Core8051s v2.4 Handbook

Actel
POWER MATTERS

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200084-2

Release: September 2010

Trademarks

Table of Contents

Introduction

This document describes the architecture of a small, general-purpose processor, called the Core8051s. This processor is compatible with the instruction set of the 8051 microcontroller, and preserves the three distinct software memory spaces so that it may be targeted by existing 8051 C compilers. To make it smaller and more flexible than the 8051, the following microcontroller-specific features of the original 8051 are not present:

- SFR-mapped peripherals
- Power management circuitry
- Serial channel
- I/O ports
- Timers

The following set of 8051 microcontroller features are available in Core8051s, but are either optional or reduced in scope:

- Multiply and divide instructions (MUL, DIV, and DA) – present by default, but may optionally be implemented as NOPs
- Second data pointer (data pointer 1) – not enabled by default
- Of the 64 kbytes allocated to external data memory, 4 kbytes are allocated to an APB-based peripheral bus and 60 kbytes is allocated to an external data memory interface
- Interrupt control logic for 2 interrupts

Supported Actel FPGA Families for the Core8051s are as follows:

- IGLOO®/e/PLUS
- ProASIC3®/E/L
- Fusion
- ProASIC PLUS®
- Axcelerator®
- RTAX-S

Utilization and Performance

Table 1 through Table 7 on page 13 give resource usage and performance data for various configurations of Core8051s for each type of FPGA technology. These tables do not cover every possible configuration, but instead list a range of configurations which should give a good indication of the expected resource usage and performance of the core. Abbreviated versions of configuration options are used in the tables to aid readability. The meanings of the entries in the debug, program memory access control, data memory access control, and internal RAM columns are described in the following paragraphs.

Debug Column

- None: Debug logic is not included.
- I/Os: Debug logic is included and general purpose I/Os are used for the debug connection.
- UJTAG: Debug logic is included and the dedicated JTAG pins of the device and the UJTAG macro are used for the debug connection.

Program Memory Access Control

- ACK: Acknowledge signal (MEMPSACKI) is used to control access to program memory.
- X: X (where X can range from 0 to 7) wait states are inserted in each access to program memory, instead of using acknowledge control.

Data Memory Access Control

- ACK: Acknowledge signal (MEMACKI) is used to control accesses to data memory.
- X: X (where X can range from 0 to 7) wait states are inserted in each access to data memory, instead of using acknowledge control.

Internal RAM

- Instantiated: Internal 256x8 RAM is implemented using an instantiated RAM block.
- Inferred: Internal 256x8 RAM is implemented by inferring RAM during synthesis.

Registers

Registers (FPGA tiles) are inferred for the 256x8 RAM during synthesis.

Table 1 • **Core8051s Utilization and Performance for IGLOO 1.2 V Devices (STD speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	3,435	1	14.8
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	3,833	1	14.9
ujtag	No	0	No	Yes	ack	ack	32	Instantiated	3,792	1	14.4
ujtag	No	1	No	Yes	ack	ack	32	Instantiated	4,080	1	14.7
ujtag	No	4	No	Yes	ack	ack	32	Instantiated	5,029	1	14.4
ujtag	Yes	0	No	Yes	ack	ack	32	Instantiated	3,974	3	15.4
ujtag	Yes	1	No	Yes	ack	ack	32	Instantiated	4,455	3	14.5
ujtag	Yes	4	No	Yes	ack	ack	32	Instantiated	5,538	3	14.6
None	–	–	Yes	Yes	ack	ack	32	Instantiated	3,686	1	14.9
None	–	–	No	Yes	2	2	32	Instantiated	3,376	1	14.8
None	–	–	No	Yes	5	5	32	Instantiated	3,308	1	15.3
None	–	–	No	Yes	ack	ack	16	Instantiated	3,311	1	15.1
None	–	–	No	Yes	ack	ack	8	Instantiated	3,318	1	15.2
None	–	–	No	Yes	ack	ack	32	Inferred	3,457	1	14.7
None	–	–	No	Yes	ack	ack	32	Registers	7,853	0	13.9
ujtag	Yes	4	Yes	Yes	ack	ack	32	Registers	10,098	2	12.1
None	–	–	No	No	ack	ack	8	Instantiated	2,849	1	14.7

Table 2 • **Core8051s Utilization and Performance for IGLOO 1.5 V Devices (STD speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	3,110	1	23.9
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	3,548	1	22.7
ujtag	No	0	No	Yes	ack	ack	32	Instantiated	3,483	1	24.3
ujtag	No	1	No	Yes	ack	ack	32	Instantiated	3,772	1	23.6
ujtag	No	4	No	Yes	ack	ack	32	Instantiated	4,847	1	23.3
ujtag	Yes	0	No	Yes	ack	ack	32	Instantiated	3,742	3	22.9
ujtag	Yes	1	No	Yes	ack	ack	32	Instantiated	4,083	3	23.9
ujtag	Yes	4	No	Yes	ack	ack	32	Instantiated	5,125	3	23.8
None	–	–	Yes	Yes	ack	ack	32	Instantiated	3,318	1	24.2
None	–	–	No	Yes	2	2	32	Instantiated	3,386	1	24.2
None	–	–	No	Yes	5	5	32	Instantiated	3,357	1	22.9
None	–	–	No	Yes	ack	ack	16	Instantiated	2,995	1	24.5
None	–	–	No	Yes	ack	ack	8	Instantiated	2,915	1	23.9
None	–	–	No	Yes	ack	ack	32	Inferred	3,136	1	24.8
None	–	–	No	Yes	ack	ack	32	Registers	7,633	0	23.3
UJTAG	Yes	4	Yes	Yes	ack	ack	32	Registers	9,917	2	19.9
None	–	–	No	No	ack	ack	8	Instantiated	2,568	1	23.8

Table 3 • **Core8051s Utilization and Performance for Fusion, ProASIC3, and ProASIC3E Devices (–2 speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	3,324	1	37.1
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	3,776	1	36.5
ujtag	No	0	No	Yes	ack	ack	32	Instantiated	3,758	1	35.9
ujtag	No	1	No	Yes	ack	ack	32	Instantiated	4,024	1	37.5
ujtag	No	4	No	Yes	ack	ack	32	Instantiated	4,941	1	35.4
ujtag	Yes	0	No	Yes	ack	ack	32	Instantiated	4,053	3	37.1
ujtag	Yes	1	No	Yes	ack	ack	32	Instantiated	4,262	3	36.7
ujtag	Yes	4	No	Yes	ack	ack	32	Instantiated	5,330	3	36.4
None	–	–	Yes	Yes	ack	ack	32	Instantiated	3,546	1	39.9
None	–	–	No	Yes	2	2	32	Instantiated	3,356	1	35.9
None	–	–	No	Yes	5	5	32	Instantiated	3,335	1	37.9
None	–	–	No	Yes	ack	ack	16	Instantiated	3,190	1	38.7
None	–	–	No	Yes	ack	ack	8	Instantiated	3,081	1	36.9
None	–	–	No	Yes	ack	ack	32	Inferred	3,384	1	37.5
None	–	–	No	Yes	ack	ack	32	Registers	7,739	0	35.9
ujtag	Yes	4	Yes	Yes	ack	ack	32	Registers	9,937	2	28.9
None	–	–	No	No	ack	ack	8	Instantiated	2,748	1	37.3

Table 4 • **Core8051s Utilization and Performance for ProASIC3L (–1 speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	2,936	1	25.7
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	3,360	1	25.4
ujtag	No	0	No	Yes	ack	ack	32	Instantiated	3,261	1	25.1
ujtag	No	1	No	Yes	ack	ack	32	Instantiated	3,624	1	23.9
ujtag	No	4	No	Yes	ack	ack	32	Instantiated	4,637	1	24.5
ujtag	Yes	0	No	Yes	ack	ack	32	Instantiated	3,541	3	25.5
ujtag	Yes	1	No	Yes	ack	ack	32	Instantiated	3,844	3	24.4
ujtag	Yes	4	No	Yes	ack	ack	32	Instantiated	4,926	3	24.7
None	–	–	Yes	Yes	ack	ack	32	Instantiated	3,116	1	24.2
None	–	–	No	Yes	2	2	32	Instantiated	2,931	1	24.5
None	–	–	No	Yes	5	5	32	Instantiated	2,928	1	26.5
None	–	–	No	Yes	ack	ack	16	Instantiated	2,778	1	26.4
None	–	–	No	Yes	ack	ack	8	Instantiated	2,718	1	25.5
None	–	–	No	Yes	ack	ack	32	Instantiated	2,943	1	23.6
None	–	–	No	Yes	ack	ack	32	Instantiated	7,391	0	25.4
ujtag	Yes	4	Yes	Yes	ack	ack	32	Instantiated	9,755	2	23.1
None	–	–	No	Yes	ack	ack	8	Instantiated	2,444	1	24.9

Table 5 • **Core8051s Utilization and Performance for ProASIC^{PLUS} Devices (STD speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	4,000	1	26.2
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	4,318	1	26.4
UJTAG	No	0	No	Yes	ack	ack	32	Instantiated	4,271	1	25.8
UJTAG	No	1	No	Yes	ack	ack	32	Instantiated	4,709	1	25.7
UJTAG	No	4	No	Yes	ack	ack	32	Instantiated	6,004	1	24.4
UJTAG	Yes	0	No	Yes	ack	ack	32	Instantiated	4,580	4	26.8
UJTAG	Yes	1	No	Yes	ack	ack	32	Instantiated	5,065	4	23.5
UJTAG	Yes	4	No	Yes	ack	ack	32	Instantiated	6,368	4	23.5
None	-	–	Yes	Yes	ack	ack	32	Instantiated	4,344	1	26.8
None	-	–	No	Yes	2	2	32	Instantiated	4,185	1	27.8
None	-	–	No	Yes	5	5	32	Instantiated	4,135	1	29.8
None	-	–	No	Yes	ack	ack	16	Instantiated	3,821	1	28.1
None	-	–	No	Yes	ack	ack	8	Instantiated	3,773	1	28.9
None	-	–	No	Yes	ack	ack	32	Inferred	4,056	1	26.2
None	-	–	No	Yes	ack	ack	32	Registers	10,888	0	25.2
UJTAG	Yes	4	Yes	Yes	ack	ack	32	Registers	13,652	3	19.9
None	–	–	No	No	ack	ack	8	Instantiated	3,146	1	28.8

Table 6 • **Core8051s Utilization and Performance for Axcelerator Devices (–2 speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	2,111	1	50.9
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	2,343	1	44.9
I/Os	No	1	No	Yes	ack	ack	32	Instantiated	2,608	1	42.4
I/Os	No	4	No	Yes	ack	ack	32	Instantiated	3,197	1	44.1
I/Os	Yes	0	No	Yes	ack	ack	32	Instantiated	2,554	3	47.4
I/Os	Yes	1	No	Yes	ack	ack	32	Instantiated	2,797	3	44.1
I/Os	Yes	4	No	Yes	ack	ack	32	Instantiated	3,413	3	42.5
None	–	–	Yes	Yes	ack	ack	32	Instantiated	2,196	1	53.4
None	–	–	No	Yes	2	2	32	Instantiated	2,091	1	55.8
None	–	–	No	Yes	5	5	32	Instantiated	2,104	1	54.2
None	–	–	No	Yes	ack	ack	16	Instantiated	2,066	1	53.3
None	–	–	No	Yes	ack	ack	8	Instantiated	1,977	1	56.3
None	–	–	No	Yes	ack	ack	32	Inferred	2,104	1	50.1
None	–	–	No	Yes	ack	ack	32	Registers	5,245	0	42.9
I/Os	Yes	4	Yes	Yes	ack	ack	32	Registers	6,714	2	33.1
None	–	–	No	No	ack	ack	8	Instantiated	1,757	1	53.4

Table 7 • **Core8051s Utilization and Performance for RTAX-S Devices (–1 speed grade)**

Configuration									Utilization and Performance		
Debug	Include Trace RAM	Hardware Triggers	Include Second Data Pointer	Include MUL, DIV, and DA Instructions	Program Memory Access Control	Data Memory Access Control	APB Data Width	Internal RAM	Tiles	RAM Blocks	MHz
None	–	–	No	Yes	ack	ack	32	Instantiated	2,123	1	39.9
I/Os	No	0	No	Yes	ack	ack	32	Instantiated	2,357	1	33.1
I/Os	No	1	No	Yes	ack	ack	32	Instantiated	2,607	1	30.1
I/Os	No	4	No	Yes	ack	ack	32	Instantiated	3,137	1	28.6
I/Os	Yes	0	No	Yes	ack	ack	32	Instantiated	2,547	3	29.9
I/Os	Yes	1	No	Yes	ack	ack	32	Instantiated	2,836	3	33.6
I/Os	Yes	4	No	Yes	ack	ack	32	Instantiated	3,351	3	26.5
None	–	–	Yes	Yes	ack	ack	32	Instantiated	2,192	1	39.7
None	–	–	No	Yes	2	2	32	Instantiated	2,057	1	37.8
None	–	–	No	Yes	5	5	32	Instantiated	2,118	1	38.4
None	–	–	No	Yes	ack	ack	16	Instantiated	2,042	1	39.6
None	–	–	No	Yes	ack	ack	8	Instantiated	1,987	1	39.4
None	–	–	No	Yes	ack	ack	32	Inferred	2,146	1	38.7
None	–	–	No	Yes	ack	ack	32	Registers	5,224	0	29.2
I/Os	Yes	4	Yes	Yes	ack	ack	32	Registers	6,694	2	22.8
None	–	–	No	No	ack	ack	8	Instantiated	1,778	1	39.8

1 – Core8051s Overview

The Core8051s is a high-performance, eight-bit microcontroller IP Core. It is a fully functional eight-bit embedded controller that executes all ASM51 instructions and has the same instruction set as the 80C31. Core8051s provides software and hardware interrupts.

