the **Open** button to load a memory blocks file. The **Open Memory blocks** standard dialog box appears. Select a memory map file, then click the **OK** button. The dialog box closes, and the system loads the memory blocks file.

The *Mode* combo box changes to indicate the mode contained in the memory map file.

The list box lists the memory blocks loaded from the file, selecting the first memory block. Appropriate data appears in the fields **Name**, **Type**, **Start**, **End**, **Wait state**, **Bus width** and **Access Details**.

Save Memory Block

Click the **Save** button to store the current memory blocks configuration. The **Save Memory blocks** standard dialog box appears. Enter a file name, then click the **OK** button. The dialog box closes, and the system stores the memory block configuration into the file.

Memory Check Options

The Memory Check group box consists of three checkboxes, all checked when you bring up the Memory Configuration dialog box:

- Stop if no memory — Check this box to have the simulator stop upon an access to non-existent memory. (If you do not want the simulator to stop, clear this checkbox.)
- Stop on read undefined — Check this box to have the simulator stop upon a read of undefined memory. (If you do not want the simulator to stop, clear this checkbox.)

- Stop on write protected — Check this box to have the simulator stop upon a write to read-only (write-protected) memory. (If you do not want the simulator to stop, clear this checkbox.)

Memory Configuration Module Startup

Memory configuration is a *dynamically loaded* facility. That is, the new entry **Configure**... appears in the *Simulator* menu upon loading of the target (the Simulator dll). Selecting **Configure**... opens the Memory Configuration dialog box, so that you can configure memory.

Memory Block Setting

You must set memory blocks within the available memory; each block must cover a certain range. The *start address* and *end address* define each memory block.

Memory Block Properties

Table 1.2 lists the properties you may specify for a memory block:

Table 1.2 The Memory Block Properties

Item	Description
name	Name of the memory block.
type	RAM, ROM, FLASH, EEPROM or I/O
start	Start address of the memory block
end	End address of the memory block
wait state	Time used for reading or writing a specific number of bytes
bus width	Width of the bus that accesses the memory
read access	Table that defines read-access details on Byte, Word, Word misaligned, Long, and Long misaligned
write access	Table that defines write-access details on Byte, Word, Word misaligned, Long, and Long misaligned
auto configure	Flag that directs automatic computation of read and write accesses

Table 1.2 The Memory Block Properties

Item	Description
allow misaligned access	Flag that allows Word misaligned and Long misaligned
block type	USER_DEF (block you define), AUTO_GEN (block automatically generated), AUTO_MEM (master block for standard configuration), AUTO_RAM (RAM master block for auto configuration), or AUTO_ROM (ROM master block for auto configuration)

Memory Configuration Command Buttons

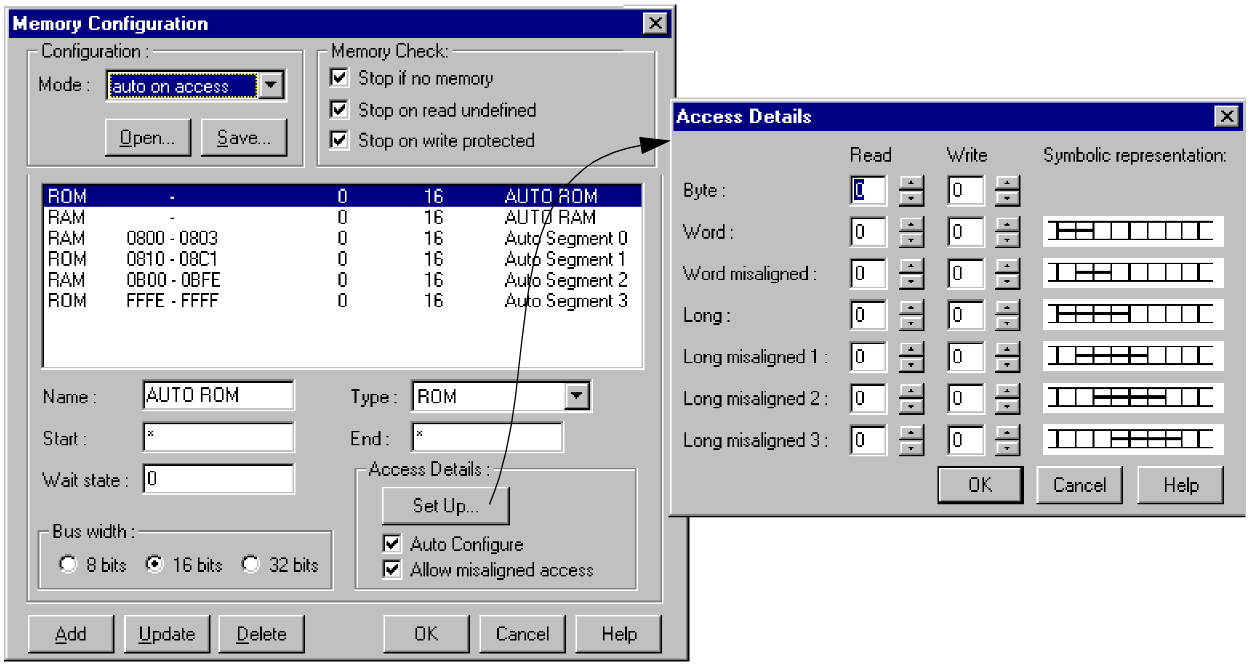
The command buttons of this dialog box are:

- **Add** — Fills a new memory block according to the current data of the **Name**, **Type**, **Start**, **End**, **Bus width**, and **Access Details** controls. This new memory block appears at the end of the list box. If there are any errors in this new block (such as an improper field value), the system generates a message box that informs you of the problem.
- **Update** — Updates the current memory block according to the current data of the **Name**, **Type**, **Start**, **End**, **Bus width**, and **Access Details** controls.
- **Delete** — Removes the currently selected memory block from the list box. The list box contents adjust, to reflect this deletion.
- **OK** — Closes the dialog box and validates the list of modified memory blocks. The parent class can access this list, updating its own list.
- **Cancel** — Closes the dialog box, canceling your modifications.
- **Help** — Opens the dialog-box help file.

