Core8051s Handbook

v2.0

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200084-0

Release: November 2006

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 kB allocated to external data memory, 4 kB are allocated to an APB-based peripheral bus and 60 kB is allocated to an external data memory interface
- Interrupt control logic for 2 interrupts

Supported Actel FPGA Families for the Core8051s are as follows:

- ProASIC$^{PLUS®}$
- ProASIC3®/E
- Fusion

Utilization and Performance

Core8051s can be implemented in several Actel FPGA devices. Table 1 gives typical utilization figures using standard synthesis tools for five different core configurations. The utilization and performance numbers for ProASIC3 and ProASIC3E families are identical to the Fusion numbers.

Table 1. Core8051s Utilization Data

Family	Data Width	Config.	RAM	Tiles	Device	Utilization	Frequency
Fusion	8	1	1	2,463	AFS600	18%	32
ProASIC	8	1	1	2,652	APA600	12%	17
Fusion	8	2	1	2,739	AFS600	20%	32
ProASIC	8	2	1	3,026	APA600	14%	19
Fusion	32	3	1	2,902	AFS600	21%	32
ProASIC	32	3	1	3,151	APA600	15%	20
Fusion	32	4	1	5,067	AFS600	37%	32
ProASIC	32	4	1	5,569	APA600	26%	20
Fusion	32	5	3	5,365	AFS600	39%	32
ProASIC	32	5	4	5,916	APA600	28%	19

The configurations referred to in Table 1 on page 5 are as follows:

Configuration 1

- No on-chip instrumentation (OCI)
- APB Data width = 8
- MUL, DIV, or DA instructions not present

Configuration 2

- No on-chip instrumentation (OCI)
- APB Data width = 8
- MUL, DIV, or DA instructions present

Configuration 3

- No on-chip instrumentation (OCI)
- APB Data width = 32
- MUL, DIV, or DA instructions present

Configuration 4

- On-chip instrumentation (OCI) present
  4 hardware event triggers
  No trace memory present
- APB Data width = 32
- MUL, DIV, or DA instructions present

Configuration 5

- On-chip instrumentation (OCI) present
  4 hardware event triggers
  Trace memory present
- APB Data width = 32
- MUL, DIV, or DA instructions present

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.

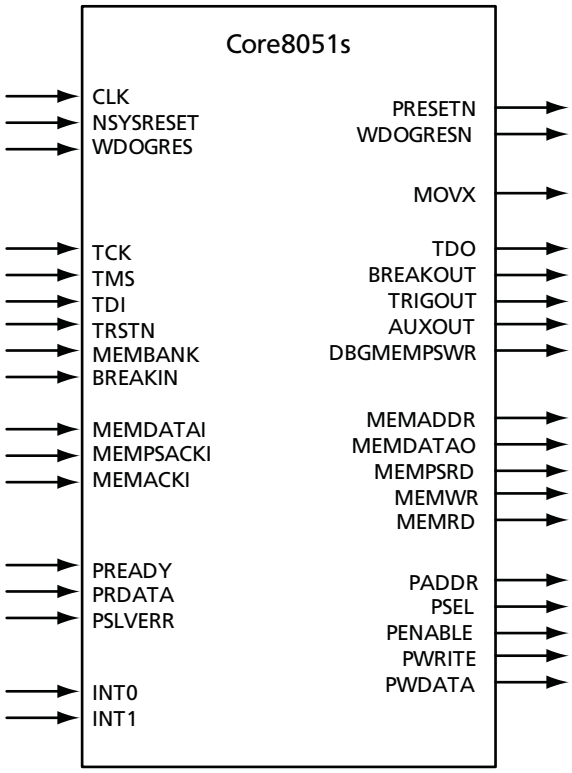

Figure 2-1. Core8051s I/O Signals

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

Table 2-1. Core8051s Ports (Continued)

Signal Name	Type	Polarity/Bus Size	Description
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	Input	8	Memory data output
MEMADDR	Output	16	Memory address
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.

Table 2-1. Core8051s Ports (Continued)

Signal Name	Type	Polarity/Bus Size	Description
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 RTL. These may be modified by the user to configure the Core8051s as required.

Table 2-2. Core8051s Parameters / Generics

Generics	Default Value	Description
USE_OCI	0	Set this to 1 to instantiate OCI logic.
INCL_TRACE	0	Set this to 1 to include OCI trace RAM.
TRIG_NUM	0	No triggers: Set value to 0. 1 trigger: Set value to 1. 2 triggers: Set value to 2. 4 triggers: Set value to 4.
APB_DWIDTH	32	APB data width—possible values are 8, 16, and 32
INCL_DPTR1	0	Set to 1 to include second data pointer (DPTR1).
INCL_MUL_DIV	1	Set to 1 to include multiply and divide instruction functionality.
VARIABLE_STRETCH	1	Set this parameter/generic to 1 to control the External Data Memory interface by MEMACKI. If set to 0, a fixed number of stretch cycles are added to each read or write of external data memory. The number of stretch cycles is given by the STRETCH_VAL parameter/generic.
STRETCH_VAL	1	Only applicable if VARIABLE_STRETCH is 0. This gives the number of stretch cycles to be added to reads or writes of external data memory. Range is 0 to 7.
VARIABLE_WAIT	1	Set this parameter/generic to 1 to control Program Memory interface by MEMPSACKI. If set to 0, a fixed number of wait cycles are added to each read of program memory. The number of wait cycles is given by the WAIT_VAL parameter/generic.
WAIT_VAL	0	Only applicable if VARIABLE_WAIT is 0. This gives the number of wait cycles to be added to reads of program memory. Range is 0 to 7.

3 Tool Flows

CoreConsole Operation

Core8051s systems may be deployed using the Actel CoreConsole tool. The following is a simple example of deploying a basic subsystem for the Core8051s. Basic peripherals and features are enabled in a step-by-step method. This system consists of a Core8051s, CoreGPIO, CoreUARTapb and CoreWatchdog.

1. Create a new CoreConsole Project by clicking on the **File** menu and selecting **New.**
2. In the **Components** Tab in **CoreConsole**, select the **Core8051s** processor and click **Add**. This will add the processor to the schematic window. See Figure 3-1.

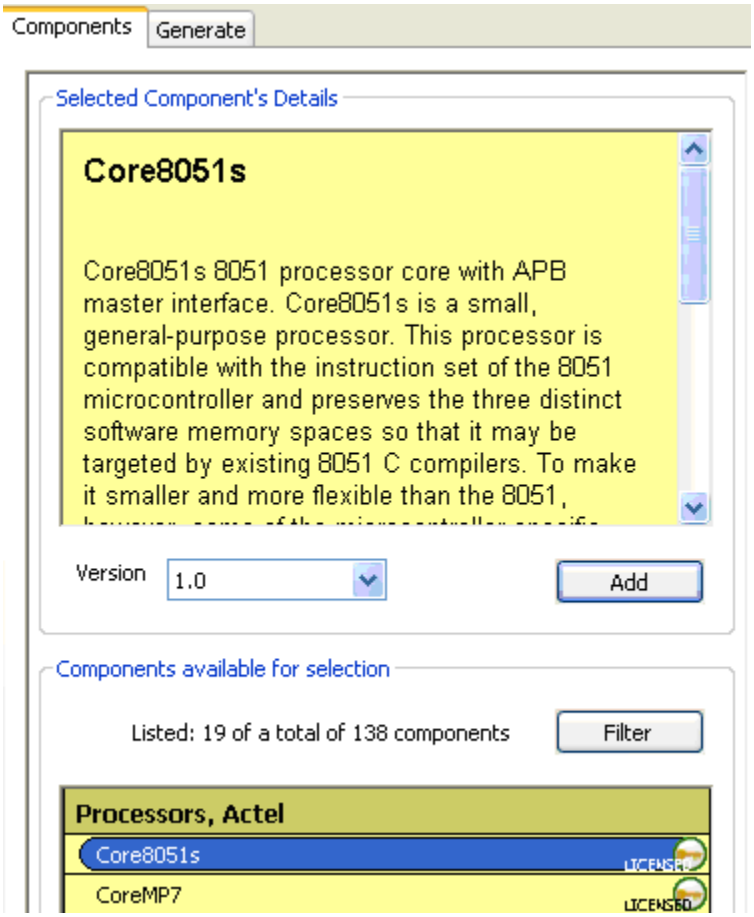

Figure 3-1. Example Adding Core8051s Processor

3. Now add the **CoreAPB3, CoreGPIO, CoreUARTapb,** and the **CoreWatchdog** in a similar fashion.

   Note: CoreAPB3 must be used for connecting Core8051s to APB peripherals. Core8051s is not compatible with CoreAPB. CoreAPB3 is not currently present in the CoreConsole v1.2.1 IP database. Therefore the user must import CoreAPB3 into CoreConsole in order to build a Core8051s-based system.

   Note also that the APB slot size in CoreAPB3 must be configured to be 256 locations.
4. You may connect these blocks manually in CoreConsole, or use CoreConsole's auto stitch feature.

5. Click the **Actions** menu and select **Auto Stitch**. A small window appears with various options to auto stitch. For this example, auto stitch only the default selections. Click **Stitch** to connect all the devices through their respective ports to their default connections in the APB. See Figure 3-2.