The Core8051s architecture eliminates redundant bus states and implements parallel execution of fetch and execution phases. Since a cycle is aligned with memory fetch when possible, most of the one-byte instructions are performed in a single cycle. Core8051s uses one clock per cycle. This leads to an average performance improvement rate of 8.0 (in terms of MIPS) with respect to the Intel device working with the same clock frequency.

The original Intel 8051 had a 12-clock architecture. A machine cycle needed 12 clocks, and most instructions were either one or two machine cycles. Therefore, the 8051 used either 12 or 24 clocks for each instruction, except for the MUL and DIV instructions. Furthermore, each cycle in the 8051 used two memory fetches. In many cases, the second fetch was a “dummy” fetch and extra clocks were wasted.

Table 1-1 shows the speed advantage of Core8051s over the standard Intel 8051. A speed advantage of 12 in the first column means that Core8051s performs the same instruction 12 times faster than the standard Intel 8051. The second column in Table 1-1 lists the number of types of instructions that have the given speed advantage. The third column lists the total number of instructions that have the given speed advantage. The third column can be thought of as a subcategory of the second column. For example, there are two types of instructions that have a three-time speed advantage over the classic 8051, for which there are nine explicit instructions.

***Table 1-1* • Core8051s Speed Advantage Summary**

Speed Advantage	Number of Instruction Types	Number of Instructions (Opcodes)
24	1	1
12	27	83
9.6	2	2
8	16	38
6	44	89
4.8	1	2
4	18	31
3	2	9
Average: 8.0	Sum: 111	Sum: 255

2 – Supported Interfaces

Ports

The port signals of Core8051s are illustrated in Figure 2-1.

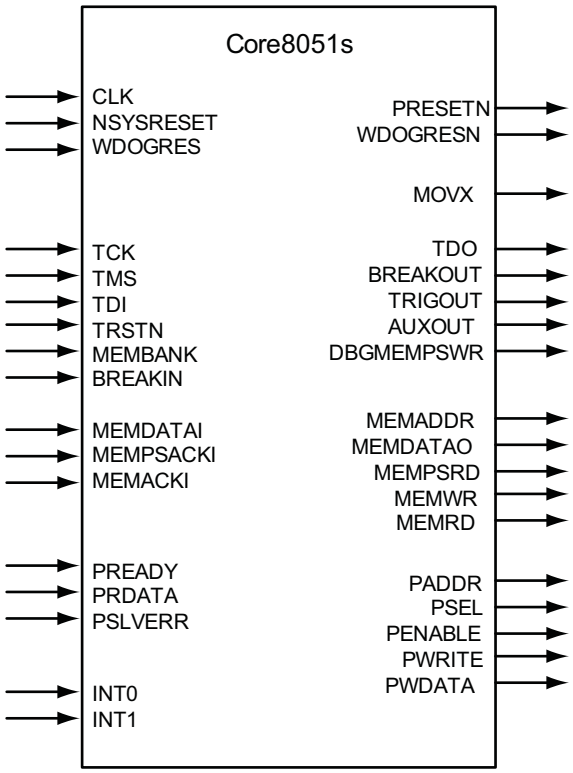

***Figure 2-1* • Core8051s I/O Signals**

The signals listed in Table 2-1 are present at the Core8051s boundary.

***Table 2-1* • Core8051s Ports**

Signal Name	Type	Polarity/Bus Size	Description
System Signals			
CLK	Input	Rise	Clock input for internal logic. This signal must also be used to clock any APB peripherals, if present.
NSYSRESET	Input	Low	Hardware reset input. A logic zero on this signal for two clock cycles while the oscillator is running resets the device.
PRESETN	Output	Low	Synchronized reset output. This signal should be used to reset any APB peripherals, if present.
WDOGRES	Input	High	Watchdog timeout indication
WDOGRESN	Output	Low	Reset signal for watchdog
MOVX	Output	High	MOVX instruction executing
On-Chip Debug Interface (Optional)			
TCK	Input	Rise	JTAG test clock. If OCI is not used, connect to logic 1.
TMS	Input	High	JTAG test mode select. If OCI is not used, connect to logic 0.
TDI	Input	High	JTAG test data in. If OCI is not used, connect to logic 0.
TDO	Output	High	JTAG test data out
TRSTN	Input	Low	JTAG test reset. If OCI is not used, connect to logic 1.
MEMBANK	Input	4	Optional code memory bank selection. If not used, connect to logic 0.
BREAKIN	Input	High	Break bus input. When sampled high, a breakpoint is generated. If not used, connect to logic 0.
BREAKOUT	Output	High	Break bus output. This is driven high when Core8051s stops emulation. This can be connected to an open-drain break bus that connects to multiple processors, so that when any CPU stops, all others on the bus are stopped within a few clock cycles.
TRIGOUT	Output	High	Trigger output. This signal can be optionally connected to external test equipment to cross-trigger with internal Core8051s activity.
AUXOUT	Output	High	Auxiliary output. This signal is an optional general purpose output that can be controlled via the OCI debugger software.
DBGMEMPSWR	Output	High	Debug program store write.
External Interrupts			
INT0	Input	High	External Interrupt 0 (low priority)
INT1	Input	High	External Interrupt 1 (high priority)
External Memory Bus Interface			
MEMPSACKI	Input	High	Program memory read acknowledge
MEMACKI	Input	High	Data memory acknowledge
MEMDATAI	Input	8	Memory data input
MEMDATAO	Output	8	Memory data output
MEMADDR	Output	16	Memory address

***Table 2-1* • Core8051s Ports (continued)**

Signal Name	Type	Polarity/Bus Size	Description
MEMPSRD	Output	High	Program store read enable
MEMWR	Output	High	Data memory write enable
MEMRD	Output	High	Data memory read enable
APB3 Interface			
PADDR	Output	12	This is the APB address bus.
PSEL	Output	1	This signal indicates that the slave device is selected and a data transfer is required.
PENABLE	Output	High	This strobe signal is used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of an APB transfer. The rising edge of PENABLE occurs in the middle of the APB transfer.
PWRITE	Output	High	When high, this signal indicates an APB write access. When low, it indicates an APB read access.
PRDATA	Input	8, 16, or 32	The read data bus is driven by the selected slave during read cycles (when PWRITE is low). The width of this bus matches the width of the widest peripheral in the system.
PWDATA	Output	8, 16, or 32	The write data bus is driven by the Core8051s during write cycles (when PWRITE is high). The width of this bus matches the width of the widest peripheral in the system.
PREADY	Input	1	This signal is the ready signal for the APB interface. This signal conforms to APB version 3.0. Using this signal, APB slave peripherals can stall reads or writes, if not ready to complete the transaction.
PSLVERR	Input	1	This signal is specified in v3.0 of the APB specification. It is currently unused in Core8051s.

Interface Descriptions

Parameters/Generics

The Verilog parameters or VHDL generics shown in Table 2-2 are present in the Core8051s RTL code. These may be modified by the user to configure Core8051s as required. When working with SmartDesign, these parameters/generics are set to appropriate values using the Core8051s configuration window.

***Table 2-2* • Table x. Core8051s Parameters/Generics**

Parameter/Generic	Default Value	Description
DEBUG	0	0 = On-chip instrumentation (OCI) debug logic not included. 1 = OCI debug logic included; general purpose FPGA I/Os used for debug connection 2 = OCI debug logic included, dedicated JTAG pins of device (along with UJTAG macro) used for debug connection
INCL_TRACE	0	0 = Trace RAM not included 1 = Trace RAM included
TRIG_NUM	0	Number of hardware triggers. Possible settings are 0, 1, 2, or 4.
INCL_DPTR1	0	0 = Second data pointer not included. 1 = Second data pointer included.
INCL_MUL_DIV	1	0 = MUL, DIV, and DA instructions not included. 1 = MUL, DIV, and DA instructions included.
VARIABLE_WAIT	1	0 = Program store memory related acknowledge input (MEMPSACKI) not used for controlling accesses to program memory. A fixed number of wait states (defined by WAIT_VAL parameter) is used for each access to program memory.
WAIT_VAL	0	This setting is only used when VARIABLE_WAIT = 0 and defines the (fixed) number of wait states inserted in each access to program memory. Possible values are 0 to 7.
VARIABLE_STRETCH	1	0 = Data memory related acknowledge input (MEMACKI) not used for controlling accesses to program memory. A fixed number of wait states (defined by WAIT_VAL parameter) is used for each access to program memory. 1 = Data memory related acknowledge input (MEMACKI) is used for controlling accesses to program memory.
STRETCH_VAL	1	This setting is only used when VARIABLE_STRETCH = 0 and defines the (fixed) number of wait states inserted in each access to data memory. Possible values are 0 to 7.
APB_DWIDTH	32	Data width in number of bits for APB bus. Possible settings are 8, 16, or 32.
INTRAM_IMPLEMENTATION	0	This parameter is used to control how the internal (256x8) RAM is implemented. Possible settings are: 0 = Instantiate RAM block 1 = Infer RAM block during synthesis 2 = Infer registers for RAM during synthesis

3 – Tool Flows

SmartDesign

Core8051s is available for download to the SmartDesign IP Catalog via the Libero® Integrated Design Environment (IDE) web repository. For information on using SmartDesign to instantiate, configure, connect, and generate cores, refer to the Libero IDE online help.

The advanced peripheral bus (APB) version 3 interface of Core8051s will typically be connected to the mirrored master interface of CoreAPB3, with various APB or APB3 slaves connected to the slave interfaces of CoreAPB3. The external memory interface (ExternalMemIf) of Core8051s must be connected to program and data memories, which can be implemented either on-chip or off-chip. If debug functionality is enabled, the JTAG signals (TCK, TMS, TDI, TDO, and TRSTN) of the debug interface (DebugIf) must be routed to the top level of your design. Either the dedicated JTAG pins of the device or general purpose I/O pins can be used for the JTAG debug connection. The UJTAG macro is employed when the dedicated JTAG pins are used for the debug connection.

Figure 3-1 shows the Core8051s configuration window, along with cross-references to the corresponding top-level parameters. The parameters/generics of the core are fully described in the "Parameters/Generics" section on page 20.

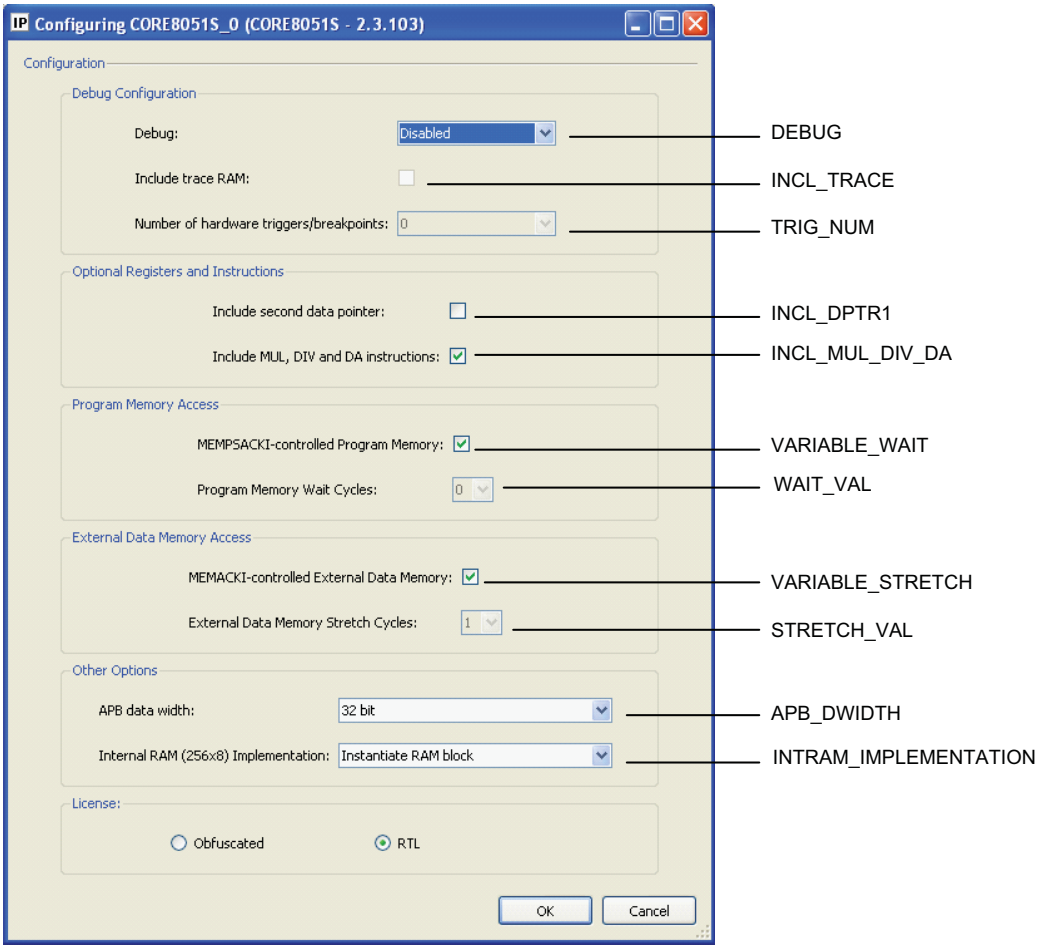

***Figure 3-1* • Core8051s Configuration Window**

The configuration options for Core8051s are described in the following paragraphs. The Core8051s configuration window is used to adjust the values of the underlying parameters/generics in the RTL code for the core. Each configuration option presented in the configuration window corresponds directly to an actual parameter/generic in the RTL code for Core8051s.