Access Details Dialog Box Features

Figure 1.5 shows the **Access Details** dialog box, which lets you change read and write access values for seven types.

Figure 1.5 Access Dialog Window



Memory Configuration

Configuration: Mode: auto on access

Memory Check:
- [✓] Stop if no memory
- [✓] Stop on read undefined
- [✓] Stop on write protected

```
ROM     -              0    16    AUTO ROM
RAM     -              0    16    AUTO RAM
RAM     0800 - 0803    0    16    Auto Segment 0
ROM     0810 - 08C1    0    16    Auto Segment 1
RAM     0B00 - 0BFE    0    16    Auto Segment 2
ROM     FFFE - FFFF    0    16    Auto Segment 3
```

Name: AUTO ROM — Type: ROM
Start: * — End: *
Wait state: 0

Bus width: ( ) 8 bits [✓ 16 bits] ( ) 32 bits

Access Details:
- [✓] Auto Configure
- [✓] Allow misaligned access

Access Details

	Read	Write
Byte :	0	0
Word :	0	0
Word misaligned :	0	0
Long :	0	0
Long misaligned 1 :	0	0
Long misaligned 2 :	0	0
Long misaligned 3 :	0	0

Follow this guidance to use the **Access Details** dialog box::

- To modify the value of each read or write type, change the value of the associated spin box.
- The lowest possible value is -1.
- The highest possible value is 100.
- To store changes into the currently selected memory block, click the **OK** button. The **Access Details** dialog box disappears, and the system clears the **Auto Configure** checkbox.
- To abandon your changes, click the **Cancel** button. The **Access Details** dialog box disappears; the system discards your changes.
- To bring up appropriate help information, click the **Help** button.

Output

You can save the current memory configuration into the file you defined at the outset.

Default Memory Configuration File

Listing 1.2 shows the format of the Default Memory Configuration File, in EBNF notation.

Listing 1.2 Format: Default Memory Configuration File.

```
memConfFile = head mode numberBlocks data
head = number
mode = STD_MODE | AUTO_MODE | MAN_MODE
numberBlocks = number
data = {memoryBlock}
memoryBlock = name type start end waitState busWidth accessRead
accessWrite autoConfigure allowMisalignedAccess blockType
name = string
type = string
start = number
end = number
waitState = number
busWidth = number
accessRead = array of number
accessWrite = array of number
autoConfigure = boolean
allowMisalignedAccess = boolean
blockType = USER_DEF | AUTO_GEN | AUTO_MEM | AUTO_RAM | AUTO_ROM
```

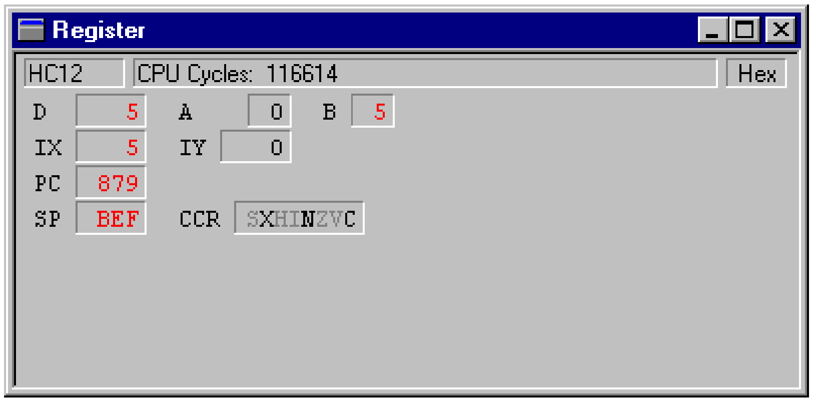
HC12 Simulator Specifics

This section introduces HC12 Simulator features.

HC12 Registers

The **Register Components** window (Figure 1.6) displays the HC12 processor registers. Values can be in any of five formats: hexadecimal, binary, octal, decimal, or unsigned decimal.

Figure 1.6 The HC12 Register Components Window



- D: 16-bit general accumulator value
- A and B: General 8-bit purpose accumulator register values
- IX and IY: Index register values
- PC: 16-bit program counter register value
- SP : 16-bit stack pointer register value
- CCR : 8-bit condition code register value

NOTE For more information, see the Motorola HC12 reference manual.

Special Environment Variable

If changes to local variables or parameters free any space on the stack, the system marks such space as undefined (current feature). The instruction LDS is an exception: if any byte allows initialization of the stack pointer without influencing the old or new stack pointer, the system does *not* mark that byte undefined.

If you wish to disable this feature, assign the value **OFF** to the environment variable **UNDEFSTACK**, in the `project.ini` file, or your defined project (`.ini`) file in the project directory.

Sample12 I/O Simulation

You can load this I/O Simulation component in HI-WAVE to simulate the I/O mechanisms of HC12 derivatives. **Sample12** is a free I/O component that includes advanced HC12-derivative features.

Introduction

This chapter explains the simulated mechanisms and implemented features that match the HC12 derivatives. It also explains simulation limitations. (For technical specifications of all I/O mechanisms, please see the Motorola Microcontrollers Technical Summary of your specific HC12 derivative.)

Simulated I/O

Register Block (A4, B32, D60, DA/DG128)

You can reassign the 1-kilobyte register block to any 2-kilobyte boundary within the standard 64-kilobyte address space.

Related register:

INITRG Initialization of Internal Register Position Register, simulated.

Memory Expansion Registers (A4, DA/DG128)

The system fully simulates the Program Page mechanism within CALL and RTC instructions for **banked memory model**; Data Page and Extra Page simulated.

NOTE Also see the Programming in Bank Windows section of this manual for application programs creation/adaptation.