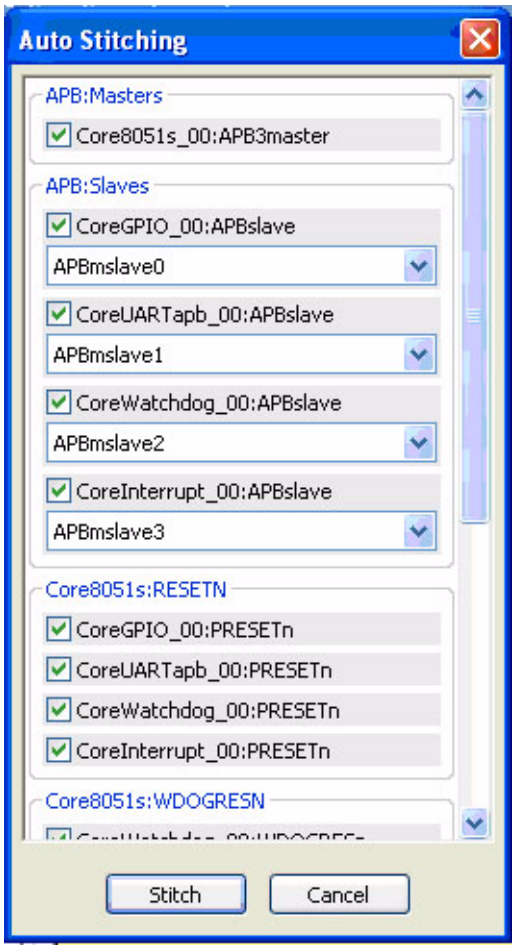

Figure 3-2. Example Auto Stitching

6. You may also need to bring connections from ports of the blocks within this subsystem to the top level of the design. For example, you can connect the program and data memory to the Core8051s at the top level. To auto

stitch to the top level, select the **Actions** menu once more, and select **Auto Stitch to Top level.** A new window appears with various options. See Figure 3-3.

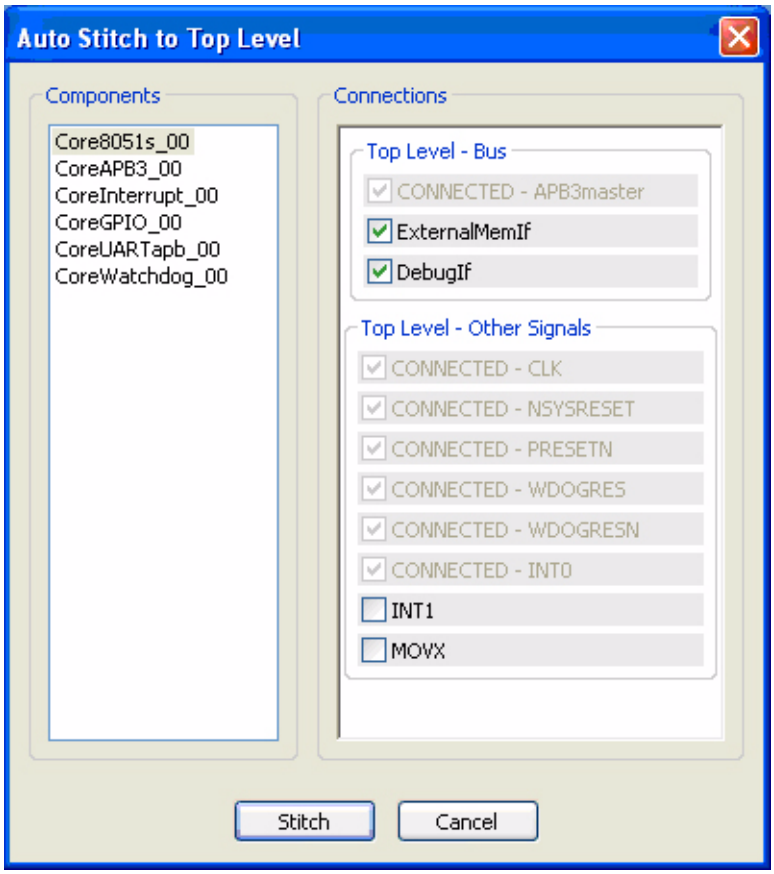

Figure 3-3. Example Auto Stitching to Top Level

- In the **Auto Stitch to Top Level** window, clear the check boxes for **MOVX** and **INT1**. These do not need to be connected to the top level for this example. Click **Stitch** when ready.

7. Auto stitch the other components in the following fashion. Select the **CoreUARTapb** component in the open window. Clear the **txrdy** and **receive_full** check boxes. These do not need to be connected to the top level for this example. Click **Stitch** when ready. See Figure 3-4.

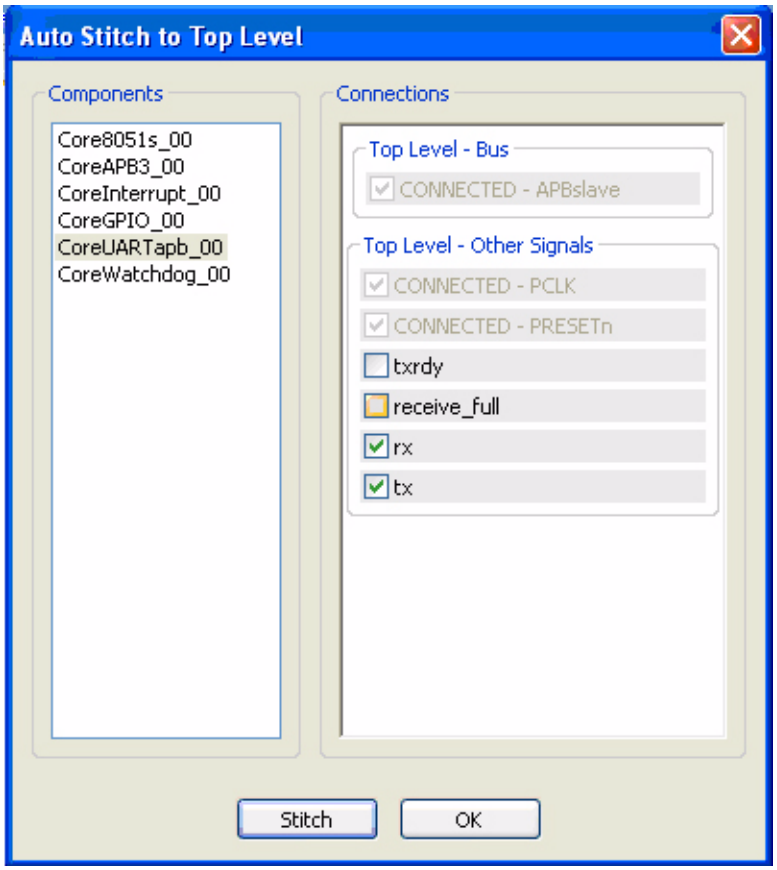

Figure 3-4. Example CoreUARTapb Component

8. Select the **CoreGIPO_00** component in the open window. None of the settings in this window need to be changed. Click **Stitch** when ready. See Figure 3-5.

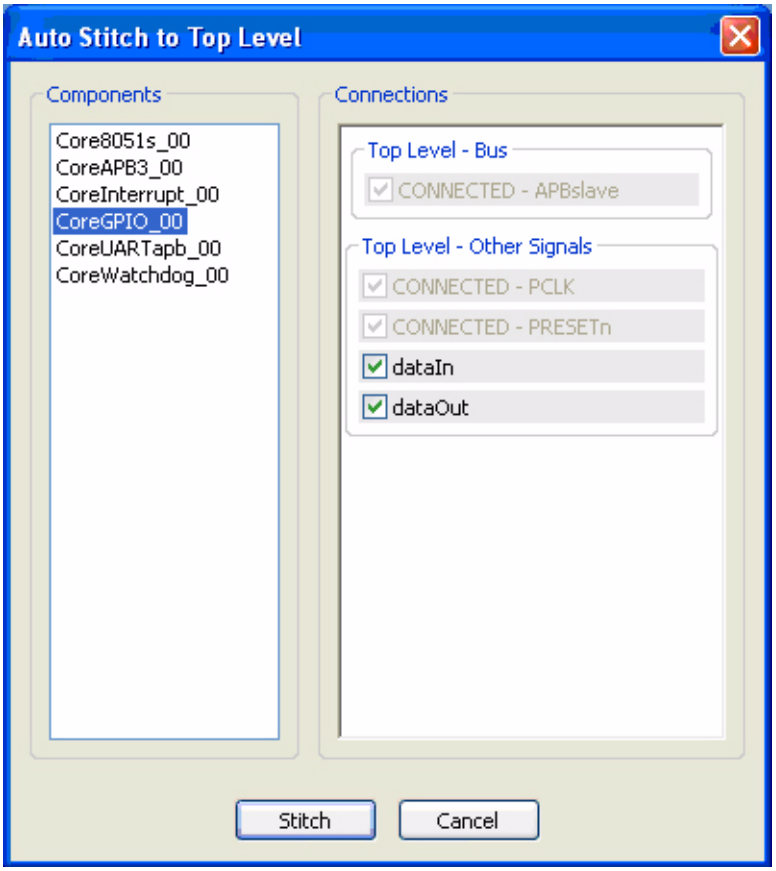

Figure 3-5. Example CoreGPIO_00 Components

9. In the **Actions** menu, select **Re-Center Design** to center the design. In the same menu, select **Auto-Layout** to lay out the schematic for ease of use.
10. The finished diagram should resemble Figure 3-6 on page 18. The order of the components may differ. On the upper right of the Schematic window, the ports in the subsystem have been connected to the top level. These include the data ports, external memory and debug interfaces, and the transmit/receive ports throughout the subsystem.