Debug Configuration

- There are three debug-related configuration options. Set the Debug option to choose to enable or disable on-chip instrumentation (OCI) debug functionality and to control how any debug connection is implemented. When this functionality is enabled, you can connect a debugger to the processor via a JTAG connection. You can disable the debug functionality if you do not intend to use a debugger and want to minimize the number of tiles consumed by the processor. There are two possibilities for implementing the JTAG connection. From the Debug drop-down menu, choose one of these options:
- **Disabled** to exclude debug functionality
- **Enabled using UJTAG** to include debug functionality and to use the dedicated JTAG pins of the device (via the UJTAG macro) for the debug connection. This setting is mostly used when only one debug connection is required. With this setting you can make use of the FlashPro3 or low-cost programming stick (LCPS) connection for the debug connection.
- **Enabled using I/Os** to include debug functionality and to use general purpose I/O pins for the debug connection. Select this option if the UJTAG macro is either not present on your device or is already in use and not available for the Core8051s debug connection.
- When Debug is set to **Enabled using UJTAG or Enabled using I/Os**, two additional debug options are available for added control over the debug functionality to be included:
- Select **Include trace RAM** to include a 256-byte deep trace RAM within Core8051s. No trace RAM is present if this option is not selected. Including the trace RAM increases the tile count for the processor and consumes RAM blocks on the device.
- Set **Number of hardware triggers/breakpoints** to 0, 1, 2, or 4 to set the maximum number of hardware triggers/breakpoints available when debugging a Core8051s system. Increasing the number of hardware triggers/breakpoints increases the tile count of the processor.

Optional Registers and Instructions

Select **Include second data pointer** to include a second data pointer. When this option is selected, two additional special function registers (SFRs) are included to implement the second (16-bit) data pointer.

Select **Include MUL, DIV, and DA instructions** to include the multiply, divide, and decimal adjust instructions. If the software to be run on the processor does not make use of the MUL, DIV, and DA instructions, this option check box can be cleared to reduce the tile count of the core. The behavior of the processor is undefined when attempting to execute a MUL, DIV or DA instruction while the processor is not configured to include support for these instructions.

Program Memory Access

There are two possible methods for controlling accesses by the processor to program memory:

- Select **MEMPSACKI-controlled Program Memory** when the MEMPSACKI (program store memory acknowledge input) signal is used to control accesses to program memory. When this option is selected, the program memory or memory subsystem must assert MEMPSACKI when a write to program memory has completed and when valid read data is available.
- Clear the check box for **MEMPSACKI-controlled Program Memory** and set a fixed number of wait cycles for each access to program memory by adjusting the Program Memory Wait Cycles option.

Note: Program Memory Wait Cycles is only enabled when **MEMPSACKI-controlled Program Memory** is not selected.

External Data Memory Access

There are two possible methods for controlling accesses by the processor to external data memory.

- Select **MEMACKI-controlled External Data Memory** when the MEMACKI (data memory acknowledge input) signal is used to control accesses to data memory. When this option is selected, the data memory or memory subsystem must assert MEMACKI when a write to data memory has completed and when valid read data is available.
- Clear the check box for **MEMACKI-controlled External Data Memory** and set a fixed number of wait cycles for each access to data memory by adjusting the External Data Memory Stretch Cycles option.

Note: Note that the External Data Memory Stretch Cycles is only enabled when MEMACKI-controlled External Data Memory is not selected.

The external data memory is external to the processor but can be implemented using either on-chip or off-chip memory resources.

Other Options

Set APB data width to **8 bit**, **16 bit**, or **32 bit** to select the appropriate data width for the APB interface of the processor. When the APB data width is 16 bits or 32 bits, extra SFRs are used to store the upper bytes of APB data when the (8-bit) processor core carries out an access to APB space. See the "External Data Memory Space" section on page 30 for more information on the APB interface.

The Internal RAM (256x8) Implementation option is used to control how the internal 256x8 RAM is implemented. Three choices are available:

- **Instantiate RAM block**: A RAM macro block is directly instantiated in the RTL code.
- **Infer RAM block during synthesis**: A synthesis directive (in the form of a structured comment) is used in the RTL code to cause the synthesis tool to infer RAM during synthesis. A RAM macro block will be used in this case, which means that this choice gives a very similar outcome to Instantiate RAM block.
- **Infer registers for RAM during synthesis**: A synthesis directive (in the form of a structured comment) is used in the RTL code to cause the synthesis tool to use registers (FPGA tiles) to implement the 256x8 internal RAM. This considerably increases the tile count for the core but has the benefit of enhancing the fault-tolerant capabilities of Core8051s.

Example System

A typical system that includes Core8051s is shown in Figure 3-2. Connections can be made automatically in SmartDesign using the Auto Connect menu option.

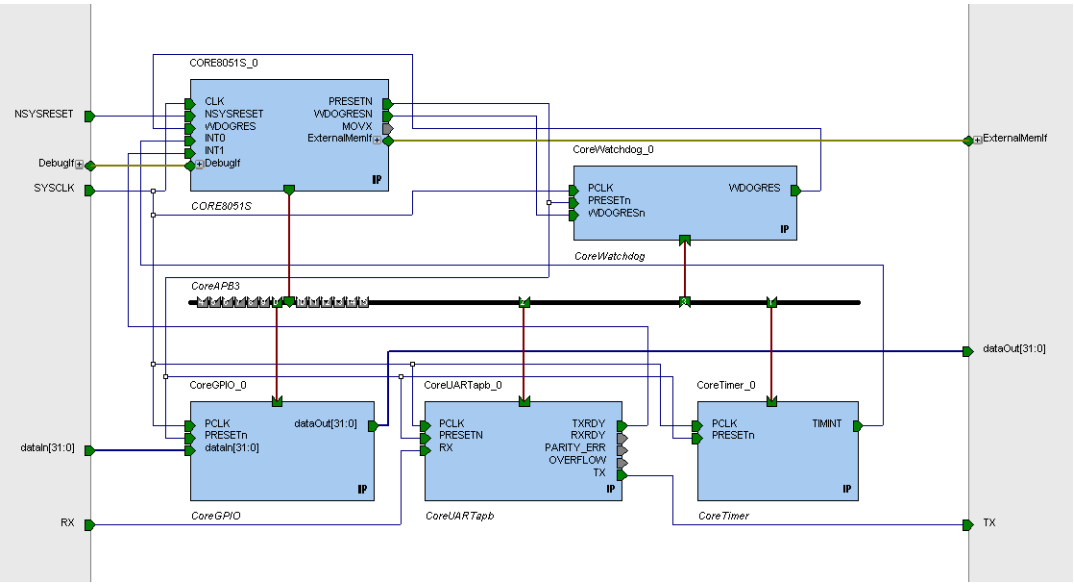

Figure 3-2 • **Example System Including Core8051s**

Simulation

Core8051s comes with a verification testbench and also supports bus functional model (BFM)-based simulation of a system in which it is instantiated. The BFM only simulates transactions on the APB interface of Core8051s and does implement a complete model of the processor. It is not possible to simulate code running on the processor with a
BFM-based simulation.

Core8051s simulation can be invoked from the Libero IDE Project Manager. After the design has been generated, click the **Simulation** button in the Libero IDE to run a simulation.

The Core8051s component must be set as the design root (right-click Core8051s and select **Set As Root**), before running a Core8051s simulation. However, if intending to run a BFM-based simulation, you must first compile the component which instantiates Core8051s. To do this, set the design root one level of hierarchy above the Core8051s component and click the **Simulation** button to invoke ModelSim® and compilation of the relevant components. When the (automatically generated) ModelSim script finishes, exit ModelSim. Now set the design root to the Core8051s component and click the **Simulation** button again. This enables you to run a BFM-based simulation of your Core8051s system. The Core8051s verification testbench can be run directly, without the need to first compile the component that instantiates Core8051s.

The following message will appear in the ModelSim transcript window when running (pre-synthesis) Core8051s simulation:

```
The following (pre-synthesis) simulation options are available for
your Core8051s-based system:
   bfm - APB Bus Functional Model (BFM-driven) simulation of your system
   oci - Run Core8051s On Chip Instrumentation (OCI) tests
   opcode - Run Core8051s opcode test suite, consisting of 256 opcode tests
   <num> - Enter a number in the range 1 to 256 to run a specific opcode test
Enter "bfm", "oci", "opcode" or a number between 1 and 256 and hit return key to select
simulation type
```

Follow the instructions in the ModelSim transcript window to choose the type of simulation to run. BFM-based simulation is not supported after synthesis has been run and *bfm* does not appear as a simulation option in the post-synthesis ModelSim message, which is shown below:

```
The following (post-synthesis) simulation options are available for
your Core8051s-based system:
   oci - Run Core8051s On Chip Instrumentation (OCI) tests
   opcode - Run Core8051s opcode test suite, consisting of 256 opcode tests
   <num> - Enter a number in the range 1 to 256 to run a specific opcode test
(Note: BFM-driven simulation not available post-synthesis)
Enter "oci", "opcode" or a number between 1 and 256 and hit return key to select
simulation type
```

BFM-Based Simulation

When running a BFM-based simulation of a Core8051s system, a BFM command script is used to control the simulation. This command script is dynamically generated by SmartDesign, based on the components connected to the APB interface of Core8051s. The command script file is named subsystem.bfm and is located in the simulation folder. You can modify the command script, refer to "BFM-Script Language" for details on the syntax used in the file.

During simulation, the BFM generates a series of transfers on the APB bus. These write to and read from registers within peripherals attached to the APB bus, of which Core8051s is master. This verifies that the APB interface is fully operational. The BFM tests do not perform any verification on the Core8051s itself. The advantage of BFM-driven simulation is that you can exercise the system using a simple scripting language, before writing any C code or 8051 assembler code.

BFM-Script Language

The following script commands are defined for use by the BFM:

memmap

This command is used to associate a label, representing a system resource, with a memory map location. The other BFM script commands may perform accesses to locations within this resource by referencing this label and a register offset relative to this base address.

Syntax

memmap resource_name base_address;

- resource_name: This is a string containing the user-friendly instance name of the resource being accessed. For BFM scripts generated automatically by SmartDesign, this name corresponds to the instance name of the associated core in the generated subsystem Verilog or VHDL.
- base_address: This is the base address of the resource, in hexadecimal format.

write

This command causes the BFM to perform a write to a specified offset, within the memory map range of a specified system resource.

Syntax

write width resource_name byte_offset data;

- width: This takes on the enumerated values of W, H, or B, for word, halfword, or byte.
- resource_name: This is a string containing the user-friendly instance name of the resource being accessed.
- byte_offset: This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.
- data: This is the data to be written. It is specified as a hexadecimal value.

Example

write W videoCodec 20 11223344;

read

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource.

Syntax

read width resource_name byte_offset;

- width: This takes on the enumerated values of W, H, or B, for word, halfword, or byte.
- resource_name: This is a string containing the user-friendly instance name of the resource being accessed.
- byte_offset: This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.

Example

read W videoCodec 20;

readcheck

This command causes the BFM to perform a read of a specified offset, within the memory map range of a specified system resource, and to compare the read value with the expected value provided.

Syntax

readcheck width resource_name byte_offset data;

- width: This takes on the enumerated values of W, H, or B, for word, halfword, or byte.
- resource_name: This is a string containing the user-friendly instance name of the resource being accessed.
- byte_offset: This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.
- data: This is the expected read data. It is specified as a hexadecimal value.

Example

readcheck W videoCodec 20 11223344;

poll

This command continuously reads a specified location until a requested value is obtained. This command allows one or more bits of the read data to be masked out. This allows, for example, poll waiting for a ready bit to be set, while ignoring the values of the other bits in the location being read.

Syntax

poll width resource_name byte_offset data bitmask;

- width: This takes on the enumerated values of W, H, or B, for word, halfword, or byte.
- resource_name: This is a string containing the user-friendly instance name of the resource being accessed.
- byte_offset: This is the offset from the base of the resource, in bytes. It is specified as a hexadecimal value.
- bitmask: The bitmask is ANDed with the read data and the result is then compared to the bitmask itself. If equal, then all the bits of interest are at their required value and the poll command is complete. If not equal, then the polling continues.

wait

This command causes the BFM script to stall for a specified number of clock periods.

Syntax

wait num_clock_ticks;

- num_clock_ticks: This is the number of clock periods during which the BFM stalls (does not initiate any bus transactions).

waitint0

This command causes the BFM to wait until an interrupt event (Low to High transition) is seen on the INT0 pin before proceeding with the execution of the remainder of the script.

Syntax

waitint0;

waitint1

This command causes the BFM to wait until an interrupt event (Low to High transition) is seen on the INT1 pin before proceeding with the execution of the remainder of the script.

Syntax

waitint1;

Synthesis in Libero IDE

To run synthesis on the core with the parameter settings selected in SmartDesign, set the design root appropriately, and click the **Synthesis** button in the Project Manager. The Synthesis window appears, displaying the Synplicity® project. To perform synthesis, click the **Run** button.

Place-and-Route in Libero IDE

After setting the design root appropriately and running synthesis, click the **Layout** button in the Project Manager to invoke Designer. Core8051s requires no special place-and-route settings.

4 – Core8051s Features

Software Memory Map

The Core8051s microcontroller utilizes the Harvard architecture, with separate code and data spaces. Memory organization in Core8051s is similar to that of the industry standard 8051. There are three memory areas, as shown in Figure 4-1:

- Program memory (internal RAM, external RAM, or external ROM)
- External data memory (external RAM)
- Internal data memory (internal RAM)

The software memory map for the Core8051s is shown in Figure 4-1.

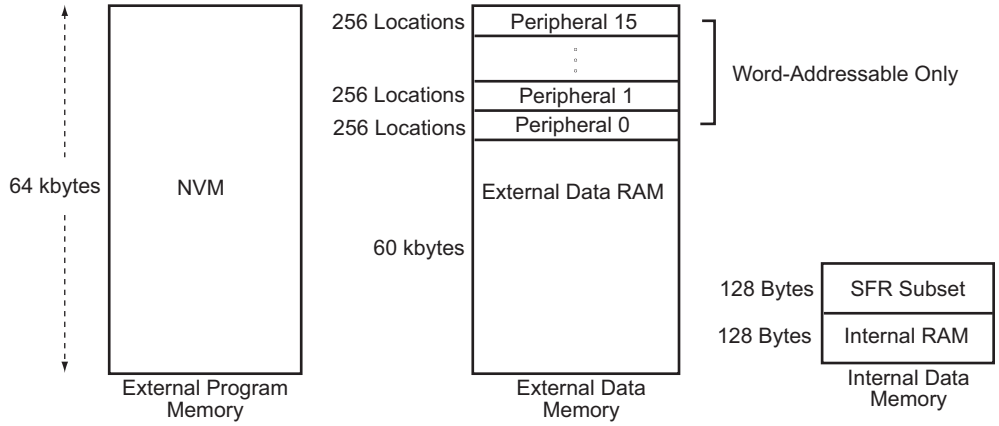

***Figure 4-1* • Core8051s Software Memory Map**

As far as the software programmer is concerned, there are three distinct memory spaces available, as shown in Figure 4-1.