Related register:

Program Page Registers DPAGE, PPAGE, EPAGE, and WINDEF (**sample12** is A4 oriented).

Lite Integration Module (A4, B32, D60, DA/DG128)

The LIM device contains the clock functions Computer Operating Properly (COP) and Real Time Interrupt (RTI).

Related registers:

CLKCTL:
The MCSA and MCSB bit of the Clock Control Register determine the clock that such I/O devices as the SCIs, RTI, and COP use. The system does not simulate the PLL functionalities.

bit 7..2 Not simulated.
bit 1 MCSBModule Clock Select, Simulated
bit 0 MCSAModule Clock Select, Simulated

RTICTL: Real Time Control Register

bit 7 RTIEReal Time Interrupt Enable, Simulated
bit 6 RSWAIRTI and COP Stop While in Wait, Not simulated
bit 5 RSBCKRTI and COP Stop While in BDM, Not simulated
bit 4 unused
bit 3 RTBYBReal Time Interrupt Divider Chain Bypass, Simulated
bit 2..0Real Time Interrupt Rate Select (RTR2..0), Simulated

RTIFLG: Real Time Interrupt Flag Register

bit 7 RTIFReal Time Interrupt Flag, Simulated
bit 6..0 unused

COPCTL: COP Control Register. Clock Monitor Not simulated.
bit 7 CMEClock Monitor Enable, Not simulated
bit 6 FCMEForce Clock Monitor Enable, Not simulated
bit 5 FCMForce Clock Monitor Reset, Not simulated
bit 4 FCOPForce COP Reset, Simulated
bit 3 DISRDisable Resets from COP and Clock Monitor, Simulated
bit 2..0 COP Watchdog Timer Rate select bits, Simulated

COPRST: The ARM/Reset COP TIMER Reset register is Simulated.

HPRIO: Highest Priority I Interrupt, Simulated

INTCR: Interrupt Control Register
bit 7 IRQEIRQ Select Edge Sensitive Only, Not simulated
bit 6 IRQENExternal IRQ Enable, not simulated
bit 5 DLYEnable Oscillator Start-up Delay, Not simulated

Serial Communication Interface (A4, B32, D60, DA/ DG128)

This I/O Device simulates the two SCI signals SCI0 and SCI1. The non-memory-mapped registers SCIInput/SCIInputH and SerialInput send characters to the SCI Module. The non-memory-mapped registers SCIOutput/SCIOutputH and SerialOutput contain the characters sent from to the SCI Module.

Related registers:

SC0BDH/SC1BDH: SCI Baud Rate Register High

bit 7 BTST Reserved for test functions, Not simulated

bit 6 BSPL Reserved for test functions, Not simulated

bit 5 BRLD Reserved for test functions, Not simulated

bit 4..0 SBR   SCI Baud Rate, Simulated

SC0BDL/SC1BDL: SCI Baud Rate Register Low

bit 7..0 SBR   SCI Baud Rate, Simulated

SC0CR1/SC1CR1: SCI Control Register 1

bit 7 LOOPS LOOP Mode, Not simulated

bit 6 WOMS Wired Or Mode, Not simulated

bit 5 RSRC Receiver Source, Not simulated

bit 4 M Mode, Simulated

bit 3 WAKE Wakeup by Address Mark/Idle, Not simulated

bit 2 ILT   Idle Line Type, Simulated

bit 1 PE Parity Enabled, Not simulated

bit 0 PT Parity Type, Not simulated

SC0CR2/SC1CR2: SCI Control Register 2

bit 7 TIE   Transmit Interrupt Enable, Simulated

bit 6 TCIE Transmit Complete Interrupt Enable, Simulated

bit 5 RIE   Receive Interrupt Enable, Simulated

bit 4 ILIE Idle Line Interrupt Enable, Simulated

bit 3 TE Transmitter Enable, Simulated

bit 2 RE Receiver Enable, Simulated

bit 1 RWU Receiver Wake Up Control, Not simulated

bit 0 SBK Send Break, Simulated

SC0SR1/SC1SR1: SCI Status Register 1

bit 7 TDRE Transmit Data Register Empty Flag, Simulated

bit 6 TC Transmit Complete Flag, Simulated

bit 5 RDRF Receive Data Register Full Flag, Simulated

bit 4 IDLE Idle Line Detection Flag, Simulated

bit 3 OR Overrun Error Flag, Simulated

bit 2 NF Noise Error Flag, Not simulated

bit 1 FE Framing Error Flag, Not simulated

bit 0 PF Parity Error Flag, Not simulated

SC0SR2/SC1SR2: SCI Status Register 2

bit 7..1 unused

bit 0 RAF Receiver Active Flag, Simulated

SC0DRH/SC1DRH: SCI Data Register High

bit 7 R8 Receive Bit 8, Simulated

bit 6 T8 Transmit Bit 8, Simulated

SC0DRL/SC1DRL: SCI Data Register Low, contains the Receive-/Transmit Data Bits 7..0.

SCIInput:

This is a non-memory-mapped register that sends a character to the SCI. A read access to the SCDR can read this value. The system takes the ninth bit from the SCIInputH register. A read access to SCIInput has no specified meaning.

bit 7..0 character send to the SCI

SCIInputH:

This is a non-memory-mapped register that sends a character, the ninth bit, to the SCI. You must write this register value before you write the SCIInput register value. A read access to SCIInputH has no specified meaning.

bit 7..1 unused

bit 0   ninth bit send to the SCI

SCIOutput:

This is a non-memory-mapped register that receives a character sent from the SCI. A write access to the SCDR triggers the value that the SCIOutput receives. The SCIOutputH register receives the nint bit. A write access to SCIOutput has no specified meaning.

bit 7..0   character send from the SCI

SCIOutputH:

This is a non-memory-mapped register that receives a character, the ninth bit, sent from the SCI. A write access to SCIOutput has no specified meaning.

bit 7..1   unused

bit 0   ninth bit send from the SCI

SerialInput:

This non-memory-mapped register is an alias for the SCIInput register. It connects the SCI to the terminal window, but does not support the ninth bit. A read access to SerialInput has no specified meaning.

bit 7..0   data from terminal window to SCI

SerialOutput:

This non-memory-mapped register is an alias for the SCIOutput register. It connects the SCI to the terminal window, but does not support the ninth bit. A write access to SerialOutput has no specified meaning.

bit 7..0   data sent from SCI to terminal window

Loading the I/O Simulation Component

You can load I/O Simulation components from within a command, from a HI-WAVE system command file (such as `STARTUP.CMD)`, or from any command file.