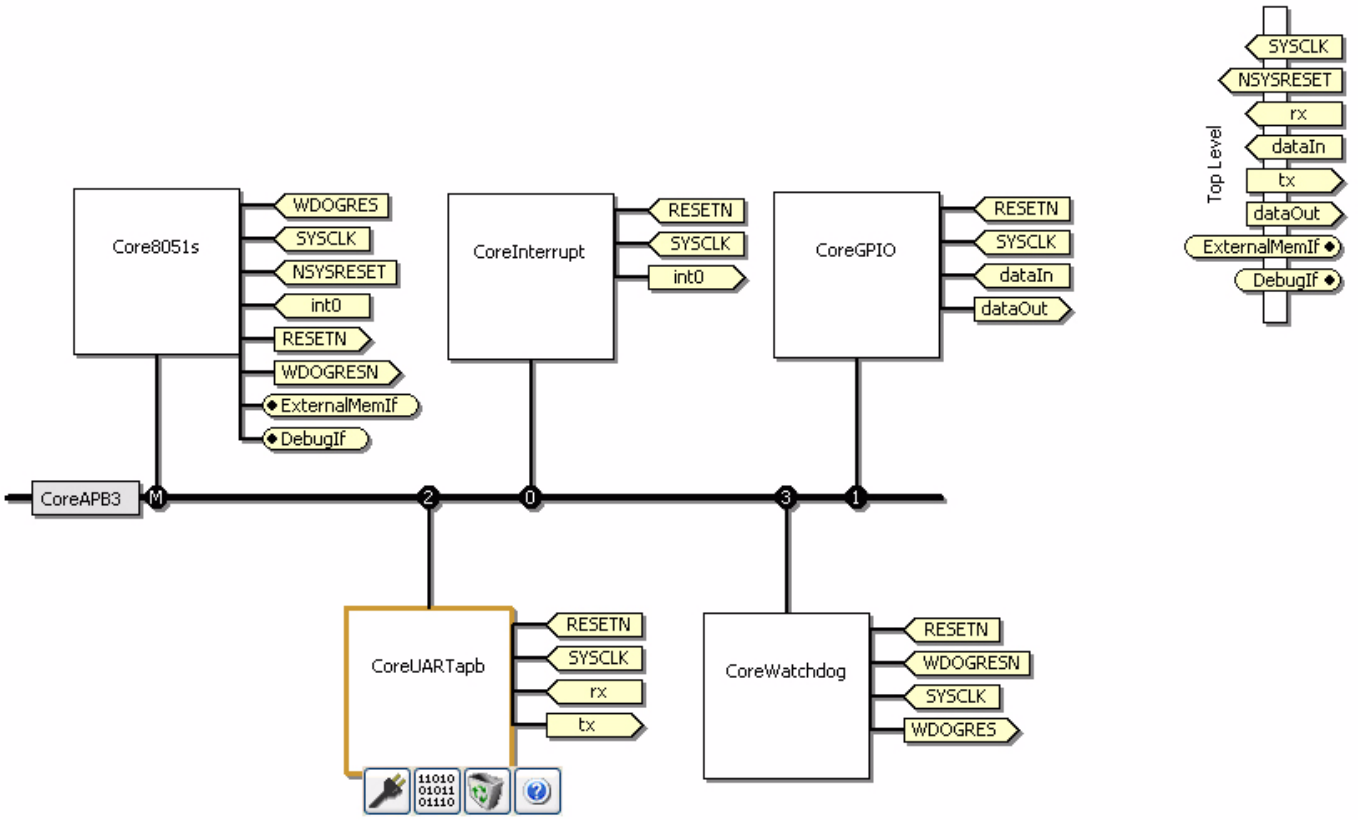

Figure 3-6. Example Finished Diagram

Configuring the Core

Now it is necessary to configure Core8051s for the system.

Float the mouse over the Core8051s component in the Schematic window. Select the **Configure** icon. The window shown in Figure 3-7 appears.

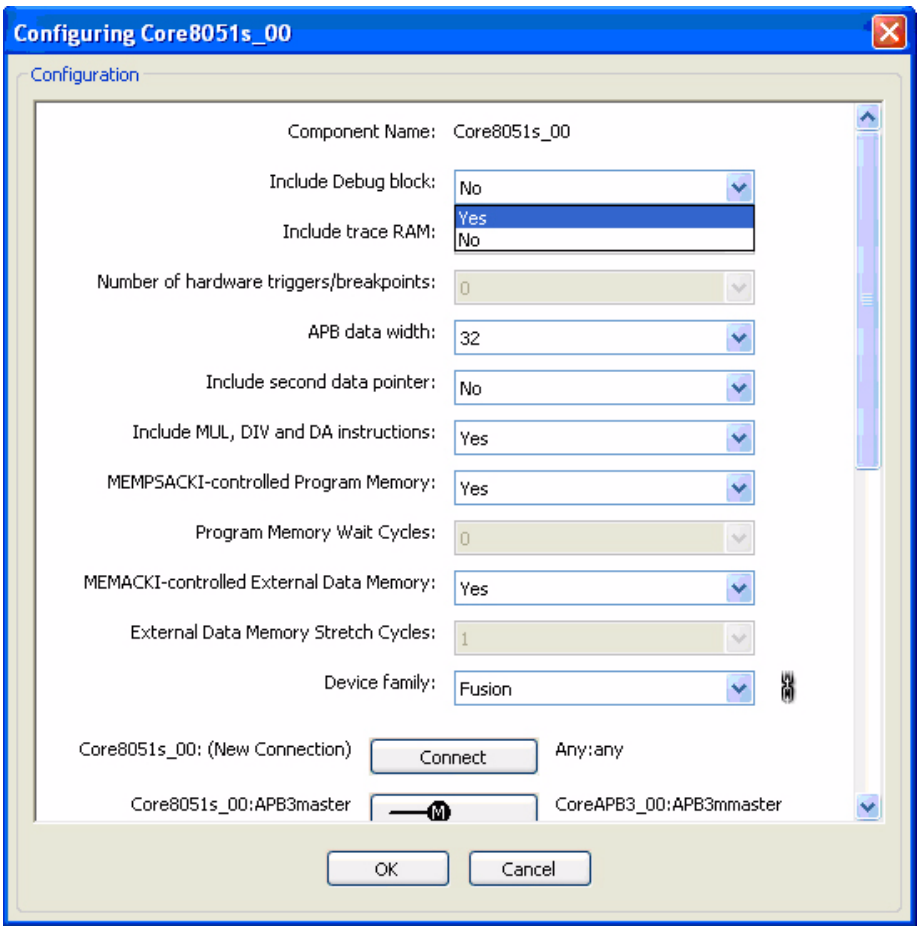

Figure 3-7. Example Configuring Core8051s

Various options are available in this window. The following sections cover each one individually and briefly explain their function.

Figure 3-7 shows a drop-down menu for Include OCI open. Two options are available, Yes and No. OCI refers to the on-chip instrumentation and enables and disables the internal trace RAM and hardware event triggers. The two boxes below this are greyed out until Include OCI is set to Yes.

Figure 3-8 is the Include trace RAM box. If user selects “Yes”, then a 256 deep trace RAM is instantiated within the Core8051s. Otherwise, no trace RAM is present.

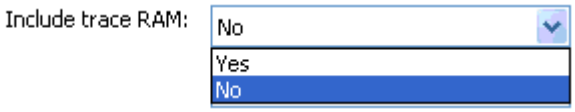

Figure 3-8. Example Include Trace RAM

With the OCI enabled, it is possible to set how many hardware triggers/breakpoints are desired in the Number of hardware triggers/breakpoints drop-down menu, ranging from 0 to 4 (Figure 3-9).

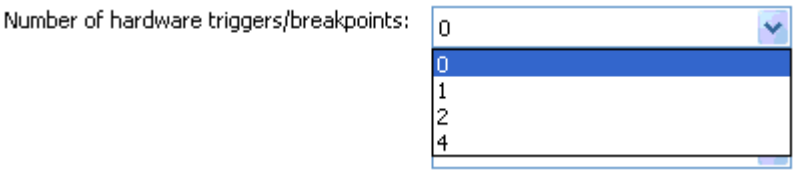

Figure 3-9. Example Number of Hardware Triggers/Breakpoints

It is also possible to set the APB data width, if smaller data width peripherals are desired. The default data width is 32 bits (Figure 3-10)

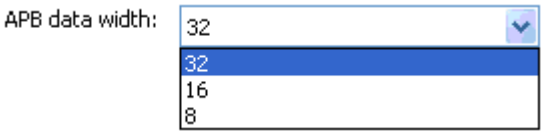

Figure 3-10. Example APB Data Width

Not enabled by default, it is possible to enable the second data pointer in this configure window. The first data pointer is data pointer 0 (Figure 3-11).

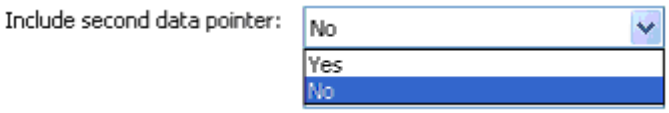

Figure 3-11. Include Second Data Pointer

There is specific hardware in the processor for MUL, DIV, and DA that can be enabled and disabled in the configuration. If MUL, DIV, and DA are not used in the any of the associated software, disabling this saves logic tiles (Figure 3-12).