Program Memory

Core8051s can address up to 64 kbytes of program memory space, from 0000H to FFFFH. The external memory bus interface (Table 4-1 on page 31) services program memory when the MEMPSRD signal is active. Program memory is read when the CPU performs fetching instructions or MOVC. After reset, the CPU starts program execution from location 0000H. The lower part of the program memory includes interrupt and reset vectors. The interrupt vectors are spaced at eight-byte intervals, starting from 0003H. Program memory can be implemented as internal RAM, external RAM, external ROM, or a combination of all three. Writing to external program memory is only supported in debug mode, using the OCI logic block and external debugger hardware and software.

The program memory can use variable length accesses (MEMPSACKI-controlled), or a fixed number of wait cycles may be inserted on each read. Refer to "Program Memory Access" on page 22 for more information about configuring access to program memory.

External Data Memory Space

Core8051s can address up to 64 kbytes of external data memory space, from 0000H to FFFFH. This memory is external to the core, not necessarily to the FPGA. In the Core8051s, the upper 4 kbytes (F000H to FFFFH) of external data memory space is mapped to an APB bus. The lower 60 kbytes is mapped to the external memory bus interface.

External Data Interface

The external memory bus interface (Table 2-1 on page 18) services data memory when the MEMRD signal is active. Core8051s writes into external data memory when the CPU executes MOVX @Ri,A or MOVX @DPTR,A instructions. The external data memory is read when the CPU executes MOVX A,@Ri or MOVX A,@DPTR instructions. There is improved variable length of the MOVX instructions to access fast or slow external RAM and external peripherals. The external data memory can use variable length accesses (MEMACKI-controlled), or a fixed number of stretch cycles may be inserted on each read or write. Refer to "External Data Memory Access" on page 23 for more information about configuring access to external data memory.

APB Interface

Core8051s based systems use an APB bus for connecting peripherals, where the Core8051s acts as the bus master. The width of the APB bus on Core8051s can be selected to match the width of the widest APB peripheral in the system (8, 16, or 32 bits). As the Core8051s is an 8-bit processor and it is not possible to indicate transaction size on the APB, reads and writes from or to the APB bus in 16-bit or 32-bit mode are accomplished by means of newly defined SFRs, hereafter referred to as X registers. For example, to perform a write to a 32-bit APB peripheral, the program running on the Core8051s must first perform three individual 8-bit writes to X registers (XWB1, XWB2, and XWB3). These registers hold the value to be written out on PWDATA [31:8]. When the program subsequently does a write to the APB address in question, the 8 bits of the write data associated with that write cycle are put out on the PWDATA [7:0] and the three write “X registers” are put onto the APB bus as PWDATA [31:8].

16-bit and 32-bit reads from the APB are handled in a similar manner. To perform a 32-bit read from an APB location, the program must perform a read of the APB location, from which it immediately obtains bits [7:0] of the 16 or 32 bits on PRDATA[7:0]. Subsequently, the program must read the three read X registers (XRB1, XRB2, and XRB3) to get bits [31:8], which were read from the APB peripheral and latched in these SFRs at the time of the APB transaction.

For the 4 kbytes of memory space allocated to the APB interface, only word access is possible, where word refers to an 8-bit, 16-bit, or 32-bit entity, for their respective APB bus implementations.

The APB interface of Core8051s will typically be connected to CoreAPB3, which can in turn connect to up to 16 peripherals such as CoreTimer and CoreGPIO. Often the programmer accessible registers in these peripherals will be located at 32-bit word boundaries in the address map. This means that consecutive registers are located at address offsets 0x00, 0x04, 0x08, 0x0C, and so on. Core8051s must take account of this when accessing such peripherals. For example, to access successive register locations in a peripheral attached to slave slot 0 on CoreAPB3, Core8051s would issue addresses 0xF000, 0xF004, 0xF008, 0xF00C, and so on.

The net effect is that only every fourth location in the APB space is usable if the peripherals are designed such that their registers are located at 32-bit word boundaries in the memory map. If all of the 4 kbytes of APB space connects to peripherals of this type, then there are only 1,024 separately addressable locations, which equates to 64 locations per peripheral, assuming CoreAPB3 is used.

Note that the APB data width is independent of the addressing scheme. Each location can hold a value which is 8, 16, or 32 bits wide. The APB data width configurable option of Core8051s should be set to match the largest data width to be accessed on the APB interface.

Internal Data Memory Space

Internal RAM

The internal data memory space services 256 bytes of data RAM and 128 bytes of SFRs. The internal data memory address is always one byte wide. The memory space is 256 bytes large (00H to FFH). Direct or indirect addressing accesses the lower 128 bytes of internal RAM. Indirect addressing accesses the upper 128 bytes of internal RAM.

The lower 128 bytes contain work registers and bit-addressable memory. The lower 32 bytes form four banks of eight registers (R0–R7). Two bits on the program memory status word (PSW) select which bank is in use. The next 16 bytes form a block of bit-addressable memory space at bit addressees 00H–7FH.

SFR Registers

The SFRs occupy the upper 128 bytes of internal data memory space. This SFR area is available only by direct addressing.

Table 4-1 lists the SFR registers present in Core8051s.

***Table 4-1* • Core8051s SFR Registers**

Register	Location	Description
SP	0x81	Stack pointer
DPL	0x82	Data pointer 0 Low
DPH	0x83	Data pointer 0 High
DPL1	0x84	Data pointer 1 Low (optional)
DPH1	0x85	Data pointer 1 High (optional)
ICON	0x88	Interrupt control register
DPS	0x92	Data pointer select (optional)
XWB1	0x9A	External write buffer 1 (optional)
XWB2	0x9B	External write buffer 2 (optional)
XWB3	0x9C	External write buffer 3 (optional)
XRB1	0x9D	External read buffer 1 (optional)
XRB2	0x9E	External read buffer 2 (optional)
XRB3	0x9F	External read buffer 3 (optional)
IE	0xA8	Interrupt enable register
PSW	0xD0	Program status word (bit-addressable)
ACC	0xE0	Accumulator (bit-addressable)
B	0xF0	B register (bit-addressable)

The above table contains the minimal subset of SFR registers (SP, DPL, DPH, PSW, ACC, and B) that are required to support existing C compilers. There is an optional second data pointer (not available by default). There are also six non-standard SFR registers shown, referred to hereafter as X registers. The XWB1 and XRB1 registers are present only if APB_DWIDTH is 16 or greater. XWB2, XWB3, XRB2, and XRB3 are present only if APB_DWIDTH is 32. They are used to provide write data and latch read data for the upper 3 bytes of the APB bus, if present, during a MOVX instruction to APB memory space (within external data memory space). The six X registers are not bit-addressable. Note also that the X registers are read/write. This is necessary to handle the situation where an ISR needs to access the APB bus, but has interrupted between the user setting up the X registers and performing the MOVX (on an APB write), or between the MOVX and reading of the X registers (on an APB read). The recommended behavior for an ISR is to read the X registers on entry into the ISR and to restore them to their original values on exiting the ISR.

Accumulator (acc)

The acc register is the accumulator. Most instructions use the accumulator to hold the operand. The mnemonics for accumulator-specific instructions refer to the accumulator as A, not ACC.

B Register (b)

The b register is used during multiply and divide instructions. It can also be used as a scratch-pad register to hold temporary data.

Program Status Word (psw)

The psw register flags and bit functions are listed in Table 4-2 and Table 4-3.

***Table 4-2* • psw Register Flags**

cy	ac	f0	rs1	rs	ov	–	p

***Table 4-3* • psw Bit Functions**

Bit	Symbol	Function
7	cy	Carry flag
6	ac	Auxiliary carry flag for BCD operations
5	f0	General purpose flag 0 available for user
4	rs1	Register bank select control bit 1, used to select working register bank
3	rs0	Register bank select control bit 0, used to select working register bank
2	ov	Overflow flag
1	–	User defined flag
0	p	Parity flag, affected by hardware to indicate odd / even number of "one" bits in the accumulator, i.e. even parity

The state of bits rs1 and rs0 from the psw register select the working registers bank as listed in Table 4-4.

***Table 4-4* • rs1/rs0 Bit Selections**

rs1/rs0	Bank selected	Location
00	Bank 0	(00H – 07H)
01	Bank 1	(08H – 0FH)
10	Bank 2	(10H – 17H)
11	Bank 3	(18H – 1FH)

Stack Pointer (sp)

The stack pointer is a one-byte register initialized to 07H after reset. This register is incremented before PUSH and CALL instructions, causing the stack to begin at location 08H.

Data Pointer (dptr)

The data pointer (dptr) is two bytes wide. The lower part is DPL, and the highest is DPH. It can be loaded as a two- byte register (MOV DPTR,#data16) or as two registers (e.g. MOV DPL,#data8). It is generally used to access external code or data space (e.g. MOVC A,@A+DPTR or MOV A,@DPTR respectively).

Program Counter (pc)

The program counter is two bytes wide, and is initialized to 0000H after reset. This register is incremented during fetching operation code or operation data from program memory.

Interrupt Enable Register (ie)

The interrupt enable register is a one-byte register initialized to 00H after reset. The IE bit functions are listed in Table 4-5. Note that the EAL and EX0 bits must both be set to 1 to enable the INT0 interrupt. Similarly, EAL and EX1 must both be set to 1 to enable the INT1 interrupt.

***Table 4-5* • Bit Functions**

Bit	Symbol	Default Value	Function
7	EAL	0	0 = Disable all interrupts
6	–	0	Unused
5	–	0	Unused
4	–	0	Unused
3	–	0	Unused
2	EX1	0	0 = Disable external interrupt 1 (INT1)
1	–	0	Unused
0	EX0	0	0 = Disable external interrupt 0 (INT0)

Interrupt Control Register (icon)

The interrupt control register is a one-byte register initialized to 00H after reset. The ICON bit functions are listed in Table 4-6. The ICON register implements a subset of the Timer Control (TCON) register, which is commonly present in implementations of the 8051 processor.

***Table 4-6* • Bit Functions**

Bit	Symbol	Default Value	Function
7	–	0	Unused
6	–	0	Unused
5	–	0	Unused
4	–	0	Unused
3	IE1	0	Interrupt 1 event flag. When IT1 = 0, this flag follows the level on the INT1 input. When IT1 = 1, this flag is set when a rising edge is observed on interrupt input INT1, and is cleared when the interrupt is processed.
2	IT1	0	Interrupt 1 type control bit. This bit selects whether a rising edge or a high level on input pin INT1 causes an interrupt. 0 = High level causes interrupt. 1 = Rising edge causes interrupt.
1	IE0	0	Interrupt 0 event flag. When IT0 = 0, this flag follows the level on the INT0 input. When IT0 = 1, this flag is set when a rising edge is observed on interrupt input INT0, and is cleared when the interrupt is processed.
0	IT0	0	Interrupt 0 type control bit. This bit selects whether a rising edge or a high level on input pin INT0 causes an interrupt. 0 = High level causes interrupt. 1 = Rising edge causes interrupt

Interrupts

Core8051s has two interrupt inputs, INT0 and INT1. INT0 is low priority (priority level 0), with a vector address of 03H. INT1 is high priority (priority level 1), with a vector address of 13H.

Note: If using the Keil C51 C compiler, an interrupt function attribute of 0 must be used for INT0 and an attribute of 2 for INT1.

The interrupt enable (IE) and interrupt control (ICON) special function registers are used to determine interrupt behavior.

Interrupts can be individually or collectively enabled or disabled using the interrupt enable register.

The interrupt control register contains an event flag and a type control bit for the INT0 and INT1 interrupts. Each type control bit is used to control whether the corresponding interrupt is rising edge or level High sensitive, with the default being level High sensitive. Each event flag is set to 1 when a rising edge or High level is detected on the corresponding interrupt input.

When rising edge sensitive operation is selected (by setting the type control bit to 1), the relevant event flag will be automatically cleared when the interrupt is serviced. This automatic clearing of the event flags is made possible by logic in the processor that detects vectoring to address 03H (in the case of an INT0 interrupt) or 13H (in the case of an INT1 interrupt).

When level sensitive interrupt operation is selected, the relevant event flag is not cleared when the interrupt is serviced and remains asserted until the source of the interrupt is cleared. The event flag effectively follows the (INT0 or INT1) interrupt input when level sensitive operation is selected. The interrupt service routine must clear the source of the interrupt when level sensitive interrupts are used.

OCI Block

The on-chip instrumentation (OCI) block communicates with external debugger hardware and software as a debugging aid to the user. The OCI debug block can be optionally included, refer to "Debug Configuration" on page 22 for more information on debug related configuration options. The following debug features are present in Core8051s:

- Run/stop control
- Single-step mode
- Software breakpoint
- Execution of a debugger program
- Hardware breakpoint
- Program trace
- Access to ACC (accumulator) register

5 – Instruction Set

The Core8051s instructions are binary code compatible and perform the same functions as the industry-standard 8051. This is the ASM51 instruction set. Some of these instructions, however, are not enabled by default and so must be explicitly enabled if required.

Table 5-1 and Table 5-2 contain notes for mnemonics used in the various instruction set tables. In Table 5-3 on page 36 through Table 5-7 on page 40, the instructions are ordered in functional groups. In Table 5-8 on page 41, the instructions are ordered in the hexadecimal order of the operation code. For more detailed information about the Core8051s instruction set, refer to the *Core8051 Instruction Set Details User's Guide*.