Use the command **OPENIO <ioname>** to load the I/O component in HI-WAVE. For example, writing:

```
OPENIO SAMPLE12
```

in `STARTUP.CMD` loads this I/O when you start HI-WAVE. Do not use the “.IO” extension.

Another way to load an I/O component is selecting `Simulator> Load IOs...`, from the HI-WAVE main menu, then choosing the I/O component from the list that appears.

Refer to the HI-WAVE main Manual for further details.

Programming in Bank Windows

Assembler Programming

If you program in assembler, implement your code in sections to be mapped to the appropriate page, in the `.PRM` file. Your source file code should have the structure that Listing 1.3 shows.

Listing 1.3 Example of Assembler Source Code for Programming in Bank Windows

```
XDEF Func1, Func2, main

Page1Code: section
Func1:
     ...
     RTS

Page2Code: section
Func2:
     ...
     RTS

UnpagedCode: section
main:
     ...
     CALL   Func1,PAGE(Func1)
     CALL   Func2,PAGE(Func2)
     ...
     ...
```

Assemble your file with the **Code Generation** option **Banked Memory Model**. As Listing 1.4 shows, the system places the Page1Code and Page2Code sections in the PAGE_1 and PAGE_2 bank windows of the `.PRM` file.

Listing 1.4 Example of Parameter File for Programming in Bank Windows

```
LINK my_appli.abs
NAMES
  my_appli.o
END

SECTIONS
  MY_RAM = READ_WRITE 0x2010 TO 0x23FF;
  MY_STK = READ_WRITE 0x2400 TO 0x24FF;
  NO_BANKED_ROM = READ_ONLY  0xC000 TO 0xFEFF;
  PAGE_1 = READ_ONLY 0x18000 TO 0x1BFFF;
  PAGE_2 = READ_ONLY 0x28000 TO 0x2BFFF;
PLACEMENT
  .data INTO MY_RAM;
  .text INTO NO_BANKED_ROM;
  .stack INTO MY_STK;
  Page1Code INTO PAGE_1;
  Page2Code INTO PAGE_2;
  UnpagedCode INTO NO_BANKED_ROM;
END
INIT main
VECTOR ADDRESS 0xFFFE main
```

C/C++ Programming

If you program in C/C++, compile your file with the **Code Generation** option **Banked Memory Model**, and link your application with the `ansib.lib` and `start12b.o` libraries (for the banked memory model). For C++, you also must link the `cppb.lib` library. Listing 1.5 shows a `.PRM` file for HC12DG128 application, where the default ROM is in page 2 and page 4, using the banked memory model. For any application, be sure to locate your code properly in a Flash address range.

Listing 1.5 Example PRM File for HC12DG128

```
LINK my_appli.abs

NAMES my_appli.o ansib.lib cppb.lib start12b.o END
SECTIONS
    MY_RAM = READ_WRITE 0x2010 TO 0x23FF;
    MY_ROM = READ_ONLY  0xC000 TO 0xFEFF;
    PAGE_2 = READ_ONLY 0x28000 TO 0x2BFFF;
    PAGE_4 = READ_ONLY 0x48000 TO 0x4BFFF;
PLACEMENT
```

```
    _PRESTART, STARTUP,
    ROM_VAR, STRINGS,
    NON_BANKED, COPY              INTO  MY_ROM;
    DEFAULT_RAM                   INTO  MY_RAM;
    MyPage, DEFAULT_ROM           INTO  PAGE_2, PAGE_4;
END
STACKSIZE 0x50
VECTOR ADDRESS 0xFFFE _Startup  /* set reset vector IN FLASH on _Startup
*/
```

Simulated I/O Ports of the MC68HC12A4 CPU

This section explains the simulated features of the MC68HC12A4 CPU in HIWAVE. The simulator implements all features according to [1].

Register Block

Table 1.3 shows the register block functionality. You can move all I/O registers, according to the INITRG (Register Block Mapping) at offset $11 inside of the register block.

Table 1.3 MC68HC12A4 Register Block

Register Name	Register Address	Initial Value	Remarks
INITRG	0x0011	0x00	

Lite Integration Module

The simulator simulates many functions of the Lite Integration Module (LIM), including:

- Interrupt handling
- Watchdog
- Periodic Interrupt

General restrictions:

- The simulator does not distinguish normal from special mode. Accordingly, it allows all write accesses, as if the chip were in special mode.
- Table 1.4 on page 24 includes restrictions relative to special registers and single bits of registers.

LIM Simulated Registers

Table 1.4 on page 24 shows the LIM Simulated Registers.

Table 1.4 LIM Simulated Registers

Register Name	Register Address	Initial Value	Remarks
CLKCTL	0x0047	0x00	LCKF, PLLON, PLLS, BCSC, BCSB, BCSA: These CLKCTL bits control settings of the PLL. But the simulator does not simulate the PLL, so values of these bits have no effect.
RTICTL	0x0014	0x00	RSWAI: The simulator does not support the CPU Clock stop, so this bit of the RTICTL register has no effect. RSBCK: The simulator does not simulate background mode, so this bit of the RTICTL register has no effect.
RTIFLG	0x0015	0x00	
COPCTL	0x0016	0x0F	CME, FCME, FCM: The simulator does not support these COPCTL bits; writing to these bits has no effect.
COPRST	0x0017	0x00	
INTCR	0x001E	0x60	The simulator does not distinguish normal from special mode. IRQE: The implementation allows any write access. In normal mode, there should be only one write to this register. In special mode, the system should ignore the first write access.
HPRIO	0x001F	0xF2	The system may write to the HPRIO register if the I mask in the CPU condition code register CCR is set. The simulator does not simulate this fact.