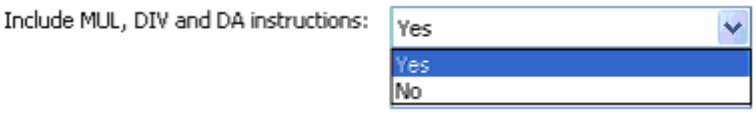

Figure 3-12. Example Include MUL, DIV, and DA Instructions

Note: MUL, DIV, and DA should only be omitted if the user is sure the program does not use them. For example, these instructions should be included if using a C compiler.

The program memory interface may be configured as either variable delay (i.e., controlled by the MEMPSACKI signal) or fixed delay (i.e, a fixed number of wait cycles). This is done using the drop-down menu in Figure 3-13.

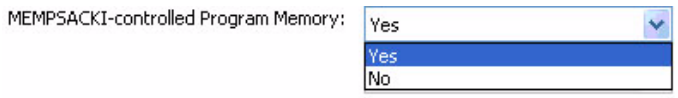

Figure 3-13. MEMPSACKI-Controlled Program Memory

If the user chooses not to have MEMPSACKI control Program Memory, the drop-down menu shown in Figure 3-14 becomes active.

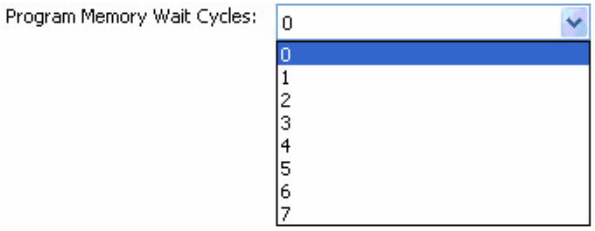

Figure 3-14. Number of Wait Cycles in Program Memory

The external data memory interface can be configured as either a variable delay (i.e., controlled by the MEMACKI signal) or a fixed delay (i.e., a fixed number of stretch cycles). This is done using the drop-down menu shown in Figure 3-15.

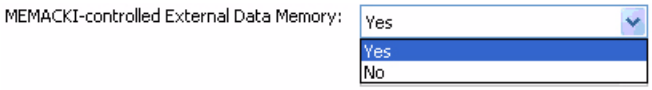

Figure 3-15. MEMACKI-Controlled External Data Memory

If the user chooses not to have External Data Memory controlled by MEMACKI, the drop-down menu shown in Figure 3-16 becomes active.

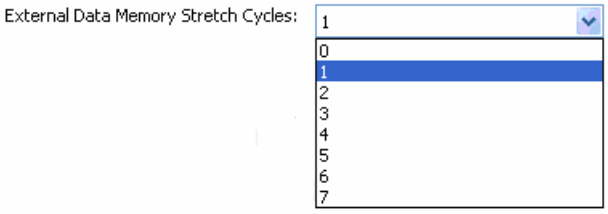

Figure 3-16. Number of Stretch Cycles in External Data Memory

In the device family it is possible to select from a range of supported device families for specific use within the processor subsystem (Figure 3-17).

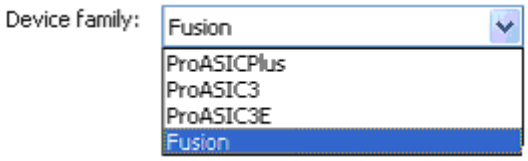

Figure 3-17. Device Family

Memory Map Generation

CoreConsole can be used to generate the memory map of the APB peripherals. The memory map shows the location of each peripheral relative to the base of APB space, which is at 0xF000 in external data memory space. Select **View > Memory Map**, to obtain the dialog box shown in Figure 3-18.

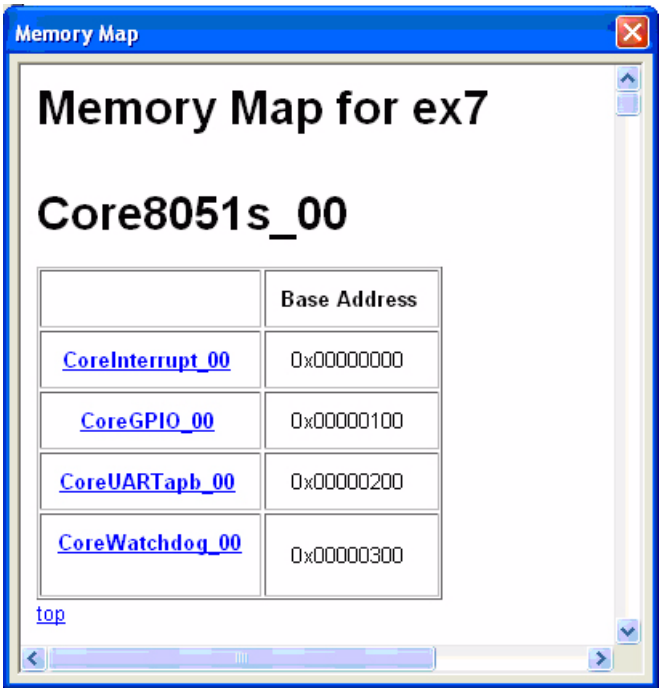

Figure 3-18. CoreConsole Generated APB Memory Map

For example, inFigure 3-18, CoreUARTapb is reported to be located at 200H relative to the base of the APB address space (which is F000H). Therefore, the software programmer knows to find the CoreUARTapb at location F200H in external data memory space.

The rest of the options in the Configure Core8051s window are all connection-related, and these show the signal connections as described earlier. See Figure 3-19.

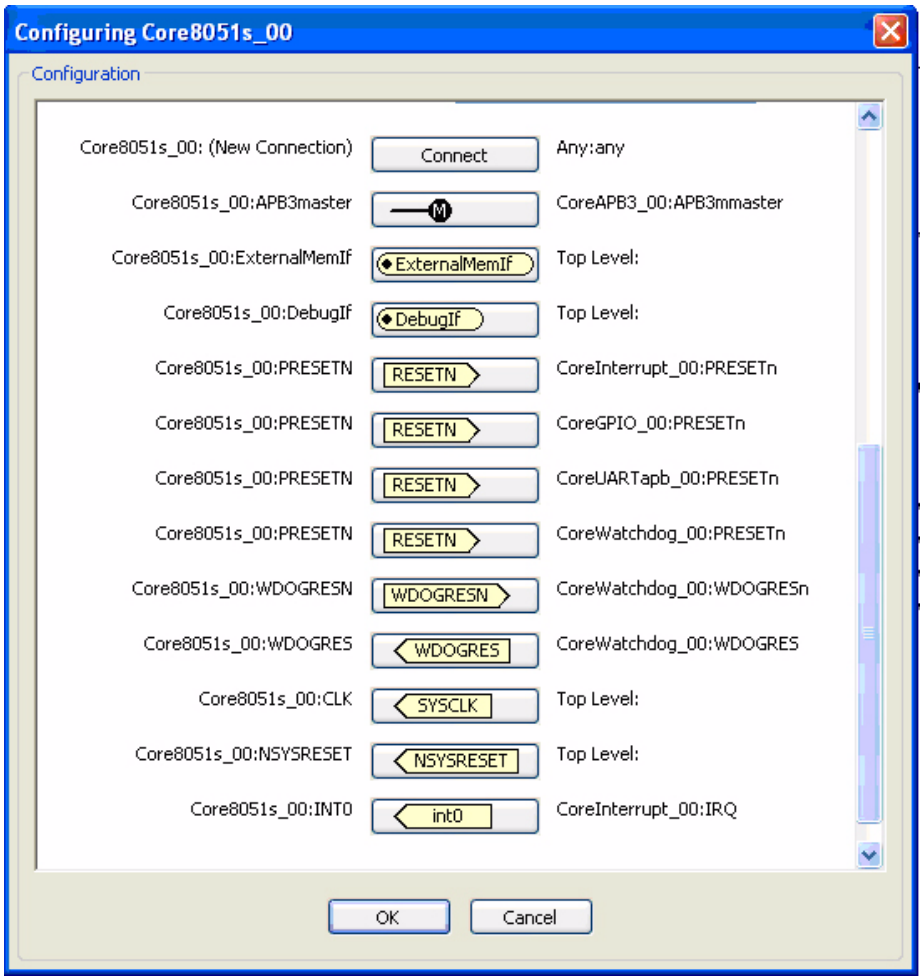

Figure 3-19. Example Configuring Core8051s_00 Connections

Simulation Flows

Three simulation testbenches are supported with Core8051s:

- BFM-based tests of User System (using BFM scripts)
- User testbench
- Full Core8051s verification testbench

The first one allows testing of the user's Core8051s-based system, as deployed in CoreConsole. The second two are for unit testing of the Core8051s.

When CoreConsole generates the Actel Libero® Integrated Design Environment (IDE) project, it installs the appropriate testbench files. To run the APB-based simulation, set the design root to the top level within Libero IDE. To run either the simple application or full verification environment, set the design root to Core8051s instantiation in the Libero IDE file manager.

BFM-Based Tests

The BFM testbench and BFM script are generated dynamically to match the exact configuration of the user's system in the CoreConsole project.This testbench uses a Bus Functional Model (BFM) to generate a series of APB bus cycles. These are used to read and write to 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 the designer can exercise the system using a simple scripting language, before getting to the stage of having C code or 8051 assembler code written.