***Table 5-1* • Notes on Data Addressing Modes**

Rn	Working register, R0–R7
direct	128 internal RAM locations, any I/O port, control or status register
@Ri	Indirect internal or external RAM location addressed by register, R0 or R1
#data	8-bit constant included in instruction
#data 16	16-bit constant included as bytes 2 and 3 of instruction
bit	128 software flags, any bit-addressable I/O pin, control or status bit
A	Accumulator

***Table 5-2* • Notes on Programming Addressing Modes**

addr16	Destination address for LCALL and LJMP may be anywhere within the 64 kbytes program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2 kbytes page of program memory as the first byte of the following instruction.
Rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is from plus 127 to minus 128 bytes, relative to the first byte of the following instruction.

Functional Ordered Instructions

Table 5-3 through Table 5-7 on page 40 list the functional ordered instructions.

***Table 5-3* • Arithmetic Instructions**

Mnemonic	Description	Byte	Cycle
ADD A,Rn	Adds the register to the accumulator.	1	1
ADD A,direct	Adds the direct byte to the accumulator.	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator.	1	2
ADD A,#data	Adds the immediate data to the accumulator.	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag.	1	1
ADDC A,direct	Adds the direct byte to A with a carry flag.	2	2
ADDC A,@Ri	Adds the indirect RAM to A with a carry flag.	1	2
ADDC A,#data	Adds the immediate data to A with carry a flag.	2	2
SUBB A,Rn	Subtracts the register from A with a borrow.	1	1
SUBB A,direct	Subtracts the direct byte from A with a borrow.	2	2
SUBB A,@Ri	Subtracts the indirect RAM from A with a borrow.	1	2
SUBB A,#data	Subtracts the immediate data from A with a borrow.	2	2
INC A	Increments the accumulator.	1	1
INC Rn	Increments the register.	1	2
INC direct	Increments the direct byte.	2	3
INC @Ri	Increments the indirect RAM.	1	3
DEC A	Decrements the accumulator.	1	1
DEC Rn	Decrements the register.	1	1
DEC direct	Decrements the direct byte.	1	2
DEC @Ri	Decrements the indirect RAM.	2	3
INC DPTR	Increments the data pointer.	1	3
MUL A,B	Multiplies A and B.	1	5
DIV A,B	Divides A by B.	1	5
DA A	Decimal adjust accumulator	1	1

***Table 5-4* • Logic Operations**

Mnemonic	Description	Byte	Cycle
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2
ANL A,#data	AND immediate data to accumulator	2	2
ANL direct,A	AND accumulator to direct byte	2	3
ANL direct,#data	AND immediate data to direct byte	3	4
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL A,#data	OR immediate data to accumulator	2	2
ORL direct,A	OR accumulator to direct byte	2	3
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	2	3
XRL direct,#data	Exclusive OR immediate data to direct bytre	3	4
CLR A	Clears the accumulator.	1	1
CPL A	Complements the accumulator.	1	1
RL A	Rotates the accumulator left.	1	1
RLC A	Rotates the accumulator left through carry.	1	1
RR A	Rotates the accumulator right.	1	1
RRC A	Rotates the accumulator right through carry.	1	1
SWAP A	Swaps nibbles within the accumulator.	1	1

***Table 5-5* • Data Transfer Operations**

Mnemonic	Description	Byte	Cycle
MOV A,Rn	Moves the register to the accumulator.	1	1
MOV A,direct	Moves the direct byte to the accumulator.	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator.	1	2
MOV A,#data	Moves the immediate data to the accumulator.	2	2
MOV Rn,A	Moves the accumulator to the register.	1	2
MOV Rn,direct	Moves the direct byte to the register.	2	4
MOV Rn,#data	Moves the immediate data to the register.	2	2
MOV direct,A	Moves the accumulator to the direct byte.	2	3
MOV direct,Rn	Moves the register to the direct byte.	2	3
MOV direct,direct	Moves the direct byte to the direct byte.	3	4
MOV direct,@Ri	Moves the indirect RAM to the direct byte.	2	4
MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM.	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM.	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM.	2	3
MOV DPTR,#data16	Loads the data pointer with a 16-bit constant.	3	3
MOVC A,@A + DPTR	Moves the code byte relative to the DPTR to the accumulator.	1	3
MOVC A,@A + PC	Moves the code byte relative to the PC to the accumulator.	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to A.	1	3–10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to A.	1	3–10
MOVX @Ri,A	Moves A to the external RAM (8-bit address).	1	4–11
MOVX @DPTR,A	Moves A to the external RAM (16-bit address).	1	4–11
PUSH direct	Pushes the direct byte onto the stack.	2	4
POP direct	Pops the direct byte from the stack.	2	3
XCH A,Rn	Exchanges the register with the accumulator.	1	2
XCH A,direct	Exchanges the direct byte with the accumulator.	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator.	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with A.	1	3

***Table 5-6* • Boolean Manipulation Operations**

Mnemonic	Description	Byte	Cycle
CLR C	Clears the carry flag.	1	1
CLR bit	Clears the direct bit.	2	3
SETB C	Sets the carry flag.	1	1
SETB bit	Sets the direct bit.	2	3
CPL C	Complements the carry flag.	1	1
CPL bit	Complements the direct bit.	2	3
ANL C,bit	AND direct bit to the carry flag.	2	2
ANL C,bit	AND complements of direct bit to the carry.	2	2
ORL C,bit	OR direct bit to the carry flag.	2	2
ORL C,bit	OR complements of direct bit to the carry.	2	2
MOV C,bit	Moves the direct bit to the carry flag.	2	2
MOV bit, C	Moves the carry flag to the direct bit.	2	3

***Table 5-7* • Program Branch Operations**

Mnemonic	Description	Byte	Cycle
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET Return	Return from subroutine	1	4
RETI Return	Return from interrupt	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (relative address)	2	3
JMP @A + DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if accumulator is zero	2	3
JNZ rel	Jump if accumulator is not zero	2	3
JC rel	Jump if carry flag is set	2	3
JNC rel	Jump if carry flag is not set	2	3
JB bit,rel	Jump if direct bit is set	3	4
JNB bit,rel	Jump if direct bit is not set	3	4
JBC bit,rel	Jump if direct bit is set and clears bit	3	4
CJNE A,direct,rel	Compares direct byte to A and jumps if not equal.	3	4
CJNE A,#data,rel	Compares immediate to A and jumps if not equal.	3	4
CJNE Rn,#data rel	Compares immediate to the register and jumps if not equal.	3	4
CJNE @Ri,#data,rel	Compares immediate to indirect and jumps if not equal.	3	4
DJNZ Rn,rel	Decrements register and jumps if not zero.	2	3
DJNZ direct,rel	Decrements direct byte and jumps if not zero.	3	4
NOP	No operation	1	1

Hexadecimal Ordered Instructions

The Core8051s instructions are listed in Table 5-8 in order of hexadecimal opcode (operation code).

***Table 5-8* • Core8051s Instruction Set in Hexadecimal Order**

Opcode	Mnemonic	Opcode	Mnemonic
00H	NOP	10H	JBC bit,rel
01H	AJMP addr11	11H	ACALL addr11
02H	LJMP addr16	12H	LCALL addr16
03H	RR A	13H	RRC A
04H	INC A	14H	DEC A
05H	INC direct	15H	DEC direct
06H	INC @R0	16H	DEC @R0
07H	INC @R1	17H	DEC @R1
08H	INC R0	18H	DEC R0
09H	INC R1	19H	DEC R1
0AH	INC R2	1AH	DEC R2
0BH	INC R3	1BH	DEC R3
0CH	INC R4	1CH	DEC R4
0DH	INC R5	1DH	DEC R5
0EH	INC R6	1EH	DEC R6
0FH	INC R7	1FH	DEC R7
20H	JB bit,rel	30H	JNB bit,rel
21H	AJMP addr11	31H	ACALL addr11
22H	RET	32H	RETI
23H	RL A	33H	RLC A
24H	ADD A,#data	34H	ADDC A,#data
25H	ADD A,direct	35H	ADDC A,direct
26H	ADD A,@R0	36H	ADDC A,@R0
27H	ADD A,@R1	37H	ADDC A,@R1
28H	ADD A,R0	38H	ADDC A,R0
29H	ADD A,R1	39H	ADDC A,R1
2AH	ADD A,R2	3AH	ADDC A,R2
2BH	ADD A,R3	3BH	ADDC A,R3
2CH	ADD A,R4	3CH	ADDC A,R4
2DH	ADD A,R5	3DH	ADDC A,R5
2EH	ADD A,R6	3EH	ADDC A,R6
2FH	ADD A,R7	3FH	ADDC A,R7

*Note: *The A5H opcode is used as a trap instruction for the implementation of software breakpoints.*

***Table 5-8* • Core8051s Instruction Set in Hexadecimal Order (continued)**

Opcode	Mnemonic	Opcode	Mnemonic
40H	JC rel	50H	JNC rel
41H	AJMP addr11	51H	ACALL addr11
42H	ORL direct,A	52H	ANL direct,A
43H	ORL direct,#data	53H	ANL direct,#data
44H	ORL A,#data	54H	ANL A,#data
45H	ORL A,direct	55H	ANL A,direct
46H	ORL A,@R0	56H	ANL A,@R0
47H	ORL A,@R1	57H	ANL A,@R1
48H	ORL A,R0	58H	ANL A,R0
49H	ORL A,R1	59H	ANL A,R1
4AH	ORL A,R2	5AH	ANL A,R2
4BH	ORL A,R3	5BH	ANL A,R3
4CH	ORL A,R4	5CH	ANL A,R4
4DH	ORL A,R5	5DH	ANL A,R5
4EH	ORL A,R6	5EH	ANL A,R6
4FH	ORL A,R7	5FH	ANL A,R7
60H	JZ rel	70H	JNZ rel
61H	AJMP addr11	71H	ACALL addr11
62H	XRL direct,A	72H	ORL C,bit
63H	XRL direct,#data	73H	JMP @A+ DPTR
64H	XRL A,#data	74H	MOV A,#data
65H	XRL A,direct	75H	MOV direct,#data
66H	XRL A,@R0	76H	MOV @R0,#data
67H	XRL A,@R1	77H	MOV @R1
68H	XRL A,R0	78H	MOV R0,#data
69H	XRL A,R1	79H	MOV R1,#data
6AH	XRL A,R2	7AH	MOV R2,#data
6BH	XRL A,R3	7BH	MOV R3,#data
6CH	XRL A,R4	7CH	MOV R4,#data
6DH	XRL A,R5	7DH	MOV R5,#data
6EH	XRL A,R6	7EH	MOV R6,#data
6FH	XRL A,R7	7FH	MOV R7,#data

*Note: *The A5H opcode is used as a trap instruction for the implementation of software breakpoints.*

***Table 5-8* • Core8051s Instruction Set in Hexadecimal Order (continued)**

Opcode	Mnemonic	Opcode	Mnemonic
80H	SJMP rel	90H	MOV DPTR,#data16
81H	AJMP addr11	91H	ACALL addr11
82H	ANL C,bit	92H	MOV bit,C
83H	MOVC A,@A+ PC	93H	MOVC A,@A+ DPTR
84H	DIV AB	94H	SUBB A,#data
85H	MOV direct,direct	95H	SUBB A,direct
86H	MOV direct,@R0	96H	SUBB A,@R0
87H	MOV direct,@R1	97H	SUBB A,@R1
88H	MOV direct,R0	98H	SUBB A,R0
89H	MOV direct,R1	99H	SUBB A,R1
8AH	MOV DIRECT,R2	9AH	SUBB A,R2
8BH	MOV DIRECT,R3	9BH	SUBB A,R3
8CH	MOV DIRECT,R4	9CH	SUBB A,R4
8DH	MOV DIRECT,R5	9DH	SUBB A,R5
8EH	MOV DIRECT,R6	9EH	SUBB A,R6
8FH	MOV DIRECT,R7	9FH	SUBB A,R7
A0H	ORL C,~bit	B0H	ANL C,~bit
A1H	AJMP addr11	B1H	ACALL addr11
A2H	MOV C,bit	B2H	CPL bit
A3H	INC DPTR	B3H	CPL C
A4H	MUL AB	B4H	CJNE A,#data,rel
A5H*	–	B5H	CJNE A,direct,rel
A6H	MOV @R0,direct	B6H	CJNE @R0,#data,rel
A7H	MOV @R1,direct	B7H	CJNE @R1,#data,rel
A8H	MOV R0,direct	B8H	CJNE R0,#data,rel
A9H	MOV R1,direct	B9H	CJNE R1,#data,rel
AAH	MOV R2,direct	BAH	CJNE R2,#data,rel
ABH	MOV R3,direct	BBH	CJNE R3,#data,rel
ACH	MOV R4,direct	BCH	CJNE R4,#data,rel
ADH	MOV R5,direct	BDH	CJNE R5,#data,rel
AEH	MOV R6,direct	BEH	CJNE R6,#data,rel
AFH	MOV R7,direct	BFH	CJNE R7,#data,rel

*Note: *The A5H opcode is used as a trap instruction for the implementation of software breakpoints.*

***Table 5-8* • Core8051s Instruction Set in Hexadecimal Order (continued)**

Opcode	Mnemonic	Opcode	Mnemonic
C0H	PUSH direct	D0H	POP direct
C1H	AJMP addr11	D1H	ACALL addr11
C2H	CLR bit	D2H	SETB bit
C3H	CLR C	D3H	SETB C
C4H	SWAP A	D4H	DA A
C5H	XCH A,direct	D5H	DJNX direct,rel
C6H	XCH A,@R0	D6H	XCHD A,@R0
C7H	XCH A,@R1	D7H	XCHD A,@R1
C8H	XCH A,R0	D8H	DJNZ R0,rel
C9H	XCH A,R1	D9H	DJNZ R1,rel
CAH	XCH A,R2	DAH	DJNZ R2,rel
CBH	XCH A,R3	DBH	DJNZ R3,rel
CCH	XCH A,R4	DCH	DJNZ R4,rel
CDH	XCH A,R5	DDH	DJNZ R5,rel
CEH	XCH A,R6	DEH	DJNZ R6,rel
CFH	XCH A,R7	DFH	DJNZ R7,rel
E0H	MOVX A,@DPTR	F0H	MOVX@DPTR,A
E1H	AJMP addr11	F1H	ACALL addr11
E2H	MOVX A,@R0	F2H	MOVX@R0,A
E3H	MOVX A,@R1	F3H	MOVX@R1,A
E4H	CLR A	F4H	CPL A
E5H	MOV A,direct	F5H	MOV direct,a
E6H	MOV A,@R0	F6H	MOV@R0,A
E7H	MOV A,@R1	F7H	MOV@R1,A
E8H	MOV A,R0	F8H	MOV R0,A
E9H	MOV A,R1	F9H	MOV R1,A
EAH	MOV A,R2	FAH	MOV R2,A
EBH	MOV A,R3	FBH	MOV R3,A
ECH	MOV A,R4	FCH	MOV R4,A
EDH	MOV A,R5	FDH	MOV R5,A
EEH	MOV A,R6	FEH	MOV R6,A
EFH	MOV A,R7	FFH	MOV R7,A

*Note: *The A5H opcode is used as a trap instruction for the implementation of software breakpoints.*

Instruction Definitions

All Core8051s core instructions can be condensed to 53 basic operations, alphabetically ordered according to the operation mnemonic section, as shown in Table 5-9.