Standard Timer Module (TIM)

All functions of the timer module TIM are simulated.

General restrictions:

- The HPRIO register [$001F] may be written to if the I mask in the CPU condition code register CCR is set. This fact is not simulated.
- The external timer output occurs at the PORTT register. This is done for testing purposes only and will be disabled in future versions.

- Restrictions considering special registers and single bits of registers are mentioned in Table 1.5.

For descriptions of all simulated actions, see [1], chapter 13.

TIM Simulated Registers

Table 1.5 shows all TIM Simulated Registers

Table 1.5 TIM Simulated Registers

Register Name	Register Address	Initial Value	Remarks
TIOS	0x0080	0x00	
CFORC	0x0081	0x00	
OC7M	0x0082	0x00	
OC7D	0x0083	0x00	
TCNT_H	0x0084	0x00	
TCNT_L	0x0085	0x00	
TSCR	0x0086	0x00	TSWAI: The simulator does not support the CPU Clock stop, so setting this bit has no effect. TSBCK: The simulator does not simulate background mode, so this bit of the TSCR register has no effect.
TQCR	0x0087	0x00	
TCTL1	0x0088	0x00	
TCTL2	0x0089	0x00	
TCTL3	0x008A	0x00	
TCTL4	0x008B	0x00	
TMSK1	0x008C	0x00	
TMSK2	0x008D	0x30	TPU: This bit controls a pull-up resistor or a pin. But the simulator does not have real pins, so setting this bit has no effect. TDRB: This bit controls the output drive of a pin. But the simulator does not have real pins, so setting this bit has no effect.
TFLG1	0x008E	0x00	
TFLG2	0x008F	0x00	
TC0_H	0x0090	0x00	

Register Name	Register Address	Initial Value	Remarks
TC0_L	0x0091	0x00	
TC1_H	0x0092	0x00	
TC1_L	0x0093	0x00	
TC2_H	0x0094	0x00	
TC2_L	0x0095	0x00	
TC3_H	0x0096	0x00	
TC3_L	0x0097	0x00	
TC4_H	0x0098	0x00	
TC4_L	0x0099	0x00	
TC5_H	0x009A	0x00	
TC5_L	0x009B	0x00	
TC6_H	0x009C	0x00	
TC6_L	0x009D	0x00	
TC7_H	0x009E	0x00	
TC7_L	0x009F	0x00	
PACTL	0x00A0	0x00	
PAFLG	0x00A1	0x00	
PACNT_H	0x00A2	0x00	
PACNT_L	0x00A3	0x00	
TIMTST	0x00AD	0x00	TCBYP, PCBYP: The simulator does not support these TIMTST bits; writing to them has no effect. (These bits have meaning only for chip testing in special mode.)
PORTT	0x00AE	0x00	
DDRT	0x00AF	0x00	

Serial Communication Interface SCI

You should implement the SCI module as a separate class, because there are several almost-identical instances of this class.

Supported Features

Table 1.6 shows the SCI supported features.

Table 1.6 SCI Supported Features

Abbr	Full Name	Implemented Meaning
	Baud Rate Control	
SBRx	Baud Rate	Bit transmittal follows current baud rate settings
BTST	Reserved for internal tests	Ignored
BSPL	Reserved for internal tests	Ignored
BRLD	Reserved for internal tests	Ignored
	Control Register	
LOOP	LOOP Mode	The LOOP mode determines SCI connection to the outer world. As this SCI is simulated, there is no connection to simulate.
WOMS	Wired Or Mode	Special feature of LOOP mode, not simulated
RSRC	Receiver Source	Special feature of LOOP mode, not simulated
M Mode	8 or 9 data bits	Supported (different timing, 9th bit)
WAKE	Wakeup by Address Mark/ Idle	Not supported
ILT	Idle Line Type	Considered in the Idle Line Detection
PE	Parity Enabled	Not simulated
PT	Parity Type	Not simulated
TIE	Transmit Interrupt Enable	Supported
TCIE	Transmit Complete Interrupt Enable	Supported
RIE	Receive Interrupt Enable	Supported
ILIE	Idle Line Interrupt Enable	Supported
TE	Transmitter Enable	Transmission process stops if this bit is clear
RE	Receiver Enable	Receive process stops if this bit is clear. As the input register is not part of the simulation, it still receives stimuli.
RWU	Receiver Wake Up Control	Not supported
SBK	Send Break	Upon the first set of the SBK Flag, the transmitter starts sending 10 (11 if M bit is set) 0 values. The counter will be set only if the flag was cleared previously. After the counter sends the required number of 0 bits, it continues send 0 bits as long as the SBK flag remains set.
	Status Registers	
TDRE	Transmit Data Register Empty Flag	The system sets this flag upon the move of the value to be transmitted from the transmit data register to the serial shift register.

Abbr	Full Name	Implemented Meaning
TC	Transmit Complete Flag	The system sets this flag if the transmission of one value ends, but no other value is yet in the transmit data register.
RDRF	Receive Data Register Full Flag	The system sets this flag upon the complete read of a value and the clearing of RDRF.
IDLE	Idle Line Detection Flag	The system sets this flag after a period without any input as stated in [3]. The system considers the ILT flag.
OR	Overrun Error Flag	The system sets this flag if the receipt of value ends, but the processor has not yet read the value.
NF	Noise Error Flag	Not supported, as no physical transmission takes place.
FE	Framing Error Flag	Not supported, as no physical transmission takes place.
PF	Parity Error Flag	Not supported, as no physical transmission takes place.
RAF	Receiver Active Flag	Supported and cleared only when going into idle mode. Detection of a false start bit does not clear this flag, as no physical transmission takes place.
	Data Register	
R8	Receive Bit 8	Supported
T8	Transmit Bit 8	Supported
Rx/Tx	Receive/Transmit Bit x	Supported, with autoclear feature

The simulator use non-memory-mapped registers to simulate SCI connection to the outer world. The simulator buffers all values sent to the input registers, then simulates receipt from another SCI (with maximum speed and no transmission errors). If the buffer contains no values, the simulator simulates an empty input line. All these sent values are available in the output registers, which Table 1.7 on page 28 lists. Other modules can subscribe to these registers to receive the sent values.