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 CoreConsole, 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.

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, as defined by the user in the memory map (when input to CoreConsole).

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, as defined by the user in the memory map (when input to CoreConsole).

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, as defined by the user in the memory map (when input to CoreConsole).

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 CoreMP7 clock periods, during which the BFM stalls (doesn't initiate any bus transactions).

waitfiq

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

Syntax

```
waitfiq;
```

waitirq

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

Syntax

```
waitirq;
```

User Testbench

Using the supplied user testbench as a guide, the user can easily customize the verification of the core by adding or removing tests. The user testbench provides a sample system that contains a Core8051s processor running 8051 assembly code from program memory. This provides a reference system, which the user can adapt or extend.

Verification Testbench

The comprehensive verification simulation testbench verifies correct operation of the Core8051s IP Core. The verification testbench applies several tests to the Core8051s IP Core, including the following:

- Operation code tests
- Miscellaneous tests
- APB tests

Set the design root in the Libero IDE to the Core8051s instantiation, and then click the **Simulation** icon. Libero IDE should automatically associate the stimulus files and compile the files.

The runsim_ver.do script, which is automatically invoked when Model*Sim*® is started from within Libero IDE, can be invoked manually by typing **do runsim.do** within the Model*Sim* GUI. The script supports optional parameters:

Synthesis

Synthesis in the Libero IDE

Having set the design route appropriately, click the **Synthesis** icon in the Libero IDE. The synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the **Verilog 2001** standard if Verilog is being used. To run synthesis, select the **Run** icon.

Place-and-Route in Libero IDE

Having set the design route appropriately and run Synthesis, click the **Layout** icon in the Libero IDE to invoke Designer. Core8051s requires no special place-and-route settings.

Synthesis Constraints

Use proper synthesis constraints to ensure correct compilation of the design. These constraints are synthesis tool specific.

Layout

Layout constraint files generated by Synplify® are automatically fed into Designer, if invoked using Libero IDE.

The user may also supply pin constraints to the Designer. A GCF file is required if using the ProASIC^{PLUS} device family. A PDC file is required if using ProASIC3, ProASIC3E or Fusion device families.

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.

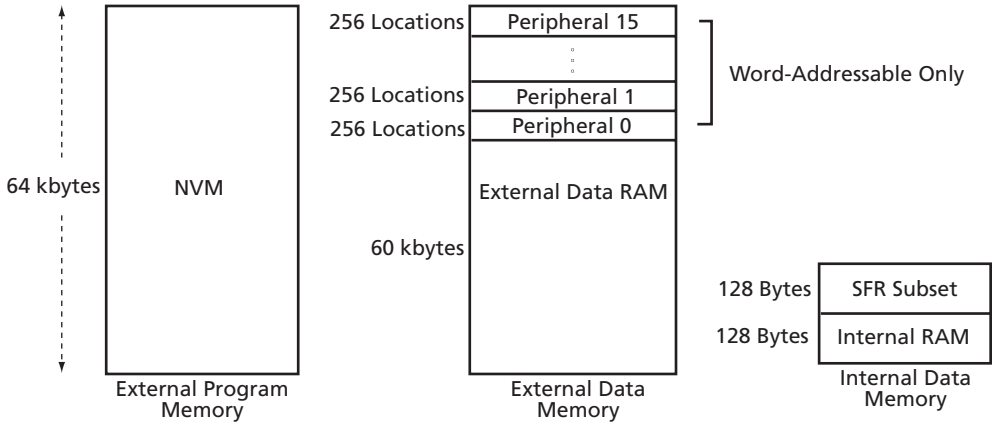

Figure 4-1. Core8051s Software Memory Map

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 kB 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. This is configured by using a hardware parameter/generic in CoreConsole.

External Data Memory Space

Core8051s can address up to 64 kB 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 kB (F000H to FFFFH) of external data memory space is mapped to an APB bus. The lower 60 kB is mapped to the external memory bus interface (Table 4-1 on page 31).

External Data Interface

The external memory bus interface (Table 4-1 on page 31) 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. This is configured by using a hardware parameter/generic in CoreConsole.

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.

There are 16 APB slots available, each with 256 locations accessible. These locations are assigned as follows:

- 256 bytes if the APB peripheral is 8 bits
- 256 half words if the APB peripheral is 16 bits
- 256 words if the APB peripheral is 32 bits

For the 4 kB of memory space allocated to the APB subsystem, only “word” access is possible, where “word” refers to an 8-bit, 16-bit, or 32-bit entity, for 8-bit, 16-bit, or 32-bit APB bus implementations respectively. As shown in Figure 4-1 on page 29, Core8051s supports 16 APB peripherals via connection to CoreAPB3.

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)
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)
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.

Interrupts

There are two interrupts in Core8051s. These are both level-sensitive and asserted HIGH. You must drive these interrupts from a CoreInterrupt. The software can enable or disable the interrupts by accessing CoreInterrupt.

INT0 is low priority, with a vector address of 03H. INT1 is high priority, with a vector address of 13H.

The interrupt service routine must clear the source of the interrupt before exiting the ISR. 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.

OCI Block

The on-chip instrumentation (OCI) block may be configured by the user as being present or not. This block communicates with external debugger hardware and software as a debugging aid to the user. 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 kB program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2 kB 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. 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

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

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
ASH[1]	-	BSH	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

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

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).

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 32 kbytes of external RAM and 32 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
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 DPS  = 0x92;
sfr XWB1 = 0x9A;
sfr XWB2 = 0x9B;
sfr XWB3 = 0x9C;
sfr XRB1 = 0x9D;
sfr XRB2 = 0x9E;
sfr XRB3 = 0x9F;
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 studio 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 53. 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 58.