***Table 5-9* • PSW Flag Modification (CY, OV, AC)**

Instruction	Flag			Instruction	Flag		
	CY	OV	AC		CY	OV	AC
ADD	X	X	X	**SETB C**	1	–	–
ADDC	X	X	X	**CLR C**	0	–	–
SUBB	X	X	X	**CPL C**	X	–	–
MUL	0	X	–	**ANL C,bit**	X	–	–
DIV	0	X	–	**ANL C,~bit**	X	–	–
DA	X	–	–	**ORL C,bit**	X	–	–
RRC	X	–	–	**ORL C,~bit**	X	–	–
RLC	X	–	–	**MOV C,bit**	X	–	–
CJNE	X	–	–				

Note: In this table, 'X' denotes that the indicated flag is affected by the instruction and can be a logic 1 or logic 0, depending upon specific calculations. If a particular box is blank, that flag is unaffected by the listed instruction.

C Compiler Support

Because the Core8051s is 100% compatible with the ASM51 instruction set and supports the three traditional 8051 microcontroller memory spaces, it may be targeted by existing 8051 C compilers.

The following section describes in more detail the considerations involved in writing C code for the 8051, when using the Keil Cx51 C compiler. Note that the considerations are similar to those required for other 8051 C compilers, such as the Small Device C Compiler (SDCC), which is bundled with Actel's SoftConsole software development environment.

ANSI C Compliance

It is theoretically possible to write fully compliant ANSI C code and target it to the Core8051s. However, there are a number of issues to be aware of, as listed below.

- Some of the types for the arguments of functions in the Keil C runtime library are modified from those defined in the standard ANSI C. This is to use smaller sizes, where possible.
- Some of the functions in the Keil C runtime library use proprietary extensions to C (as described in "Allocation of Variables in C"), such as bit and xdata types.
- Some of the functions defined by ANSI C are not present in the Keil C runtime library.
- The Keil C runtime library contains some extra functions not defined in ANSI C.

Therefore, pure standard ANSI C code is guaranteed to run only if it does not use any of the above functions when using the Keil C runtime library. Alternatively, the user may provide a runtime library other than the Keil C runtime library.

To get optimal usage of the 8051 architecture, however, many users would just modify their ANSI C application, if necessary, to make optimal use of the 8051 architecture.

Allocation of Variables in C

One of the considerations in writing C software for an 8051-based system is allocation of variables. Specifically, from which of the three memory spaces is a particular variable allocated? By default, if no C extensions are used, all variables are allocated from a single memory space, therefore allowing no confusion. The Keil C compiler allows the user to select a “memory model” from one of three possible models. These are the small, compact, and large models. The small and large models are of particular interest in targeting the Core8051s. These are described in the following sections.

Small Model

In this model, all variables, by default, reside in internal data memory. In this model, variable access is very efficient. However, all objects (if not explicitly located in another memory area) and the stack must fit into internal RAM. Stack size is critical because the stack size depends on the nesting depth of the various functions.

Large Model

In the large model, all variables, by default, reside in external data memory (which may be up to 64 kbytes). In the case of Core8051s, this covers 60 kbytes of external RAM and 4 kbytes of memory-mapped peripherals. The data pointer (DPTR) is used to address external memory, which results in slower accesses to variables than in the small model. It is likely, however, that the large model is the more appropriate of the two for targeting Core8051s without having to use language extensions, as this allows the peripheral resources to be mapped as C variables.

Proprietary Extensions to C for 8051

As mentioned above, the user may decide to write the application in portable ANSI C. However, many users will make use of nonstandard extensions provided by the various C compilers, to make more optimal use of the 8051 architecture. In particular, the features of the 8051 architecture that are of

interest are the address/data path widths as well as the different memory spaces. C compilers for the 8051 provide some extensions to C, which allow more efficient use of the 8051 memory spaces.

Memory Types

Different memory types are specified. For example, Table 5-10 summarizes some of the memory type specifiers, which may be used with the Keil Cx51 compiler.

***Table 5-10* • Memory Type Specifiers for Keil Cx51 Compiler**

Memory Type	Description
code	Program memory (64 kbytes); accessed by opcode MOVC @A + DPTR.
data	Directly addressable internal data memory. This gives the fastest access to variables (128 bytes).
idata	Indirectly addressable internal data memory. Variables with this type may be accessed across the full internal address space (256 bytes).
bdata	Bit-addressable internal data memory. This supports mixed bit and byte access.
xdata	External data memory (64 kbytes). This is accessed by opcode MOVX @DPTR.

As with signed and unsigned attributes, the memory type specifiers may be included in the variable declaration. For example:

```
char data var1;
char code text[] = “ENTER PARAMETER:”;
unsigned long xdata array[100];
float idata x,y,z;
unsigned char xdata vector[10][4][4];
char bdata flags;
```

If no memory type is specified for a variable, the compiler implicitly locates the variable in the default memory space determined by the memory model: **SMALL** or **LARGE**. Function arguments and automatic variables that cannot be located in registers are also stored in the default memory area.

Data Types

As well as the standard data types, 8051 C compilers also define specific data types, which may be used in the C code. For example, the Keil Cx51 compiler specifies the additional data types shown in Table 5-11.

***Table 5-11* • Cx51 Additional Data Types**

Data Types	Bits	Bytes	Value Range
bit	1		0 or 1
sbit	1		0 or 1
sfr	8	1	0 to 255
sfr16	16	2	0 to 65535

Note that data types relate to the sizes of the standard data types, as implemented by C compilers for the 8051. The following sizes are used:

***Table 5-12* • Size of Standard C data Types for 8051 Compilers**

Data Type	Size (bits)
char	8
int	16
long	32

***Table 5-12* • Size of Standard C data Types for 8051 Compilers**

float	32
double	64

Pointers

Because of the unique nature of the 8051 architecture, management of variable pointers becomes an issue. For example, the address of a variable in internal data memory is 8 bits and so a pointer to a variable in this space is 8 bits. Similarly, a pointer to a variable in external data or program memory is 16 bits wide.

Memory-Specific Pointers

Memory-specific pointers always include a memory type specification in the pointer declaration and always refer to a specific memory area. For example:

```
char data *str;  /* ptr to string in data */
int xdata *numtab;  /* ptr to int(s) in xdata */
long code *powtab;  /* ptr to long(s) in code */
```

Memory-specific pointers can be stored using only one byte (idata, data, bdata pointers) or two bytes (code and xdata pointers).

Generic Pointers

The Keil Cx51 compiler allows the use of generic pointers. Generic pointers are declared like standard C pointers. For example:

```
char *s;  /* string ptr */
int *numptr; /* int ptr */
```

Generic pointers are always stored using three bytes. The first byte is the memory type, the second is the high-order byte of the offset, and the third is the low-order byte of the offset. Generic pointers may be used to access any variable, regardless of its location in 8051 memory space. Code that uses generic pointers runs more slowly and is larger due to the conversion required and the need to link in other library routines. However, it is worthwhile if there is a need to mix different memory spaces. An example is the case where a display function is required to accept pointers to code for fixed message prompts and pointers to xdata for messages put together by software during execution. If a message stored in code space is passed to a display function that uses xdata space, the result is garbage.

In summary, by selecting a specific memory model and by the use of generic pointers and a modified runtime library, it is possible for a programmer to use ANSI C to target an 8051 derivative, such as Core8051s. To achieve better system performance and smaller code size, however, the user may utilize language extensions specified by the C compiler.

C Header Files

reg51.h

A customized version of the reg51.h file is required when compiling C code for Core8051s. This contains the following:"

```
/*-------------------------------------------------------------------------
reg51.h

Header file for Actel Core8051s microcontroller.
Copyright (c) Actel Corporation 2006.
All rights reserved.
-------------------------------------------------------------------------*/

#ifndef __REG51_H__
#define __REG51_H__
```

```
/*  BYTE Registers  */
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr DPL1 = 0x84;
sfr DPH1 = 0x85;
sfr ICON = 0x88;
sfr DPS  = 0x92;
sfr XWB1 = 0x9A;
sfr XWB2 = 0x9B;
sfr XWB3 = 0x9C;
sfr XRB1 = 0x9D;
sfr XRB2 = 0x9E;
sfr XRB3 = 0x9F;
sfr IE   = 0xA8;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;

/*  BIT Register  */
/*  PSW   */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

#endif
"
```

stdio.h

Core8051s requires a custom-designed stdio library, as it doesn't contain the serial channel normally found in 8051-based microcontrollers.

6 – Instruction Timing

Program Memory Bus Cycle

The execution for instruction N is performed during the fetch of instruction N + 1. A program memory fetch cycle without wait states is shown in Figure 6-1. A program memory fetch cycle with wait states is shown in Figure 6-2 on page 52. A program memory read cycle without wait states is shown in Figure 6-3 on page 52. A program memory read cycle with wait states is shown in Figure 6-4 on page 53. Figure 6-1 through to Figure 6-12 on page 57 have been taken from the *Core8051 Datasheet*. The following conventions are used in Figure 6-1 to Figure 6-14 on page 57.

***Table 6-1* • Conventions Used in Figure 18 to Figure 31**

Convention	Description
Tclk	Time period of clk signal
N	Address of actually executed instruction
(N)	Instruction fetched from address N
N+1	Address of next instruction
Addr	Address of memory cell
Data	Data read from address Addrl
read sample	Point of reading the data from the bus into the internal register
write sample	Point of writing the data from the bus into memory
ramcs	Off-core signal is made on the base ramwe and clk signals