Table 1.7 Input, Output, Serial Output Registers

Name	Meaning	Comment
Input	Adds a value to be received. The system takes the 9th bit from the last value written to InputH. Read has no specified meaning	
InputH	9th Input bit; must be written before Input. Read has no specified meaning	
Output	Contains the last value sent. A notification is sent every time a new value is written. Write has no specified meaning	
OutputH	9th Output bit. Must be read immediately after Output. Write has no specified meaning	

Name	Meaning	Comment
SerialInput	Alias for Input for SCI 0; connects SCI 0 to terminal window. Only supports 8 bits.	Only available in SCI 0.
SerialOutput	Alias for Output for SCI 0; connects SCI 0 to terminal window. Only support 8 bits.	Only available in SCI 0

Serial Peripheral Interface SPI

Table 1.8 describes the SPI interface.

Table 1.8 SPI interface

Abbr.	Full Name	Implemented Meaning
	Control Register 1	
SPIE	Interrupt Enable	Implemented
SPE	System Enable	If set, the simulator supports SPI functions
SWOM	Port S Wired-OR Mode	Not simulated, as no physical transmission takes place.
MSTR	Master Slave Mode Select	Master or Slave mode select
CPOL	Clock Polarity	Not simulated, as no physical transmission takes place.
CPHA	Clock Phase	Not simulated, as no physical transmission takes place.
SSOE	Slave Select Output Enable	Not simulated, as no physical transmission takes place.
LSBF	LSB First Enable	Not simulated, as no physical transmission takes place.
	Control Register 2	
PUPS	Pull Up Port S Enable	Not simulated, as no physical transmission takes place.
RDS	Reduce Drive of Port S	Not simulated, as no physical transmission takes place.
SPC0	Serial Pin Control 0	Selects Normal or Bidirectional transmission mode
	Baud Rate Register	
SPRx	Baud Rate Register	Baud rate of the SPI transmission
	Status Register	
SPIF	Interrupt Request	System sets SPIF after the eighth SCK cycle in a data transfer and clearing by reading the Status Register, followed by a read or write access to the SPI data register.
WCOL	Write Collision Status Register	System sets this flag upon the writing of new data to the Data Register, during a serial data transfer.
MODF	Mode Error Interrupt Status Flag	Not simulated, as no physical transmission takes place.

Abbr.	Full Name	Implemented Meaning
	Data Register	
SP0DR		8-bit Data Register for SPI data.
	Port S	
PORTS	Port S Data Register	Not simulated, as no physical transmission takes place.
	Data Direction Register	
DDRSx	Data Direction for Port S Bit x	Direction of Data. Only bits 4 and 5 have any effect.

Virtual register Value simulates the data register of a second SPI device. This permits simulate communication with a second SPI device. The transmission can be in Normal or a Bidirectional Mode; the device can be set as Master or Slave. See also “Technical Summary MC68HC812A4” page 84, figure 24.

Key Wakeups

Table 1.9 on page 30 defines the Key Wakeups.

Table 1.9 Key Wakeups

Abbr.	Full Name	Implemented Meaning
	Key Wakeups Registers	
PORTD	Port D Register	Implemented
DDRD	Port D Data Direction Register	Implemented
KWIED	Port D Interrupt Enable Register	Implemented
KWIFD	Port D Flag Register	A falling edge on the associated pin sets each flag, provided that the corresponding DDRD Register bit is reset. To clear the flag, write one to the corresponding bit of the KWIFD register.
PORTH	Port H Register	Implemented
DDRH	Port H Data Direction Register	Implemented
KWIEH	Port H Interrupt Enable Register	Implemented
KWIFH	Port H Flag Register	A falling edge on the associated pin sets each flag, provided that the corresponding DDRH Register bit is reset. To clear the flag, write one to the corresponding bit of the KWIFH register.

Abbr.	Full Name	Implemented Meaning
PORTJ	Port J Register	Implemented
DDRJ	Port J Data Direction Register	Implemented
KWIEJ	Port J Interrupt Enable Register	Implemented
KWIFJ	Port J Flag Register	A falling edge on the associated pin sets each flag, provided that the corresponding DDRJ Register bit is reset. To clear the flag, write one to the corresponding bit of the KWIFJ register.
KPOLJ	Port J Polarity Register	Implemented
PUPSJ	Port J Pull-Up/Pulldown Select Register	Not simulated, as there are no physical outputs.
PULEJ	Port J Pull-Up/Pulldown Enable Register	Not simulated, as there are no physical outputs.

The simulator does not implement Port-D register mapping in wide expanded modes. The simulator does not implement this mapping in special expanded narrow mode with MODE Register bit EMD set.

Memory-Mapped Page Registers

Table 1.10 describes the Memory-Mapped Page Registers.