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

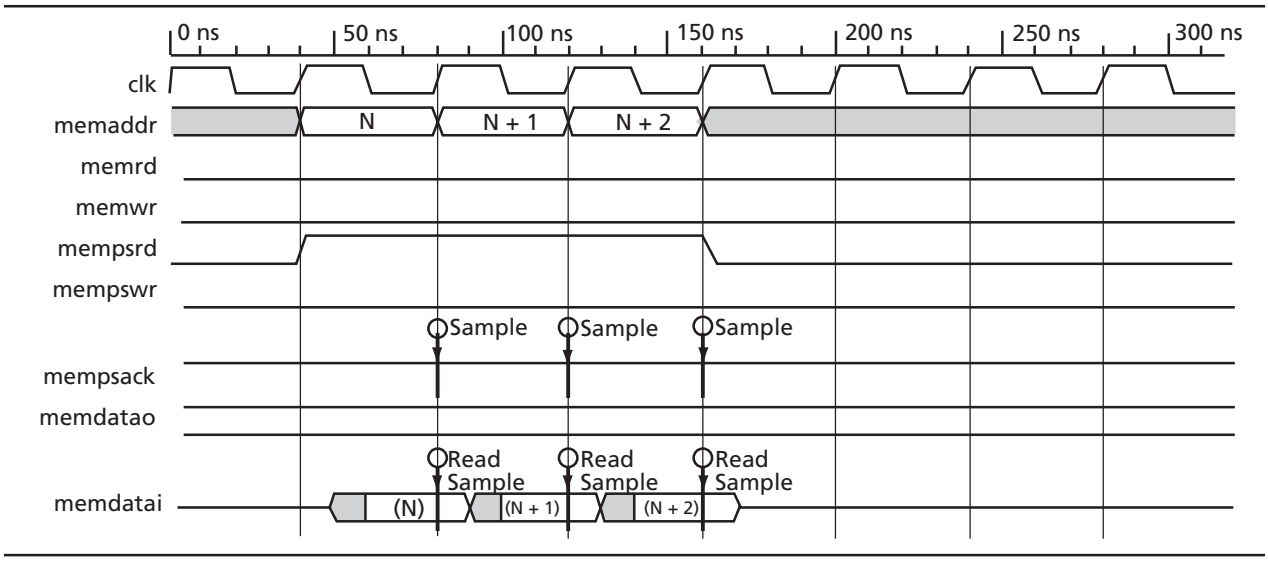

Figure 6-1. Program Memory Fetch Cycle Without Wait States

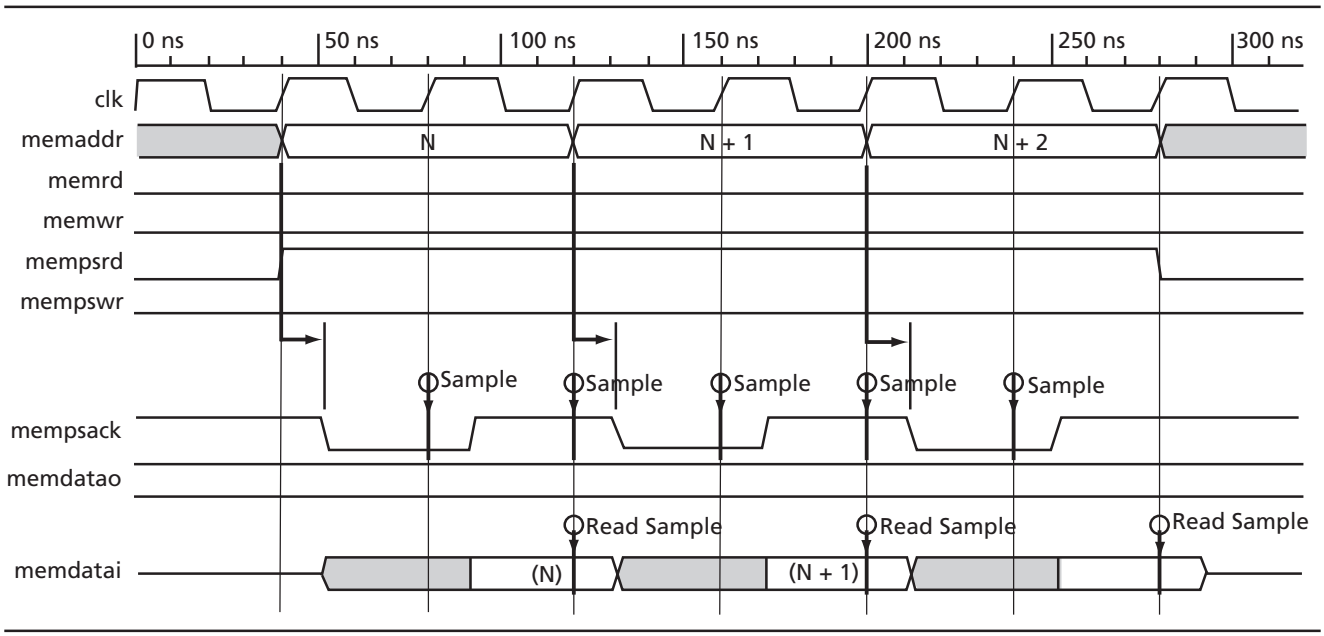

Figure 6-2. Program Memory Fetch Cycle With Wait States

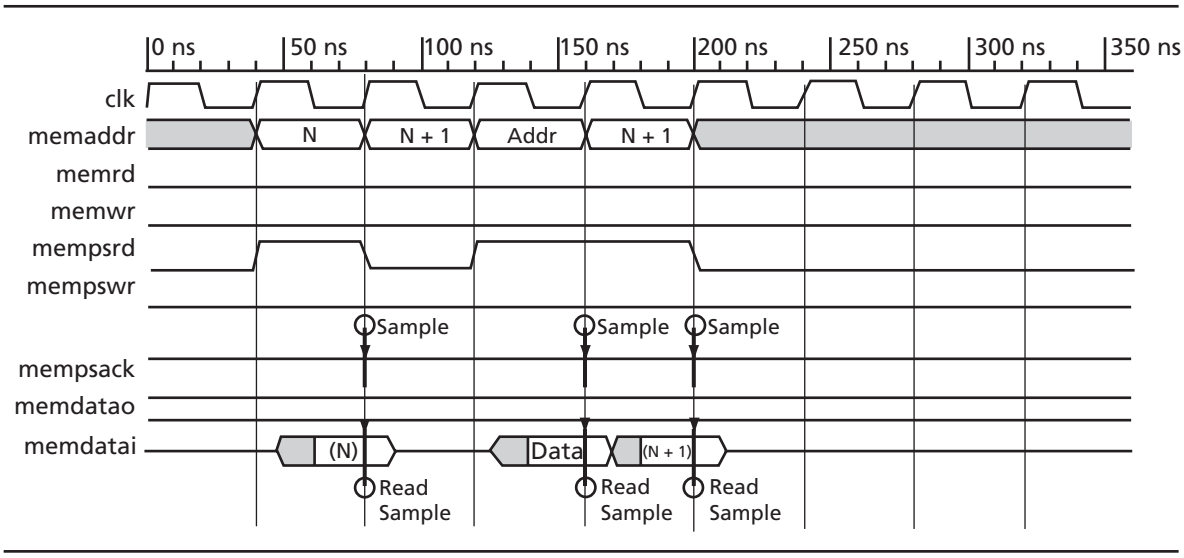

Figure 6-3. Program Memory Read Cycle Without Wait States

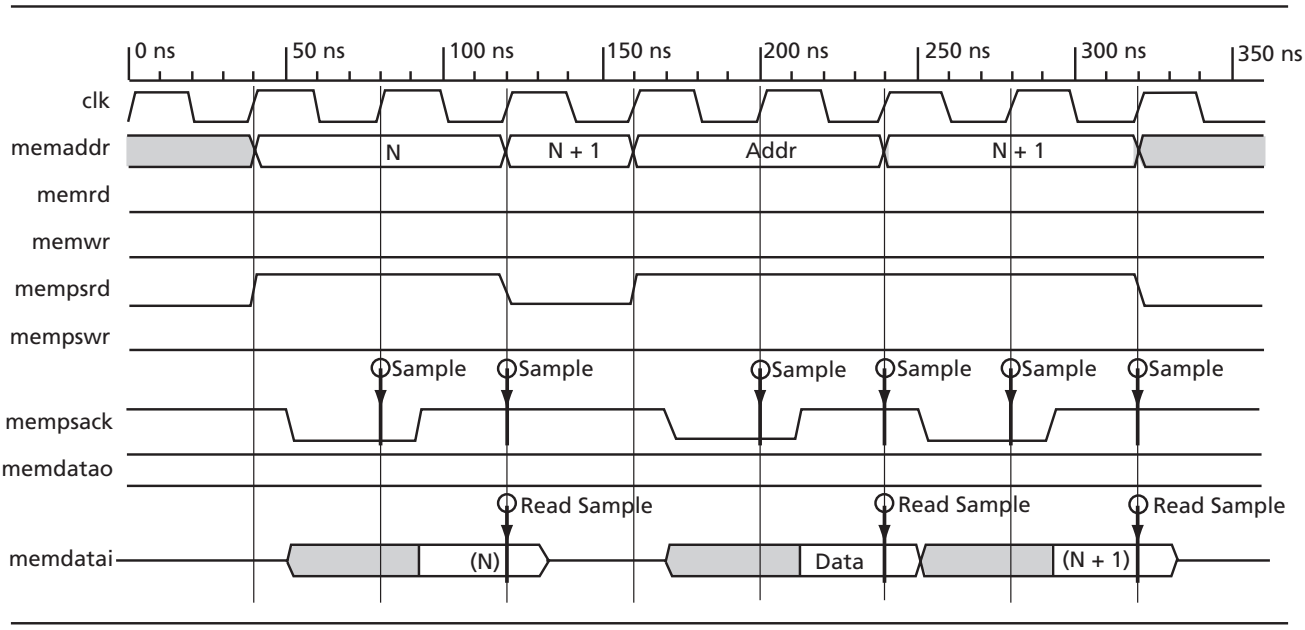

Figure 6-4. Program Memory Read Cycle with Wait States

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 shows an external data memory read cycle without stretch cycles.