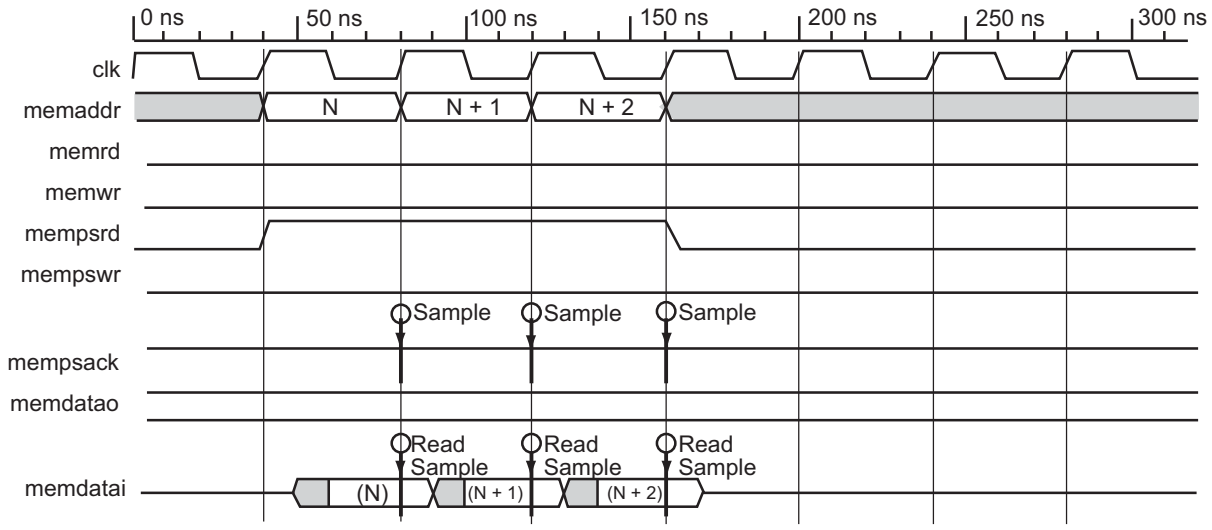

***Figure 6-1* • Program Memory Fetch Cycle Without Wait States**

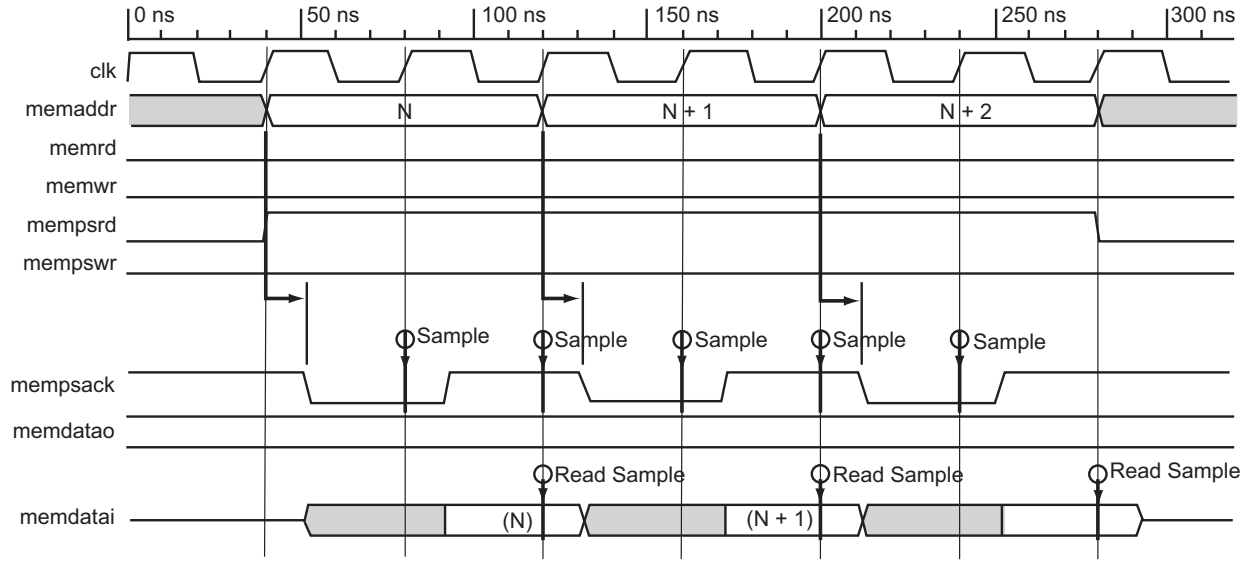

***Figure 6-2* • Program Memory Fetch Cycle With Wait States**

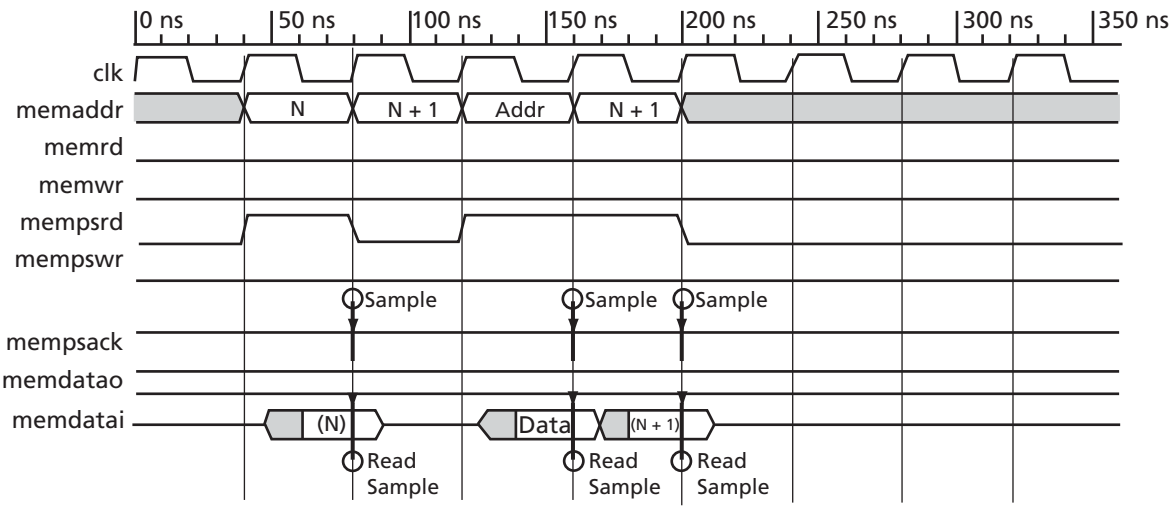

***Figure 6-3* • Program Memory Read Cycle Without Wait States**

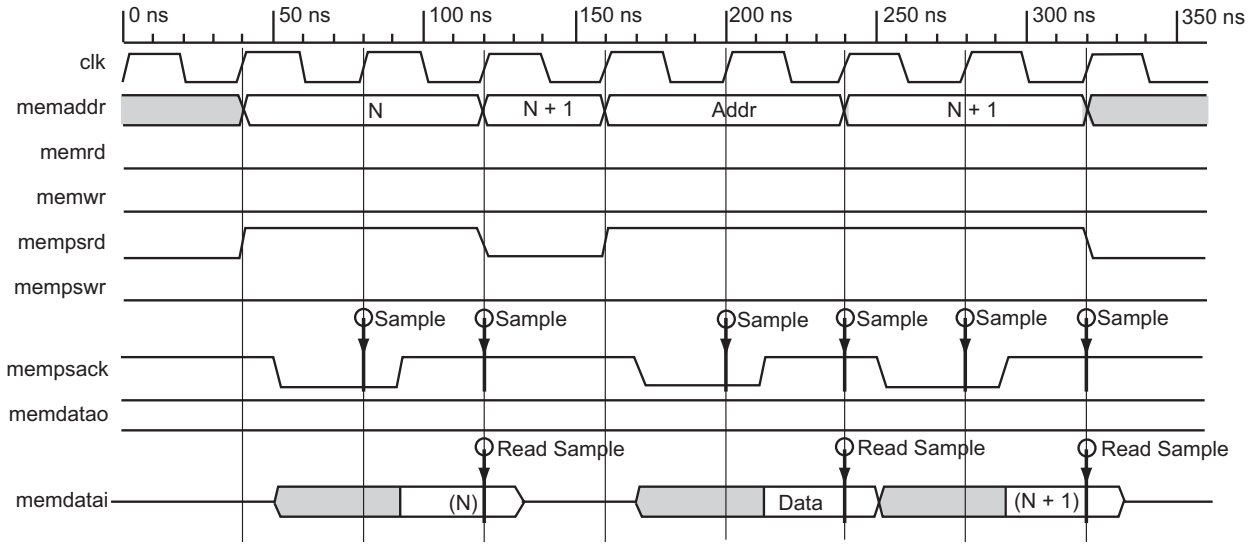

***Figure 6-4* • Program Memory Read Cycle with Wait States**

External Data Memory Bus Cycle

Example bus cycles for external data memory access are shown in Figure 6-5 through Figure 6-12 on page 57. Figure 6-5 on page 53 shows an external data memory read cycle without stretch cycles.