Table 1.10 Memory Mapped Page Registers

Abbr.	Full Name	Implemented Meaning
	Port F Register	
CS	Chip Select / General Purpose IO (Bit 0-6)	Not implemented, as there are no physical outputs.
	Port G Register	
ADDR	Memory Expansion / General Purpose IO (Bit 0-5)	Not implemented, as there are no physical outputs.
	Port F Data Direction Register	
DDRF	Data Direction Register Port F (Bit 0-6)	Not implemented, as there are no physical outputs.
	Port G Data Direction Register	
DDRG	Data Direction Register Port G (Bit 0-5)	Not implemented, as there are no physical outputs.
	Data Page Register	
PDA	Data Page	Selects the data page

Abbr.	Full Name	Implemented Meaning
	Program Page Register	
PPA	Program Page	Selects the program page
	Extra Page Register	
PEA	Extra Page	Selects the extra page
	Window Defition Register	
DWEN	Data Window Enable	Enables paging of data space
PWEN	Program Window Enable	Enables paging of program space
EWEN	Extra Window Enable	Enables paging of extra space
	Memory Expansion Assignment Register	
A21E-A16E	Memory Expansion Assignment/ General Purpose IO	Not simulated, as there are no physical outputs.

Current Non-Supported Modules

Non-Supported Modules

- A/D Converter Device

Register Block Address Map

Table 1.11 shows the mapping of the Register Block Address.

Table 1.11 Register Block Address Map

Register Block Address	Description	Remarks
\$0000-\$000D	Port access	Not simulated: memory configuration controls correct timing of memory accesses
\$000E-\$000F	Reserved	
\$0010	Internal RAM mapping	Register not simulated. Use the memory configuration dialog box to specify simulated memory configuration.
0x0011	Register Block mapping	Completely simulated
\$0012-\$0013	ROM/EEPROM mapping	Registers not simulated. Use the memory configuration dialog box to specify simulated memory configuration.
\$0014-\$0017	Clock Function Control	Completely simulated

Register Block Address	Description	Remarks
$001E-$001F	Interrupt Control & Highest Priority I Interrupt	Completely simulated
$0020-$002E	Key Wakeup Control	Completely simulated
$002F	Reserved	
$0030-$0033	Port Registers	Currently not simulated
$0034-$0038	PAGE & memory configuration Registers	Page Registers are simulated
$0039-$003B	Reserved	
$003C-$003F	Chip select control registers	Currently not simulated
$0040-$0043	PLL divider registers	Currently not simulated
$0044-$0046	reserved	
$0047	Clock Control Register	Completely simulated
$0048-$005F	Reserved	
$0060-$0069	Analog to Digital Converter	Currently not simulated
$006A-$006E	Reserved	
$006F	PORTAD	Currently not simulated
$0070-$007F	ADRxH/reserved	Currently not simulated
$0080-$009F	Timer Registers	Completely simulated
$00A0-$00A3	Pulse Accumulator Control Registers	Completely simulated
$00A4-$00AC	Reserved	
$00AD-$00AF	Timer Test, Timer Port	Completely simulated
$00B0-$00BF	reserved	
$00C0-$00C7	SCI0	Completely simulated
$00C8-$00CF	SCI1	Completely simulated
$00D0-$00D3	SPI	Completely simulated
$00D4	Reserved	
$00D5-$00D7	SPI, PORTS	Completely simulated
$00D8-$00EF	Reserved	
$00F0-$00F3	EEPROM Control	Currently not simulated

Register Block Address	Description	Remarks
$00F3-$01FF	Reserved	

Related Documentation

The following documents are available from Motorola:

- MOTOROLA SEMICONDUCTOR TECHNICAL DATA, MC68HC812A4, Technical Summary 16-Bit Microcontroller 1996
- CPU12 Reference Manual, Preliminary draft 15 July 95, AMCU Division, 1995, MOTOROLA

I/O Simulation HC12DA128 / HC12DG128

Introduction

You can load this I/O Simulation component in HI-WAVE to simulate the I/O mechanisms of HC12 derivatives HC12DA128 / HC12DG128.

This chapter explains derivative simulated mechanisms and implemented features that match the real HC12 derivatives. It also explains simulation limitations. (For technical specifications of all I/O mechanisms, please see ***MOTOROLA MC68HC912DA128/ MC68HC912DG128***
16-Bit Microcontroller Technical Summary from MOTOROLA INC., 1997, 27 August 1997, rev1.0.)

Simulated HC12DA/DG128 I/O

Register Block

You can reassign the 1-kilobyte register block to any 2-kilobyte boundary within the standard 64-kilobyte address space.

Related register:

INITRG Initialization of Internal Register Position Register, simulated.

Memory Expansion Register

The system fully simulates this mechanism within CALL and RTC instructions for **banked memory model**.

NOTE Also see the **Programming in Bank Windows** section of this manual for application programs creation/adaptation.

Related register:

Program Page Register PPAGE: PIX2/PIX1/PIX0 bits memory defined but NOT updated for HI_WAVE 5.x version of this I/O.

Enhanced Capture Timer

16-Bit Modulus Down-Counter Simulated.

8 Input Capture/Output Compare channels: all channels are **NON-BUFFERED** and identical, except channel 7 with TCRE (Timer Counter Reset Enable) also implemented.

You may configure **PORTT** pins individually as standard, parallel-port I/O pins, or as timer pins. For standard parallel I/O pins, reading and writing are transparen, behavin like reading/writing in typical RAM. For this configuration, assign the value 1 to the channel x bit IOSx, in the TIOS register (for compare mode). Assign the value 0 to the OMx and OLx bits of the TCL1 or TCTL2 register for **Timer disconnected from output pin logic** mode/output action.

Capture Stimulation on PORTT. You can simulate rising- and falling-edge input signals on PPORT with the Stimulat component (I/O Stimulation). In this case, PORTT is bit accessible via non-memory-mapped I/O registers PORTTBit0 through PORTTBit7.

The stimulation example below periodically stimulates the PORTT bit 5 to simulate an input capture.

```
def a = TIMER.PORTTBit5;
PERIODICAL 4000, 500:
   1000 a = 1;
   3000 a = 0;
END
```

Other user-designed I/O components also can set the PORTT bit value. Use **OP_SetValue**(“**RegisterBlock.PORTTBit5”,&parameter, NO_UPDATE);** function (with **parameter.n** = 0 | 1).