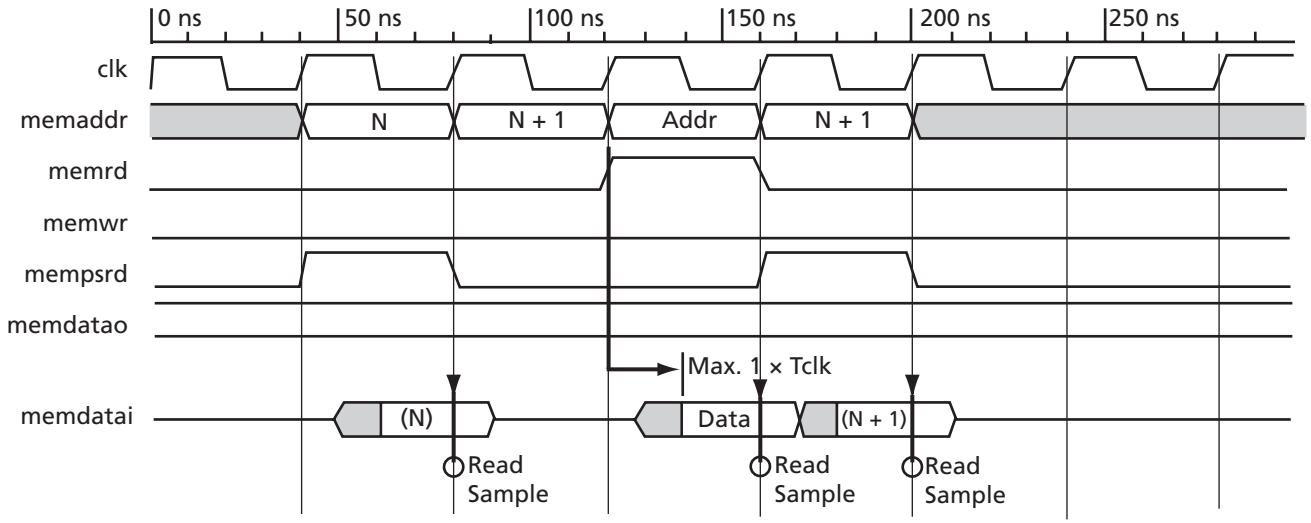

Figure 6-5. External Data Memory Read Cycle Without Stretch Cycles

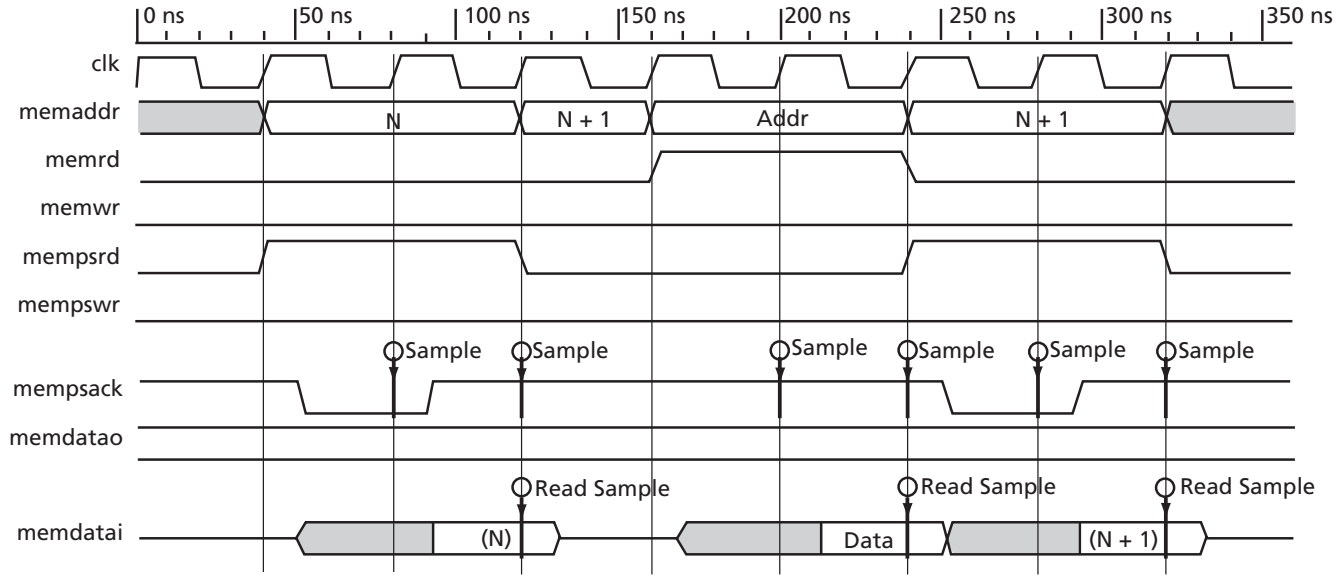

Figure 6-6. External Data Memory Read Cycle With One Stretch Cycle

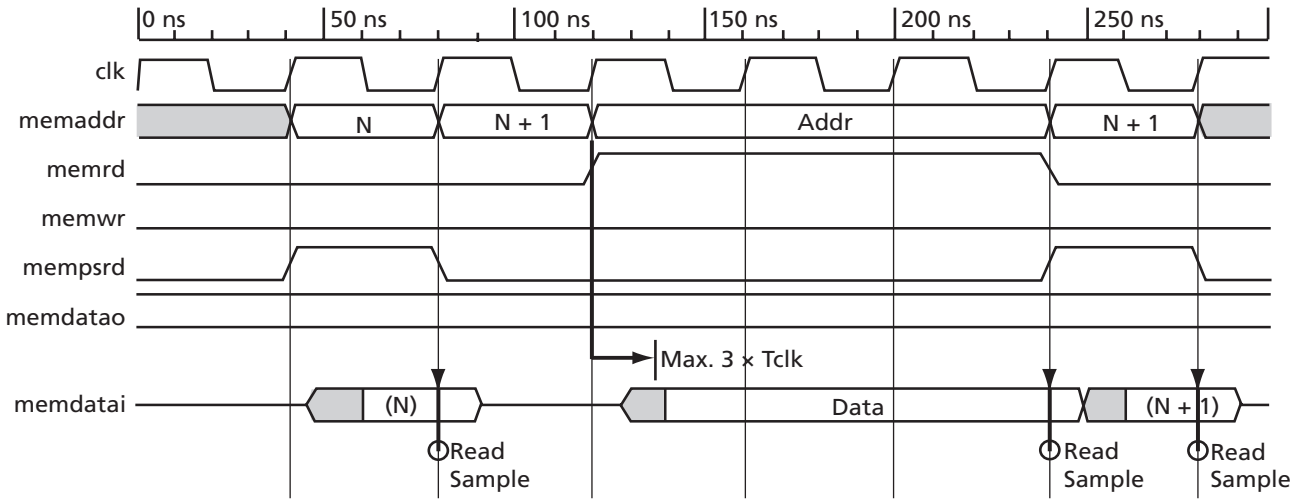

Figure 6-7. External Data Memory Read With Two Stretch Cycles

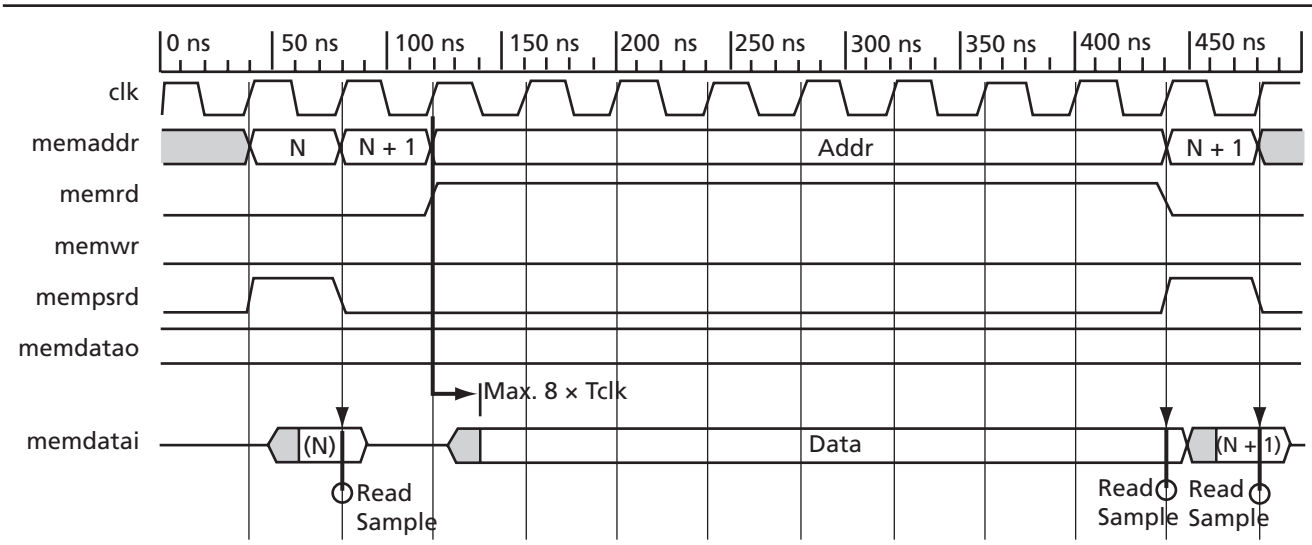

Figure 6-8. External Data Memory Read Cycle With Seven Stretch Cycles

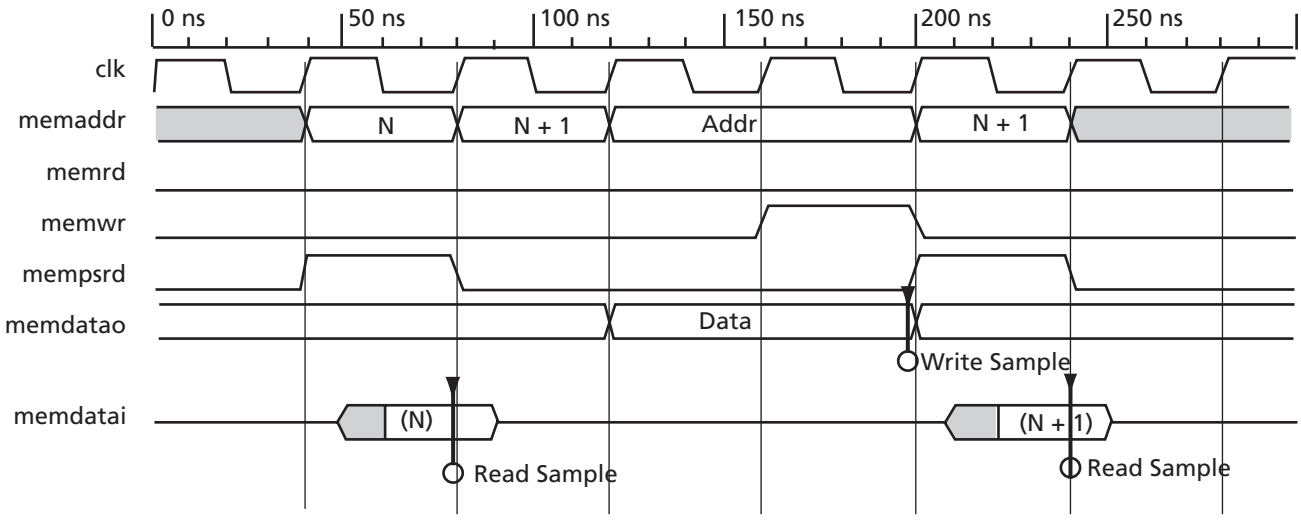

Figure 6-9. External Data Memory Write Cycle Without Stretch Cycles

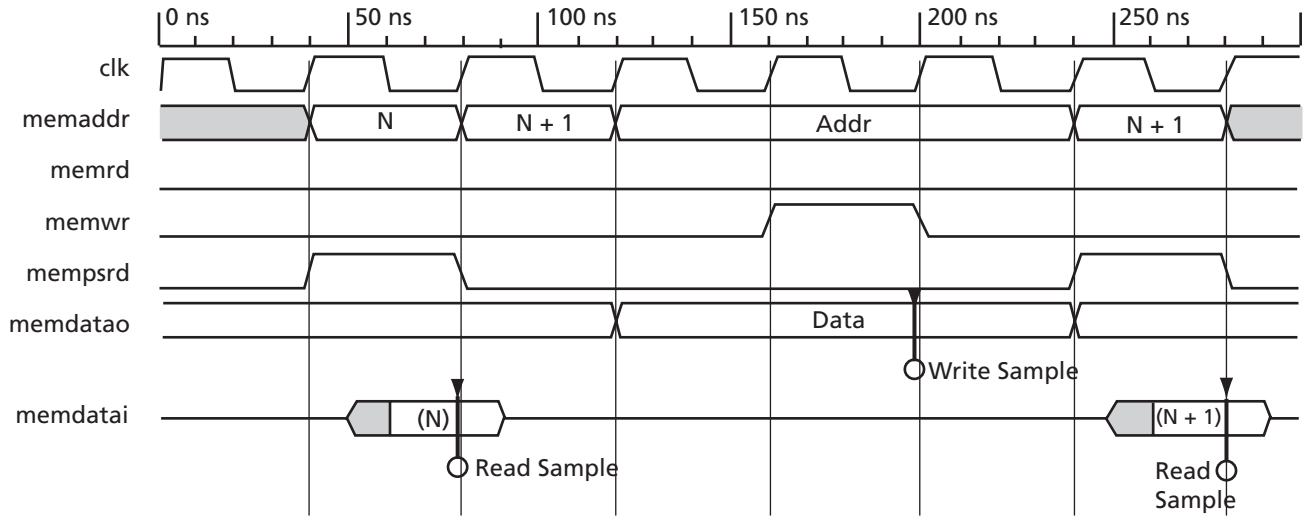

Figure 6-10. External Data Memory Write Cycle With One Stretch Cycle

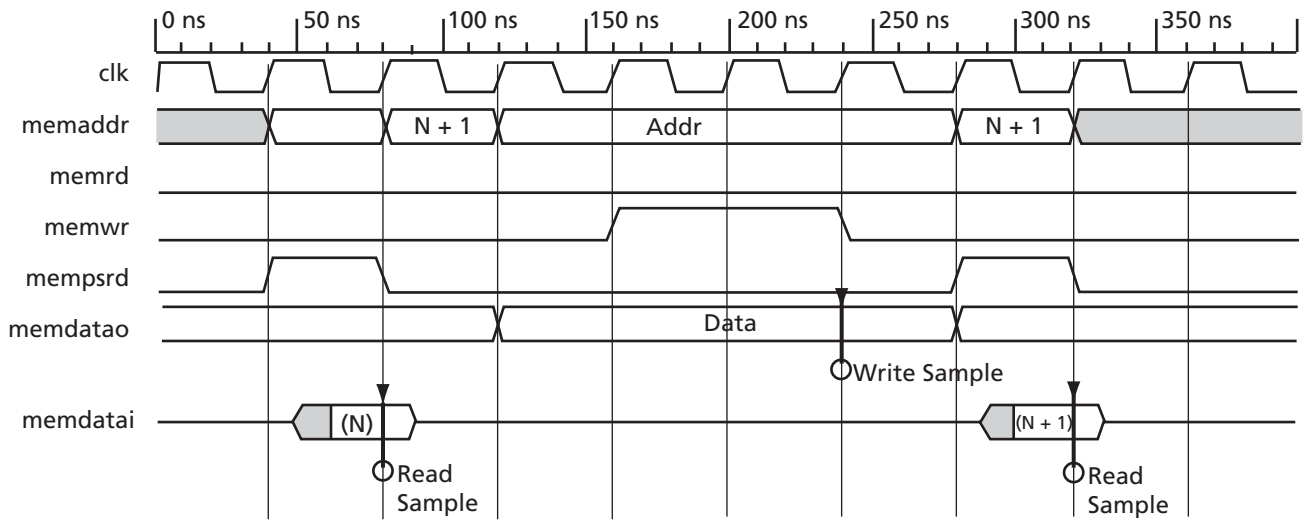