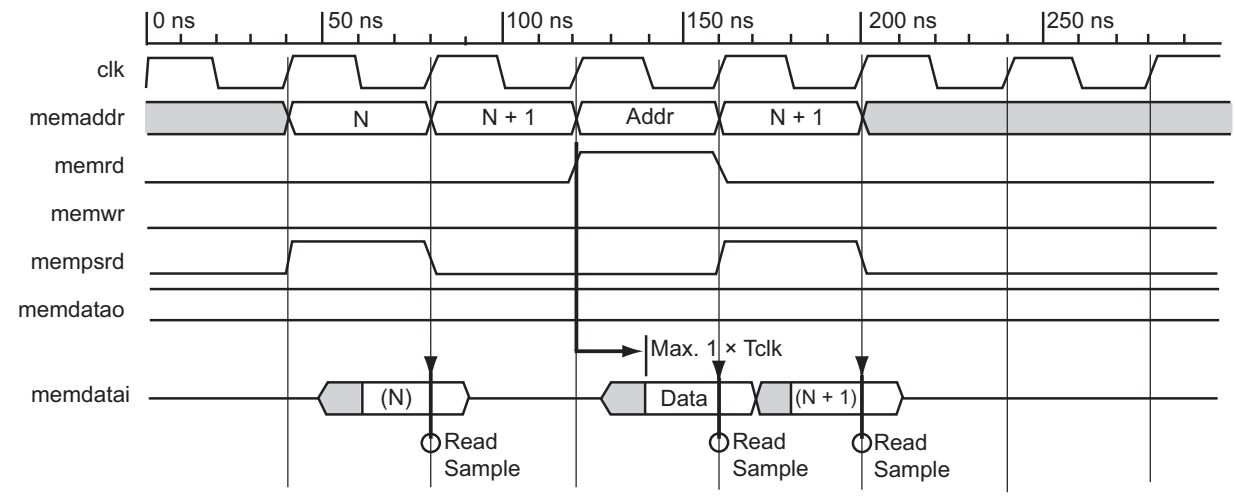

***Figure 6-5* • External Data Memory Read Cycle Without Stretch Cycles**

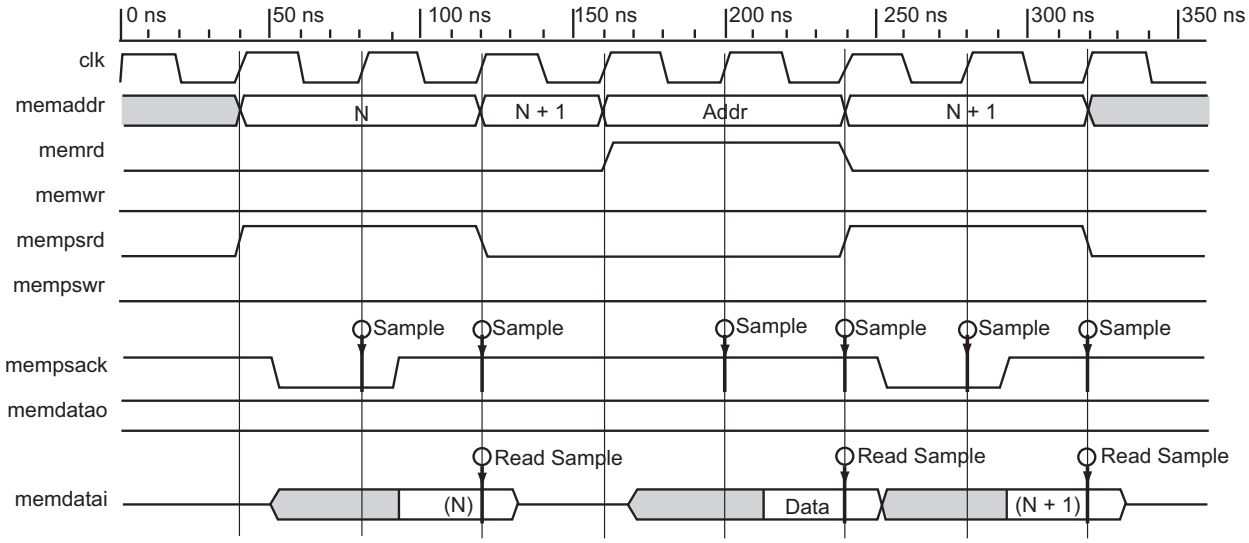

***Figure 6-6* • External Data Memory Read Cycle With One Stretch Cycle**

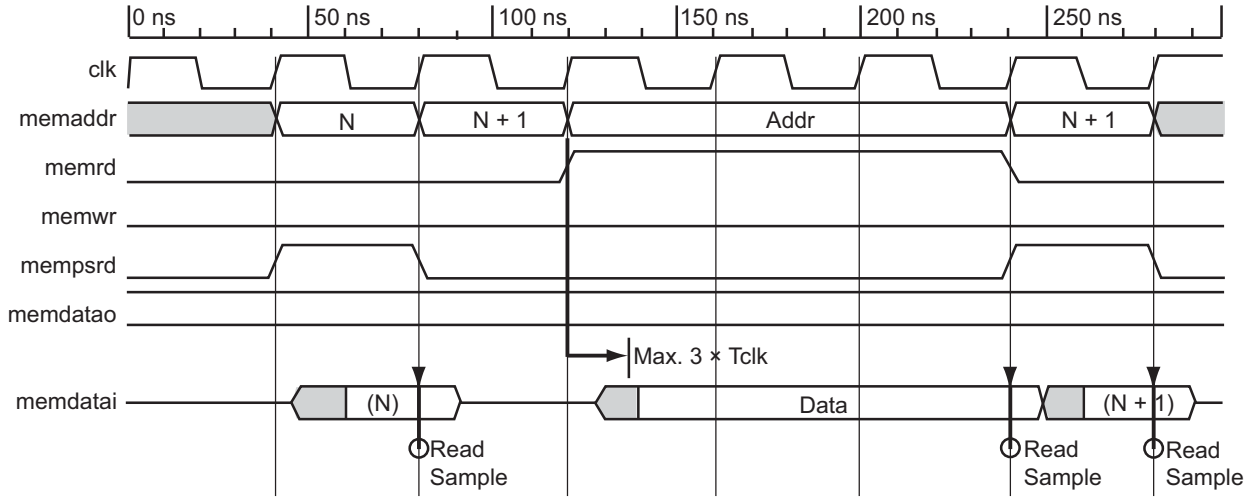

***Figure 6-7* • External Data Memory Read With Two Stretch Cycles**

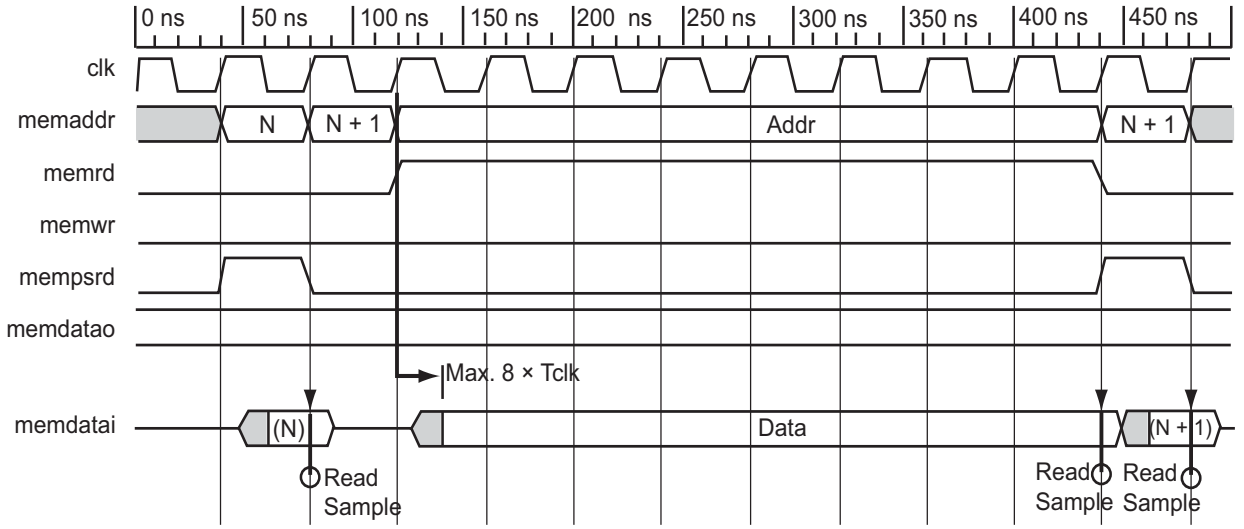

***Figure 6-8* • External Data Memory Read Cycle With Seven Stretch Cycles**

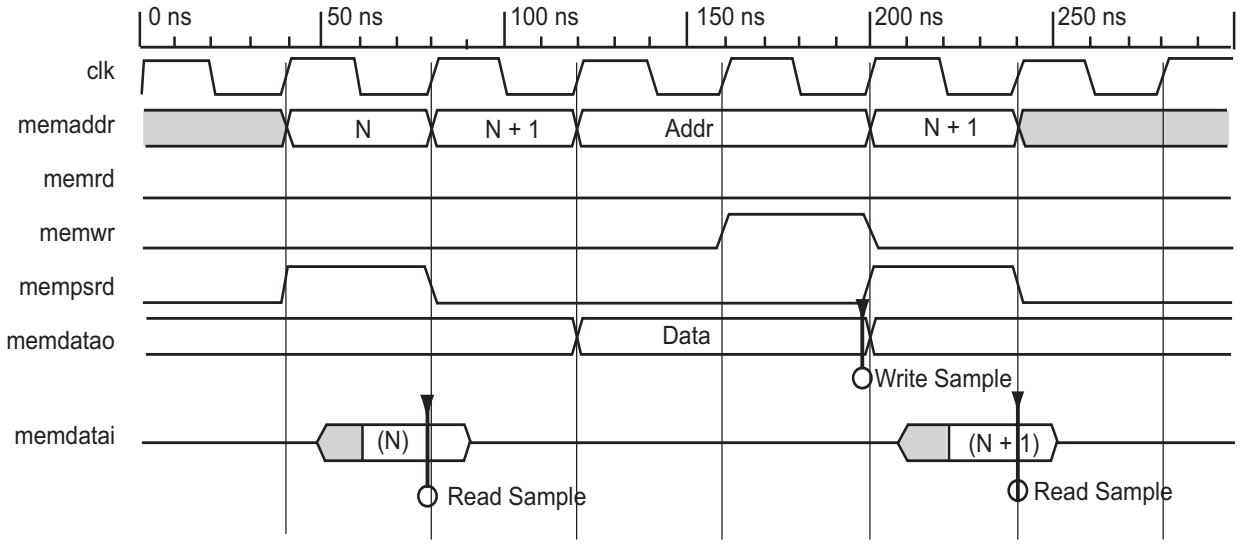

***Figure 6-9* • External Data Memory Write Cycle Without Stretch Cycles**

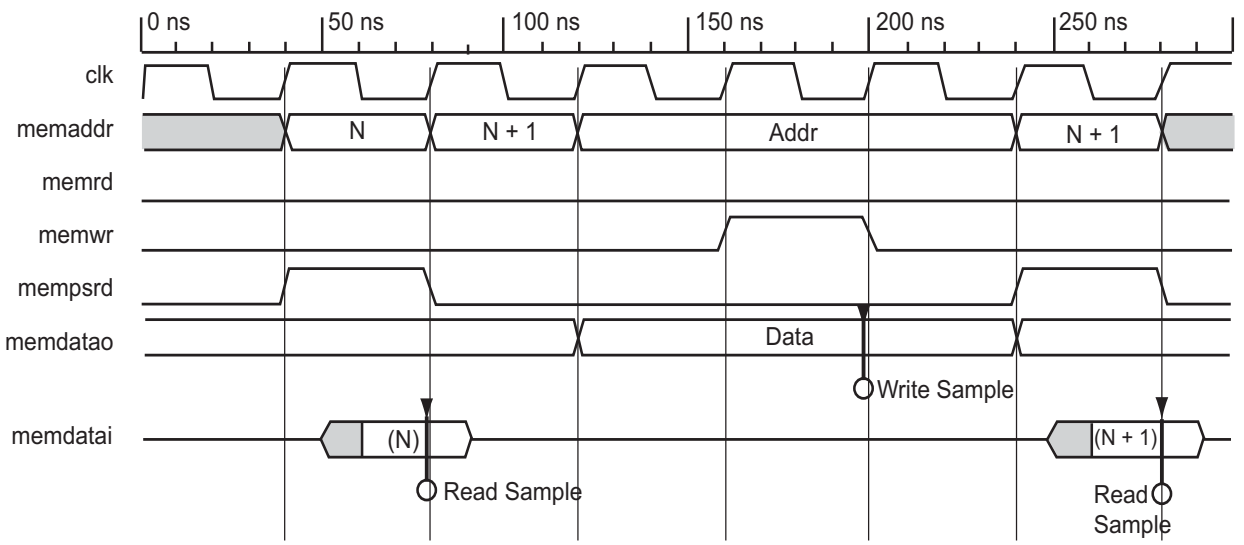

***Figure 6-10* • External Data Memory Write Cycle With One Stretch Cycle**

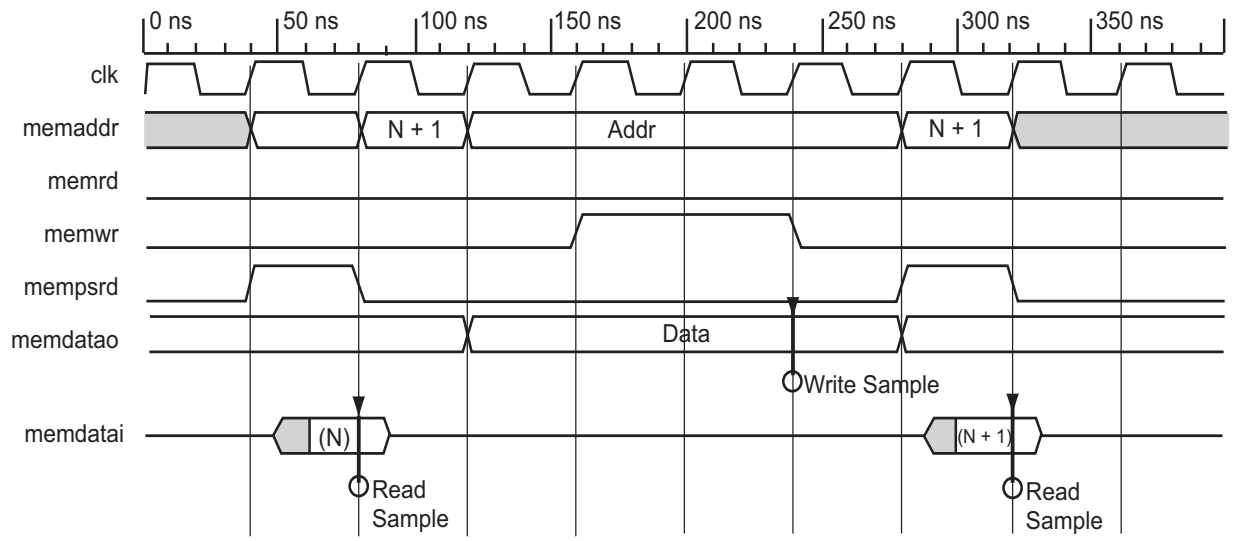

***Figure 6-11* • External Data Memory Write Cycle With Two Stretch Cycles**

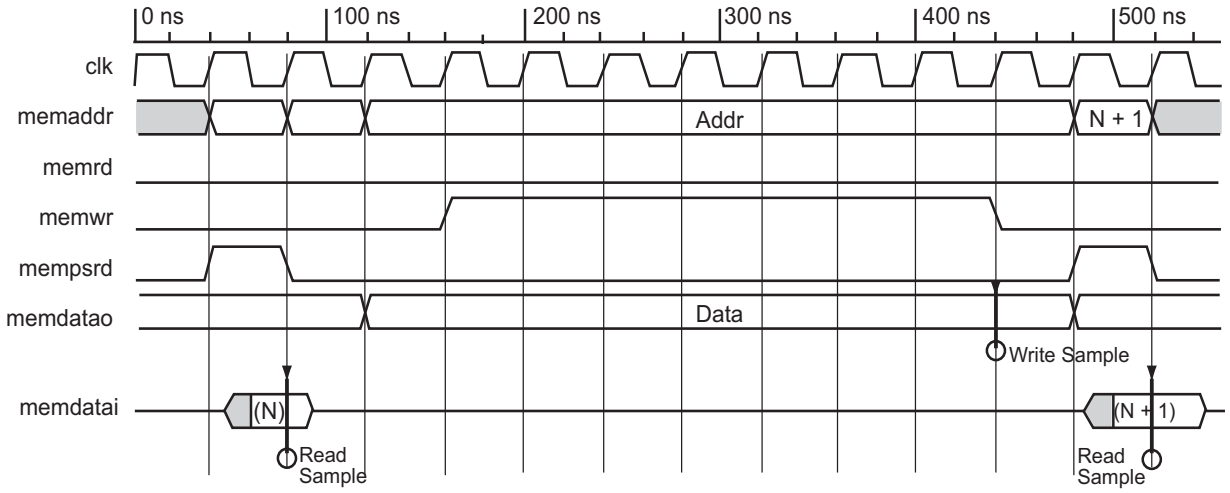

***Figure 6-12* • External Data Memory Write Cycle With Seven Stretch Cycles**

APB Bus Cycles

Example bus cycles for APB bus cycles are shown in Figure 6-13 and Figure 6-14.

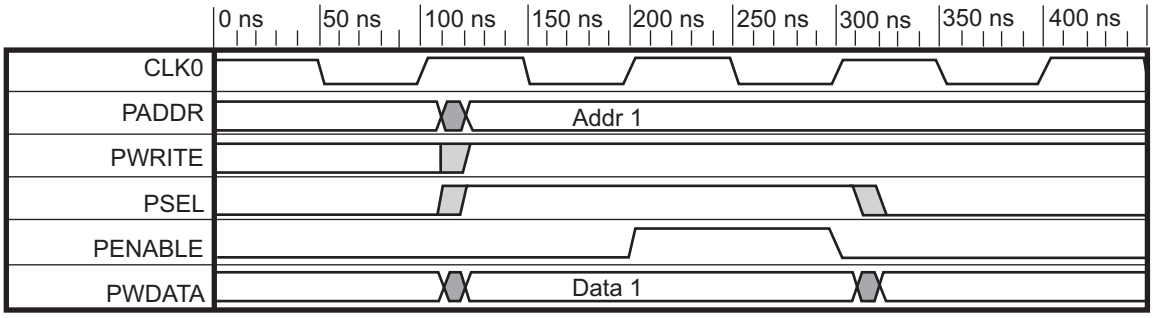

***Figure 6-13* • APB Write Transfer Bus Cycle**

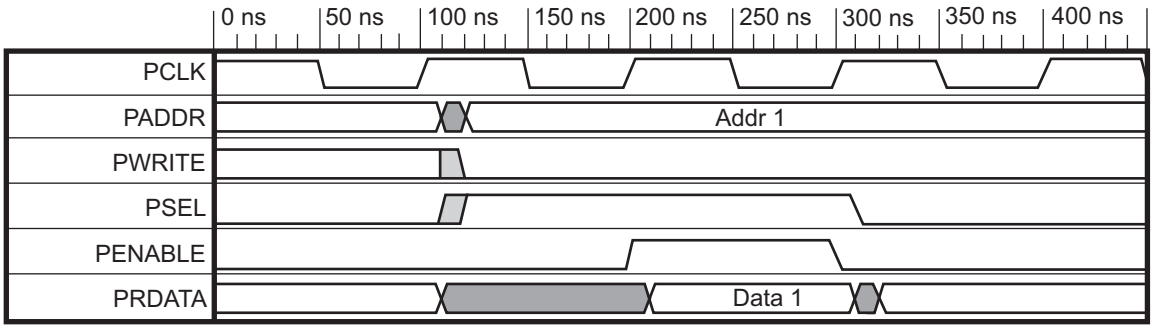

***Figure 6-14* • APB Read Transfer Bus Cycle**

7 – List of Changes

List of Changes

The following table lists critical changes that were made in each revision of the handbook

Date	Changes	Page
August 2010	The core version was updated to v2.4	N/A
	Type was changed from input to output for the MEMDATAO signal in Table 2-1 • Core8051s Ports.	18
	The "Optional Registers and Instructions" section was updated to state that the behavior of the processor is undefined when attempting to execute a MUL, DIV or DA instruction while the processor is not configured to include support for these instructions.	22
	The name of the Interrupt Enable register was changed from IEN to IE in Table 4-1 • Core8051s SFR Registers, the "Interrupt Enable Register (ie)" section, and the "Interrupts" section.	31, 33, 34
	The "Interrupt Control Register (icon)" section was revised to state that the ICON register implements a subset of the Timer Control (TCON) register.	33
	In Table 5-8 • Core8051s Instruction Set in Hexadecimal Order, the opcode "A5H*" was corrected. It had previously been listed as "ASH." The "B5H" opcode was corrected from "BSH." A footnote was added to the table stating that the A5H opcode is used as a trap instruction for the implementation of software breakpoints.	41, 43
	The "C Compiler Support" section was modified by adding the statement that the Small Device C Compiler (SDCC) is bundled with Actel's SoftConsole software development environment.	46

A – Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 a.m. to 6:00 p.m., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 a.m. to 6:00 p.m., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found on the website at www.actel.com/company/contact/default.aspx.

Index

Actel is the leader in low power FPGAs and mixed signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at www.actel.com.

Actel Corporation
2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.
River Court,Meadows Business Park
Station Approach, Blackwater
Camberley Surrey GU17 9AB
United Kingdom
Phone +44 (0) 1276 609 300
Fax +44 (0) 1276 607 540

Actel Japan
EXOS Ebisu Buillding 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668
http://jp.actel.com

Actel Hong Kong
Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

50200084-2/9.10