16-Bit Modulus Down-Counter

Related registers:

MCCTL: (16-bit modulus down counter control register) All bits simulated except ICLAT bit.

MCCNT: (modulus down-counter count register) Fully simulated.

Capture / Compare Timer

TIOS: (timer input capture/output compare select) Simulated.

CFORC: (timer compare force register) Simulated.

TCNT: (timer count register) Simulated.

TCTL1 and **TCTL2:** (timer control register - output) Simulated.

TCTL3 and **TCTL4:** (timer control register - input) Simulated.

TMSK1: (timer interrupt mask) Simulated.

TMSK2: (timer interrupt mask) Simulated bits: TOI (overflow interrupt), TCRE (timer counter reset enable), PR2,PR1,PR0 (prescaler)

TFLG1: (main timer interrupt flag) Simulated.

TFLG2: (main timer interrupt flag) Simulated.

TC0 to **TC7:** (timer input capture/output compare registers) Simulated.

Serial Communication Interface (SCI)

This I/O Device simulates the two SCI signals SCI0 and SCI1. The non-memory-mapped registers SCIInput/SCIInputH and SerialInput send characters to the SCI

Module. The non-memory-mapped registers SCIOutput/SCIOutputH and SerialOutput contain the characters sent from the SCI Module.

Related registers:

SC0BDH/SC1BDH: SCI Baud Rate Register High

bit 7 BTST Reserved for test functions, Not simulated

bit 6 BSPL Reserved for test functions, Not simulated

bit 5 BRLD Reserved for test functions, Not simulated

bit 4..0 SBR   (SCI Baud Rate) Simulated

SC0BDL/SC1BDL: SCI Baud Rate Register Low

bit 7..0 SBR   SCI Baud Rate, Simulated

SC0CR1/SC1CR1: SCI Control Register 1

bit 7 LOOPS LOOP Mode, Not simulated

bit 6 WOMS Wired Or Mode, Not simulated

bit 5 RSRC Receiver Source, Not simulated

bit 4 M Mode, Simulated

bit 3 WAKE Wakeup by Address Mark/Idle, Not simulated

bit 2 ILT   Idle Line Type, Simulated

bit 1 PE Parity Enabled, Not simulated

bit 0 PT Parity Type, Not simulated

SC0CR2/SC1CR2: SCI Control Register 2

bit 7 TIE   Transmit Interrupt Enable, Simulated

bit 6 TCIE Transmit Complete Interrupt Enable, Simulated

bit 5 RIE   Receive Interrupt Enable, Simulated

bit 4 ILIE Idle Line Interrupt Enable, Simulated

bit 3 TE Transmitter Enable, Simulated

bit 2 RE Receiver Enable, Simulated

bit 1 RWU   Receiver Wake Up Control, Not simulated

bit 0 SBK   Send Break, Simulated

SC0SR1/SC1SR1: SCI Status Register 1

bit 7 TDRE Transmit Data Register Empty Flag, Simulated

bit 6 TC Transmit Complete Flag, Simulated

bit 5 RDRF Receive Data Register Full Flag, Simulated

bit 4 IDLE Idle Line Detection Flag, Simulated

bit 3 OR Overrun Error Flag, Simulated

bit 2 NF Noise Error Flag, Not simulated

bit 1 FE Framing Error Flag, Not simulated

bit 0 PF Parity Error Flag, Not simulated

SC0SR2/SC1SR2: SCI Status Register 2

bit 7..1   unused

bit 0 RAF   Receiver Active Flag, Simulated

SC0DRH/SC1DRH: SCI Data Register High

bit 7 R8 Receive Bit 8, Simulated

bit 6 T8 Transmit Bit 8, Simulated

SC0DRL/SC1DRL: SCI Data Register Low, contains the Receive-/Transmit Data Bits 7..0.

SCIInput:

This is a non-memory-mapped register that sends a character to the SCI. A read access to the SCDR can read this value. The system takes the ninth bit from the SCIInputH register. A read access to SCIInput has no specified meaning.

bit 7..0   character send to the SCI

SCIInputH:

This is a non-memory-mapped register that sends a character, the ninth bit, to the SCI. You must write this register value before you write the SCIInput register value. A read access to SCIInputH has no specified meaning.

bit 7..1   unused

bit 0   ninth bit send to the SCI

SCIOutput:

This is a non-memory-mapped register that receives a character sent from the SCI. A write access to the SCDR triggers the value that the SCIOutput receives. The SCIOutputH register receives the ninth bit. A write access to SCIOutput has no specified meaning.

bit 7..0   character send from the SCI

SCIOutputH:

This is a non-memory-mapped register that receives a character, the ninth bit, sent from the SCI. A write access to SCIOutput has no specified meaning.

bit 7..1   unused

bit 0   ninth bit send from the SCI

SerialInput:

This non-memory-mapped register is an alias for the SCIInput register. It connects the SCI to the terminal window, but does not support the ninth bit. A read access to SerialInput has no specified meaning.

bit 7..0   data from terminal window to SCI

SerialOutput:

This non-memory-mapped register is an alias for the SCIOutput register. It connects the SCI to the terminal window, but does not support the ninth bit. A write access to SerialOutput has no specified meaning.

bit 7..0   data sent from SCI to terminal window

Displaying Special Registers

To visualize registers that the debugger **Register** component does not display, use either of two methods:

- Displaying Registers with the RD Command.
- Displaying Registers with the Visualization Tool Component.

Displaying Registers with the RD Command.

Open the debugger **Command Line** component and use the **RD** command:

Example:

```
in > RD PC
in>PC=0x450
```

NOTE For more information about the RD command, please see to the True Time Simulator and real Time Debugger manual.

Displaying Registers with the Visualization Tool Component.

Open the debugger **Visualization Tool**, then:

1. Create a new instrument (such as, value as text)
2. Set kind of port to Register
3. Set port to display to your register (for example, PC)

NOTE For more information about the Visualization Tool, please see to the Visualization Tool manual.

Index