Figure 6-11. External Data Memory Write Cycle With Two Stretch Cycles

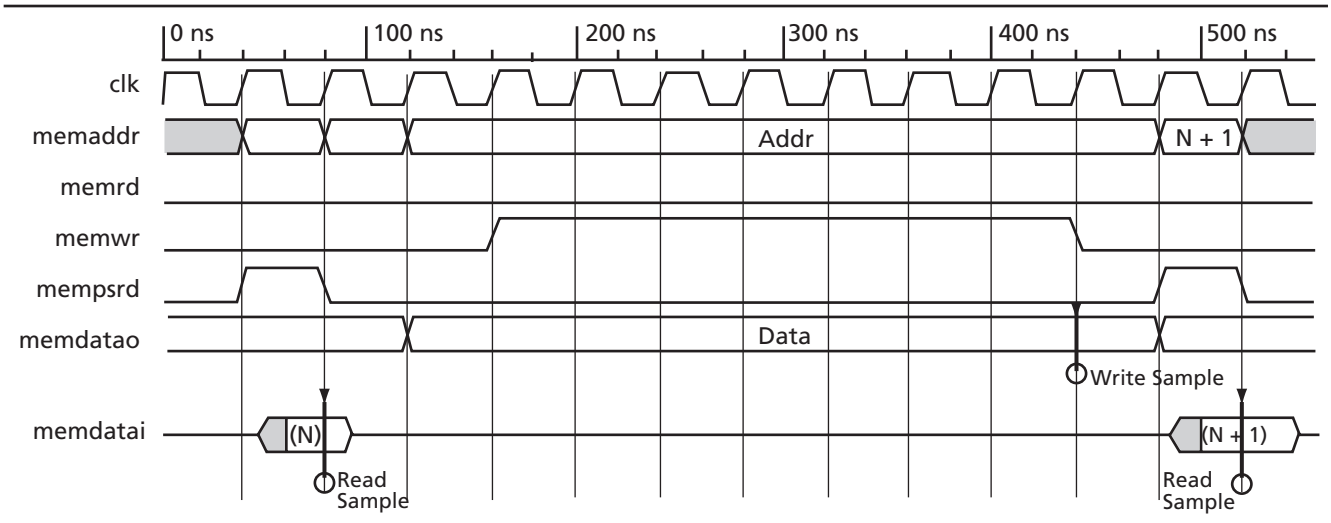

Figure 6-12. External Data Memory Write Cycle With Seven Stretch Cycles

APB Bus Cycles

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

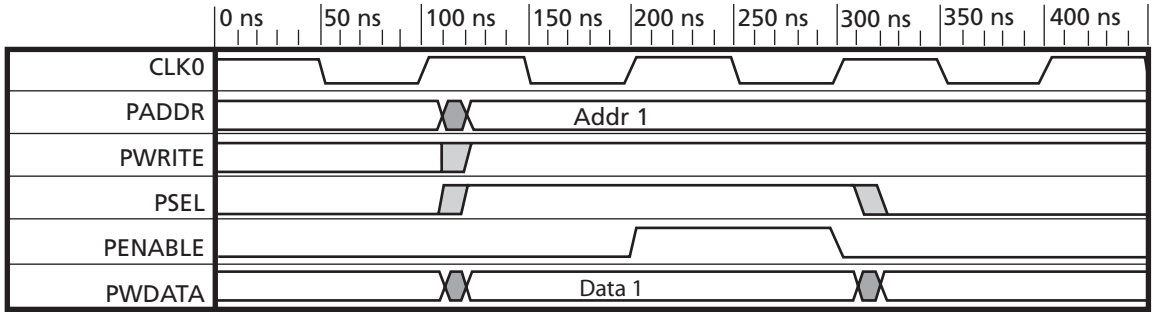

Figure 6-13. APB Write Transfer Bus Cycle

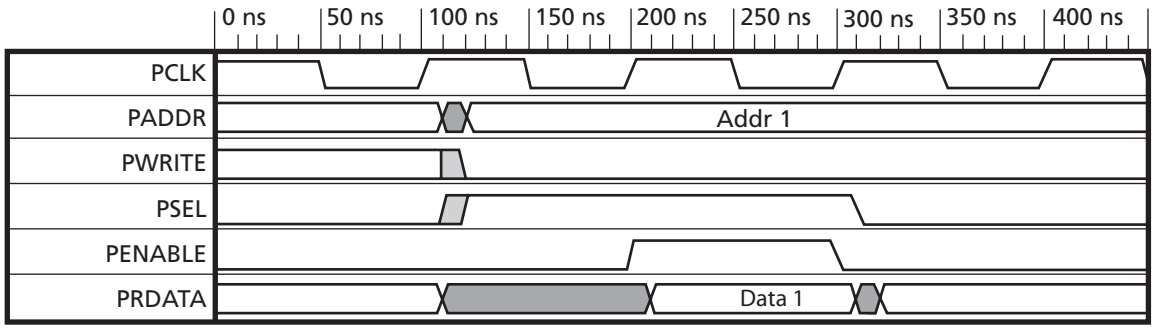

Figure 6-14. APB Read Transfer Bus Cycle

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/custsup/search.html) 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 at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

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 Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200084-0 /11.06